UNIVERSITAT OBERTA DE CATALUNYA

Portal Liferay – Intranet Ajuntament de Vinaròs

Enginyeria Informàtica

Autor: Jordi Tolosà Bel Tutor: Oscar Escudero Sanchez

12 de gener de 2015

Aquest treball està subjecte a una llicència de Reconeixement-CompartirIgual 3.0 No adaptada de Creative Commons. La llicència completa es pot consultar en http://creativecommons.org/licenses/by-sa/3.0/deed.ca.

Índex

Índex de figures

1 Introducció

1.1 Descripció del projecte final de carrera

El projecte a desenvolupar consisteix en crear una intranet corporativa per als empleats de l'ajuntament de Vinaròs.

L'ajuntament durant aquests últims anys ha esdevingut una modernització de les infraestructures de xarxa i dels servidors, fent que l'àrea de sistemes estigui adequada a les necessitats actuals. A conseqüència d'això s'ha deixat un poc de banda més la part de gestió, sobretot de les aplicacions, ja que s'han anat instal·lant moltes aplicacions en poc temps. Les aplicacions d'avui en dia cada cop estan enfocades al model web i la dispersió és evident, i a conseqüència d'això els usuaris perden molt de temps buscant les coses. Per tant l'usuari necessita un lloc centralitzat on disposi de totes les aplicacions i eines per tal de millorar l'eficiència on pugui realitzar la seva feina més còmodament. Per aquest motiu neix la idea de començar el projecte d'una intranet corporativa, on un usuari s'autenticarà a la intranet i li mostrarà el conjunt d'eines i d'aplicacions les quals té accés. Aquesta hauria de convertir-se en una eina bàsica per als usuaris per tal de centralitzar tota la informació que necessiten accedir dia a dia.

Per tant s'ha decidit començar amb dues aplicacions bàsiques per donar-li forma a la intranet. Primer de tot hi haurà una Wiki, on s'organitzarà tota la informació que hi ha dispersa actualment, i és posarà accessible per als usuaris. Aquests hi podran consultar i ser autosuficients per resoldre els seus dubtes. En aquesta s'espera tenir un control de rols i permisos per donar accés a les diferents seccions.

Per una altra banda, hi haurà una aplicació que enllaçarà les aplicacions que té disponible cada usuari, és a dir, vincularà els usuaris que hi ha a l'ajuntament i les aplicacions que utilitza cadascú dels usuaris. Mostrant-li els enllaços disponibles per a que pugui accedir. També ens servirà al departament d'informàtica per poder tenir informació útil de cadascuna de les aplicacions.

Un dels punt més importants a tenir en compte a l'hora de desenvolupar aquesta aplicació és escollir bé les tecnologies a utilitzar per tal que s'integri millor als sistemes que hi ha actualment a l'ajuntament. Per exemple disposem d'una base de dades Oracle, llavors seria interessant usar aquesta base de dades per tal d'aprofitar que tenim configurat el

OracleRAC en alta disponibilitat per tal disposar més fiabilitat en aquest servei. També integrar l'aplicació en el directori actiu, per tal d'aprofitar els usuaris i grups que hi ha al Directori Actiu per fer més fàcil la gestió de rols i permisos a les diferents aplicacions.

Pensant en el desenvolupament, la idea no seria construir una aplicació web des de zero, sinó aprofitar algun dels gestors de contingut que avui en dia hi ha al mercat, per tal d'aprofitar les avantatges que ens ofereixen aquests. L'ajuntament actualment disposa d'una web de turisme desenvolupada amb el CMS Liferay i una altra amb el CMS Joomla. Indagant per la web i buscant informació d'aquestes solucions he comprovat que el CMS Liferay té molt de potencial i que últimament s'està utilitzant per desenvolupar aplicacions al sector públic. A més a més el gestor de continguts volia que uses tecnologia J2EE. Així doncs s'ha optat per usar Liferay, com a punt de partida tindrem un gestor el qual podrà contenir tots els mòduls que es vulguin per a gestionar tota la intranet de l'empresa de forma que es vol aconseguir una infraestructura de mòduls acoblats a Liferay.

Liferay és una solució de codi lliure i obert escrit amb java, que s'integra amb moltes bases de dades, també disposa d'autenticació LDAP i és poden desenvolupar aplicacions mitjançat tecnologies J2EE. Els portals executen simultàniament múltiples porlets els quals són unitats funcionals amb les que els portals es construeixen. El Liferay incorpora ja uns quans porlets predefinits, com per exemple la Wiki. Però també hi ha l'opció de poder crear porlets a mida.

Per a desenvolupar aquests porlets, es requereix fer un estudi del patró Model-Vista-Controlador (MVC) que s'haurà d'usar, el qual facilita una capa de controladors seguint el disseny d'aquest patró, facilitant la integració amb altres eines per a la implementació de les capes de presentació i de negoci. Aquest patró separa el model de negoci de les vistes i del Controlador.

Un cop escollides ja les tecnologies a utilitzar començarà la part més pràctica del projecte, on s'haurà de definir, dissenyar i implementar l'aplicació. On es definirà també l'arquitectura utilitzada i l'entorn de treball on es vol començar ja a desenvolupar el projecte.

Per últim es començarà a implementar el projecte per tal d'arribar finalment a l'objectiu esmentat, aportar un prototipus que pugui ser utilitzat a un entorn de producció en usuaris reals.

1.2 Objectius generals i específics

Com a objectiu principal és d'endinsar-se en el mon de la programació J2EE a nivell de CMS. Ja que la combinació d'un CMS i la tecnologia J2EE aporta molt de potencial a qualsevol aplicació web, a més de les possibles integracions en altres tecnologies. Encara que soc conscient que al haver tantes tecnologies implicades haurà d'haver una fase d'anàlisis i investigació molt profunda, ja que no tinc experiència en aquest tipus de solucions. Per tant hi haurà una fase inicial prou intensa d'aprenentatge que hem servirà per conèixer amb més detall tots els components implicats en el projecte.

I sobretot m'ajudarà tenir la motivació extra de que aquesta aplicació s'usarà en un entorn de producció per tal de millorar deficiències existents de comunicació.

Més específicament, destacarem els aspectes més importants que serveixen per enunciar aquests objectius específics:

- ✓ Muntar un entorn de desenvolupament integrant el CMS Liferay en Oracle i en el Directori Actiu de l'ajuntament.
- ✓ Fer un estudi de les tecnologies que intervenen a J2EE i que s'usaran al projecte: JSP (Java Server Pages), EJB (Enterprise Java Beans), Servlets, XML, etc ..
- ✓ Analitzar el CMS Liferay i intentar aprofitar les seves característiques.
- ✓ S'analitzarà el Model Vista Controlador i s'usarà per dissenyar i implementar el nostre porlet.
- ✓ Implementació d'un porlet usant frameworks J2EE i patrons de disseny analitzats.

1.3 Pla de treball

Tot seguit es mostra tota la planificació del projecte final de carrera. Primer de tot mostrem el calendari de desenvolupament, marcat per les dates d'entrega de les pac. I després és mostra la planificació temporal i el diagrama de Gantt.

1.3.1 Calendari de treball

Aquest apartat mostrem el calendari de desenvolupament basant-nos en les diferents entregues del projecte

PAC1 (1/10/2014)

Aquesta entrega es basarà únicament amb l'aportació del document actual, per tant la memòria en aquest punt no s'haurà iniciat, però val a dir que es començarà a realitzar un cop realitzada l'entrega i s'anirà actualitzant durant tot el projecte fins l'entrega final.

Els objectius de l'entrega són:

- ✓ Fer una descripció del projecte a realitzar
- ✓ Descriure els objectius generals i específics
- ✓ Definir un pla de treball amb el calendari de desenvolupament i la planificació temporal

PAC2 (5/11/2014)

Aquest apartat inclourà la part d'anàlisis i disseny del projecte.

La qual podríem incloure els següents punts:

- ✓ Fer una anàlisi de l'arquitectura Liferay
- ✓ Fer un anàlisis dels frameworks i patrons de disseny J2EE
- ✓ Definir els requeriments funcionals
- ✓ En el disseny de l'aplicatiu inclourà tot lo referent als diagrames UML: diagrama de casos d'ús, diagrama de classes, diagrama de components, etc...

PAC3 (19/12/2014)

En aquesta última PAC ens centrarem sobretot en la part d'implementació del projecte. Serà la part més pràctica, on és muntarà l'entorn de producció, l'entorn de treball, i es començarà a implementar l'aplicatiu en les tecnologies escollides.

- ✓ Muntar entorn de producció
- ✓ Muntar entorn de treball
- ✓ Desplegar la Wiki al Liferay
- ✓ Desenvolupar porlet aplicacions-usuaris
- ✓ Integració Liferay en Oracle i LDAP
- ✓ Testejat i correcció d'errors

Entrega final , memòria, presentació i prototipus (12/01/2015)

Finalment haurem de realitzar l'entrega final, la qual constarà de:

- ✓ Una memòria final del projecte
- ✓ Una presentació del projecte
- ✓ El prototipus final de l'aplicació

1.3.2 Planificació temporal

En el següent diagrama de Gannt podem observar com queda la planificació temporal del projecte amb les seves fites i entregues.

	Nombre	Duración	Inicio	Terminado
1	**Definició del Projecte**	11 days?	**17/09/14 8:00**	**1/10/14 17:00**
2	Introducció i propòsits	4 days?	17/09/14 8:00	22/09/14 17:00
3	Objectius	2 days	23/09/14 8:00	24/09/14 17:00
4	Planificació	4 days?	25/09/14 8:00	30/09/14 17:00
5	Entrega PAC 1	1 day?	1/10/14 8:00	1/10/14 17:00
6	**Anàlisis i Disseny**	25 days?	**2/10/14 8:00**	**5/11/14 17:00**
7	Anàlisis arquitectura Liferay	3 days?	2/10/14 8:00	6/10/14 17:00
8	Anàlisis Frameworks i Patrons de Disseny	5 days?	8/10/14 8:00	14/10/14 17:00
9	Anàlisis funcional	3 days?	15/10/14 8:00	17/10/14 17:00
10	Diseny de l'arquitectura	12 days?	18/10/14 8:00	4/11/14 17:00
11	Entrega PAC 2	1 day?	5/11/14 8:00	5/11/14 17:00
12	**Implementació i Proves**	32 days?	**6/11/14 8:00**	**19/12/14 17:00**
13	Muntar entorn de treball	2 days?	6/11/14 8:00	7/11/14 17:00
14	Desplegar la Wiki al Liferay	2 days?	8/11/14 8:00	11/11/14 17:00
15	Implementar Portlet aplicacions-usuaris	23 days?	12/11/14 8:00	12/12/14 17:00
16	Muntar entorn de producció	1 day?	13/12/14 8:00	15/12/14 17:00
17	Integració Liferay Oracle i LDAP	1 day?	16/12/14 8:00	16/12/14 17:00
18	Testejat i correcció d'errors	2 days?	17/12/14 8:00	18/12/14 17:00
19	Engrega PAC 3	1 day?	19/12/14 8:00	19/12/14 17:00
20	**Documentació**	84 days?	**17/09/14 8:00**	**12/01/15 17:00**
21	Memòria	83 days?	17/09/14 8:00	9/01/15 17:00
22	Presentació	11 days?	26/12/14 8:00	9/01/15 17:00
23	Entrega final	1 day?	12/01/15 8:00	12/01/15 17:00

Figura 1: Diagrama de Gannt

2 Anàlisis frameworks i patrons de disseny J2EE

2.1 Patrons de disseny J2EE

2.1.1 Introducció

Un patró és una solució a un problema de disseny no trivial que sigui efectiva i reutilitzable. Efectiva perquè ha servit per resoldre el problema de dissenys anteriors i reutilitzable ja que la solució és la mateixa per problemes semblants.

Els patrons a grans trets pretenen:

- ✓ Proporcionar catàlegs d'elements reutilitzables en el disseny de sistemes software.
- ✓ Evitar la reiteració en la cerca de solucions a problemes ja coneguts i solucionats anteriorment.
- ✓ Estandarditzar el mode en que es realitza el disseny.
- ✓ Formalitzar un vocabulari comú entre dissenyadors.
- ✓ Facilitar l'aprenentatge de les noves generacions de dissenyadors condensat el coneixement ja existent.

Pel que fa als patrons J2EE, estan orientats específicament als problemes comuns de totes les aplicacions J2EE. Alguns estan basats en els patrons originals i per altra banda uns altres són més específics del tipus de problemes que sorgeixen específicament en J2EE, ja sigui pels tipus d'aplicacions que es solen desenvolupar en la plataforma o per les característiques de la tecnologia.

La plataforma J2EE ens ofereix possibilitats de disseny. L'elecció d'una arquitectura o una altra condicionarà el nostre disseny. No obstant això, la utilització del patró arquitectònic MVC i els Patrons de Disseny J2EE, dotaran a les nostres aplicacions de gran escalabilitat facilitant possibles futures ampliacions, migracions i canvis en general.

Segons els seu nivell d'abstracció els podem classificar en aquestes categories:

- ✓ **Patrons arquitectònics:** aconsellen l'arquitectura general que ha de seguir una aplicació, per exemple el patró Model-Vista-Control (MVC) aconsella l'arquitectura global que ha de tenir una aplicació interactiva.

- ✓ **Patrons de disseny:** expliquen com resoldre un problema concret de disseny, per exemple el patró Data Access Object (DAO) permet abstraure i encapsular els accessos a un repositori de dades (BD relacional, BD orientada a objectes, etc ...).

El disseny d'aplicacions web basades en els Patrons J2EE s'organitza al voltant de d'utilització de diversos elements: un controlador frontal, dispatchers, vistes compostes, vistes (*JSPs*) i els *helpers* de les vistes (*JavaBeans*).

La següent imatge mostra una representació gràfica del Catàleg de Patrons Principals de J2EE (Core J2EE Patterns):

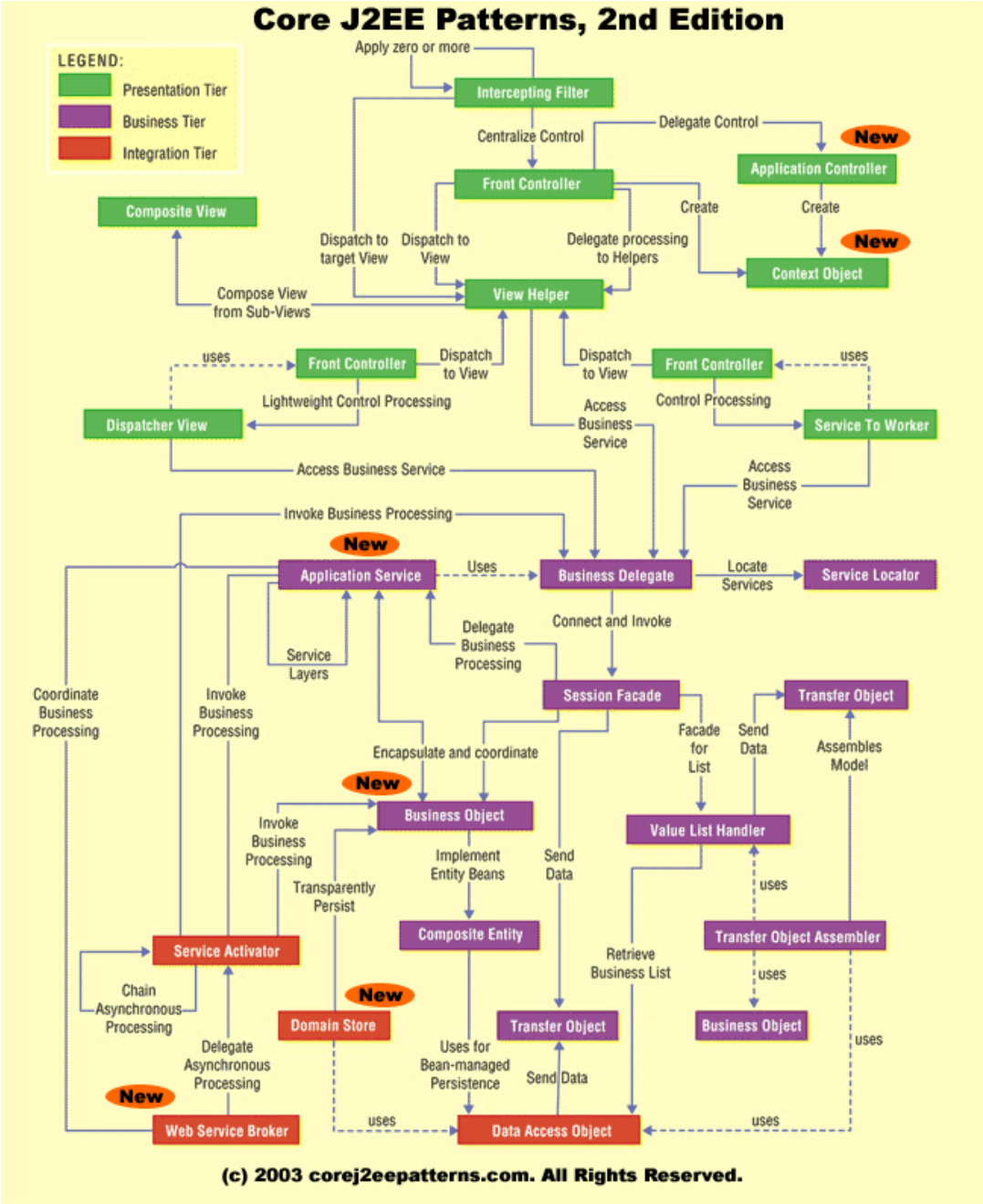


Figura 2: Catàleg principal de patrons J2EE

2.1.2 Tipus de patrons

Com ja hem comentat existeix multitud de patrons que proporcionen solucions a diferents problemes. Tot seguit es presentaran només alguns d'ells.

- ✓ **Intercepting Filter**

La solució que proporciona aquest patró és l'ús de filtres en cadena per processar serveis comuns interceptant les peticions entrants i les respostes sortints, permetent un pre i post-processament, podent afegir i eliminar aquests filtres a discreció sense necessitar canvis en el codi existent. Aquests filtres poden ser per exemple de registre, autenticació, validació de sessió, etc..

Aquest patró està relacionat amb el *Front Controller* i resolen problemes similars. També està centrat en el filtrat de peticions, en canvi, el *Front Controller* tria l'acció corresponent al rebre una petició, això podria realitzar un filtrat si la petició no arriba convenientment filtrada.

.

- ✓ **Front Controller**

Un controlador controla les peticions, incloent la invocació dels serveis de seguretat com autenticació i autorització, delega el processament de negoci, controla l'elecció de la vista adequada, controla errors, i selecciona les estratègies de creació de contingut. També crea un punt d'accés centralitzat, ja que rep totes les peticions entrants remeten al mateix temps, només si es necessari, cada petició al gestor de peticions (*Dispatcher*) adequat que s'encarregarà de gestionar la construcció de les respostes.

Utilitzant aquest patró aconseguim centralitzar el control aconseguint reduir l'acoblament entre la lògica de negoci i vista, aplicant directament el model MVC. Existeixen diverses estratègies per la implementació d'aquest patró. La utilització de pàgines JSP com a controlador, o la utilització de *servlets* com a controlador. També es recomana l'estratègia d'utilitzar un *servlet* com a controlar, aquest *servlet* faria ús de les classes d'ajuda (Helpers) en els que el controlador delegaria les responsabilitats.

Un exemple de La implementació típica del controlador normalment és un *servlet* que estén a *HttpServlet*, en el cas de les aplicacions Web.

- ✓ **View Helper**

Encapsula lògica corresponent a la presentació i l'accés a dades. Normalment són *JavaBeans* i eviten l'ús de *scriptlets* en la vista. L'ús d'etiquetes personalitzades o estàndards es podria considerar implementacions d'aquest patró amb les estratègies *JavaBeans Helper* i *JSP View* respectivament.

Quan s'utilitzen les fulles d'estil per a donar format a la vista també s'està fent ús d'aquest patró.

- ✓ **Composite View**

Gestiona els diferents elements de la vista mitjançant una plantilla que realitza la representació de les vistes més gestionables. Amb aquest patró s'intenta utilitzar vistes compostes que estan formades per múltiples subvistes.

L'abús d'aquestes subdivisions pot tenir un efecte negatiu en el rendiment per lo que es convenient assolir a un compromís entre modularitat i manteniment front a rendiment.

- ✓ **Dispatcher View**

Aquest patró és en realitat la solució d'utilitzar conjuntament els patrons de *Front Controler, Dispatcher* (gestor de peticions) i *View Helper*.

- ✓ **Session Facade**

És una aplicació del patró *Façade* en l'entorn de sessions. El patró *Façade* substitueix les interfícies d'una sèrie de classes sota una sola interfície i llavors proveís un accés unificat a un subsistema d'una aplicació.

Els sistemes s'estructuren en subsistemes formats per patrons i classes que els implementen, amb dependencies entre ells. Amb la utilització d'aquest patró s'intenta evitar que un client, al accedir un sistema (o subsistema) necessiti accedir a més d'una classe provocant dependència a cadascuna d'elles.

Aquest patró representa una cap controlador entre els client i la capa de negoci, abstrau la complexitat d'aquesta última i li presenta al client una interfície senzilla reduint

l'acoblament i dependència entre ells. Encapsula la complexitat de les interaccions entre els objectes exposant només les interfícies requerides i proporcionant un accés uniforme als clients.

Es tracta d'una capa senzilla en el model que dona suport directe per implementar els casos d'us i ocultar la tecnologia d'accés a dades realitzant les funcions de controlador dins del model.

Les capes inferiors no tenen coneixement de la capa Façade. Aquest punt d'accés a les capes inferiors del model proporciona un desacoblament que facilita la possibilita de canvis en les capes inferiors del model sense que els clients es vegin afectats.

- ✓ **Data Acces Object**

Aquest patró és molt utilitat en el context JDBC. Abstrau i encapsula tots els accessos a la font de dades aïllant els objectes de negoci d'una implementació particular de persistència. L'objecte DAO gestiona la connexió amb la font de dades per obtenir i modificar dades implementant els mecanismes d'accés requerits per a treballar amb la font de dades.

Per accedir a dades residents en bases de dades relacionals, l'API JDBC proporciona accés i manipulació de dades utilitzant sentències SQL. No obstant això, inclús en aquests entorns de base de dades de relacionals, l'actual sintaxis i format de les sentències SQL poden variar depenent de la bases de dades en particular.

Però les diferències són majors quan s'utilitzen diferents mecanismes per a l'emmagatzemament de dades. Per evitar dependències directes del codi amb els mecanismes d'emmagatzemament de dades és especialment útil l'aplicació d'aquest patró de disseny.

Aquest patró, a l'evitar dependències, fa molt menys complicada la migració de l'aplicació d'un tipus de font de dades a l'altre. Permetrà una fàcil configuració per a que components *EJB, JSP* o *servlets* treballin amb la informació ubicada en sistemes *B2B, LDAP*, tipus bases de dades relacionals (*RDBMS*), bases de dades orientades a objectes (*OODBMS*), documents *XML*, etc.. Això podrà ser així fins i tot quan les *APIs* per a l'emmagatzemament canviïn segons el venedor del producte o bé siguin *APIs* no estàndards. L'ús d'un objecte d'accés a dades (DAO), proporciona una solució front a la diversitat d'emmagatzemament persistent abstraient i encapsulant tot l'accés a la font de

dades. Aquest objecte control i gestiona la connexió amb la font per emmagatzemar i obtenir dades. Aquest patró d'accés a dades proporciona una capa que oculta completament possibles canvis en la implementació sense afectar als clients o components de negoci. Actua com un adaptador entre un component i la font de dades, permetent un accés transparent a la font de dades.

2.2 Model Vista Control (MVC)

Com hem comentat anteriorment, J2EE es basa en una arquitectura multicapa i aquesta utilitza un patró arquitectònic MVC (Model-View-Controller). Amb aquest model permet una fàcil separació de la interfície gràfica i del model de negocis, gràcies a un controlador que els manté desacoblats. Amb això podríem tenir configuracions en les que el client només disposi de la interfície gràfica i accedeixin a un servidor on s'implementi el model. D'aquesta manera, canvis en el model no es només útil en entorns d'Internet, sinó en tots els entorns empresarials.

2.2.1 Descripció del MVC

El Model Vista Controlador (MVC) és un estil d'arquitectura de software que separa les dades d'una aplicació, la interfície d'usuari, i la lògica de control en tres components diferents. Es tracta d'un model molt madur i que ha demostrat la seva validesa al llarga dels anys en tot tipus d'aplicacions, i sobre multitud de llenguatges i plataformes de desenvolupament.

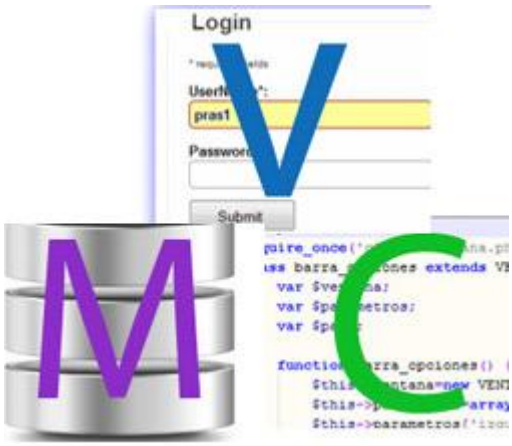


Figura 3: Model Vista Controlador

Tot seguit es fa una breu descripció de les capes i les seves funcionalitats:

- ✓ **Model:** conté una representació de les dades que gestiona el sistema, la seva lògica de negoci, i els seus mecanismes de persistència. El model és el responsable de:
  - Accedir a la capa d'emmagatzemament de dades. L'ideal és que el model sigui independent del sistema d'emmagatzemament.
  - Defineix les regles de negoci (la funcionalitat del sistema).
  - Porta un registre de les visites i controladors del sistema.
  - Si ens trobem davant un model actiu, notificarà a les vistes els canvis que en les dades pugui produir un agent extern.

- ✓ **Vista:** o interfície d'usuari, compon la informació que s'envia al client i els mecanismes d'interacció amb aquest. La vista és la responsable de:
  - Rebre dades del model i les mostra a l'usuari.
  - Té un registre del seu controlador associat.
  - Pot donar el servei d'actualització, per a que sigui invocat pel controlador o pel model.

- ✓ **Controlador:** actua com intermediari entre el Model i la Vista, gestionant el flux d'informació entre ells i les transformacions per adaptar les dades a les necessitats de cadascú. El controlador és el responsable de:
  - Rebre els esdeveniments d'entrada (un clic, un canvi al camp de text, etc...).
  - Conté regles de gestió d'esdeveniments. Aquestes accions poden suposar peticions al model o a les vistes.

2.2.2 Funcionament del MVC en J2EE

En el funcionament de les aplicacions MVC dintre de J2EE les podem diferenciar en els següents tres punts:

- ✓ **Captura de la petició en el controlador**

L'aplicació rep peticions que són centralitzades en el Controlador. Aquest és l'encarregat d'interpretar, a partir de la *URL* de la sol·licitud, el tipus d'operació que hi ha que realitzar. Normalment, això es fa analitzant el valor d'algun paràmetre que s'envia annexat a la *URL* de la petició i que s'utilitza amb aquesta finalitat.

✓ **Processament de la petició**

Un cop que el Controlador determini l'operació a realitzar, procedeix a executar les accions pertinents, invocant als diferents mètodes exposats pel Model.

Depenent de les accions a realitzar (per exemple, una alta d'un usuari en el sistema), el Model necessitarà gestionar les dades enviades pel client en la petició, dades que seran les proporcionades pel controlador. De la mateixa manera, els resultats generats pel Model (per exemple la informació resultant d'una cerca serà entregada directament al controlador).

Per tal de facilitar aquest intercanvi de dades entre el Controlador i el Model i, posteriorment, entre el Controlador i la Vista, les aplicacions MVC solen fer ús de *JavaBeans*. Un *JavaBean* no és més que una classe que encapsula un conjunt de dades amb mètodes de tipus set/get per a proporcionar un accés a aquestos des de l'exterior.

✓ **Generació de respostes**

Els resultats retornats pel Model al Controlador són guardats per aquest en una variable de petició, sessió o aplicació, segons l'abast que han de tenir. A continuació, el Controlador invoca a la pàgina JSP que s'ha d'encarregar de generar la vista corresponent, aquesta pàgina accedirà a la variable d'àmbit on estiguin dipositats els resultats i els utilitzarà per a generar dinàmicament la resposta XHTML que serà enviada al client.

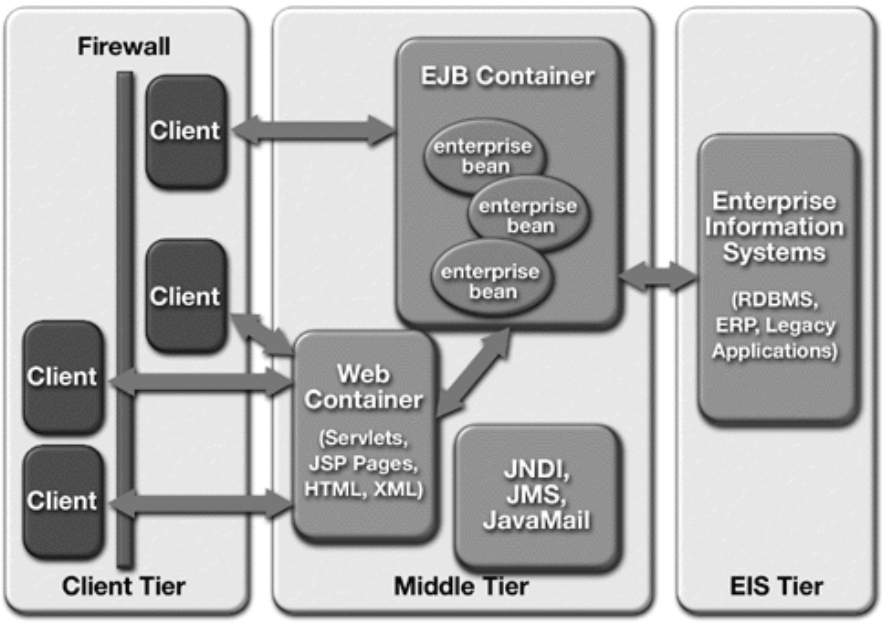


Figura 4: Model Vista Controlador en J2EE

2.3 Frameworks J2EE

Un *framework* és un conjunt de classes i interfícies que cooperen per tal de solucionar un tipus específic de problema de software. Un *framework* té les següents característiques:

- ✓ Consta de múltiples classes o components, cadascú dels quals pot proveir una abstracció d'un determinat concepte.
- ✓ Defineix com aquestes abstraccions treballen juntes per solucionar el problema.
- ✓ Els components del *framework* són reutilitzables.
- ✓ Organitza patrons a alt nivell.

Un bon *framework* deuria de proveir d'un comportament genèric el qual diferents tipus d'aplicacions poder fer ús d'aquest.

Molts dels *frameworks* J2EE ja implementen els mecanismes de comunicació entre capes seguint el MVC.

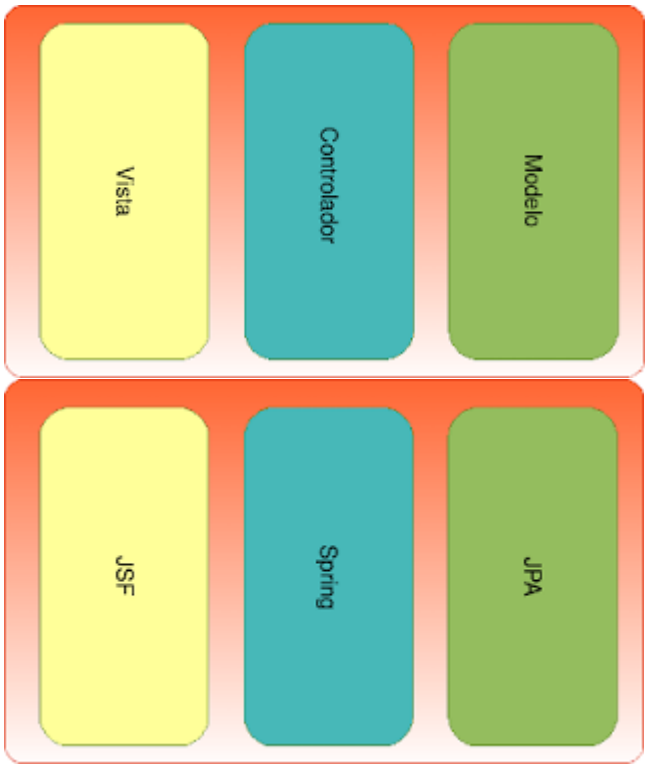


Figura 5: Relació entre Frameworks i les capes del MVC

Usar *JSF* o *Struts* per a la capa Vista, *Spring* per a la capa Controlador, i *JPA* amb *Hibernate* per a la capa Model seria una de les possibles solucions a l'arquitectura d'una aplicació J2EE.

2.3.1 Frameworks de la capa presentació

Existeixen en el mercat diverses implementacions en un ús molt ampli i estès. Seguidament es farà una descripció de les més importants.

- ✓ Struts
- ✓ Struts2
- ✓ Spring
- ✓ JSF

2.3.1.1 Struts

Struts s'encarrega de normalitzar el desenvolupament de la capa de presentació dintre de l'arquitectura MVC. No obstant això, per aconseguir-ho és necessari que també proporcioni mecanismes per treballar amb la capa de components de negoci. Però mai *Struts* es mesclarà amb la capa 3 del model. Aquesta característica principal de *Struts* permet separar la lògica de presentació de la lògica de negoci, amb els avantatges ja comentats anteriorment.

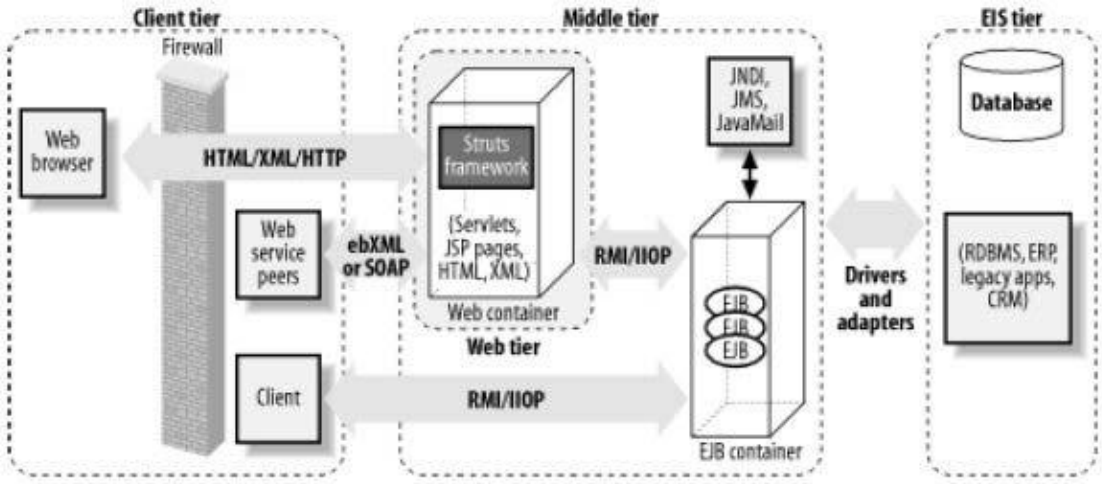


Figura 6: Arquitectura MVC del framework Struts

Struts resideix a la capa de presentació. Les aplicacions de *Struts* s'emmagatzemen en un contenidor web que fa ús de totes les funcionalitat que les subministra, així com la gestió de peticions via *HTTP* o *HTTPS*. Això allibera al desenvolupador que li permet centrar-se en la construcció de l'aplicació en buscar solucions a la lògica de negoci.

Tot seguit és fa una breu descripció de cadascuna de les capes del model:

- **Model**

El model engloba tots els Objectes de Negoci on s'implementa la lògica de negoci i on s'ha de suportar tots els requisits funcionals del sistema, sense mesclar-ho amb parts corresponents al *workflow* les quals corresponen al Controlador.

- **Vista**

La vista està composta per un conjunt de pàgines *JSP*. *Struts* proveeix suport per construir aplicacions multi-idioma, interacció amb formularis i altres utilitats mitjançant la utilització de *Tags* especials (TagLibraries).

- **Controlador**

El Controlador compren la funcionalitat involucrada des de que un usuari genera una petició *HTTP* (click en un enllaç, enviament d'un formulari, etc...) fins que es genera la interfície de resposta. Mentrestant cridarà als objectes de negoci del Model per a que resolguin la funcionalitat pròpia de la lògica de negoci i segons el resultat d'aquesta executar el *JSP* que ha de generar la interfície resultant. *Struts* inclou un *servlet* que a partir de la configuració de *struts-config.xml* rep les sol·licituds de l'usuari, crida al *Action Bean* que correspongui i , segons el que li retorni aquest, executa una *JSP*.

Funcionament de Struts

El següent gràfic mostra la funcionalitat de *Struts.*

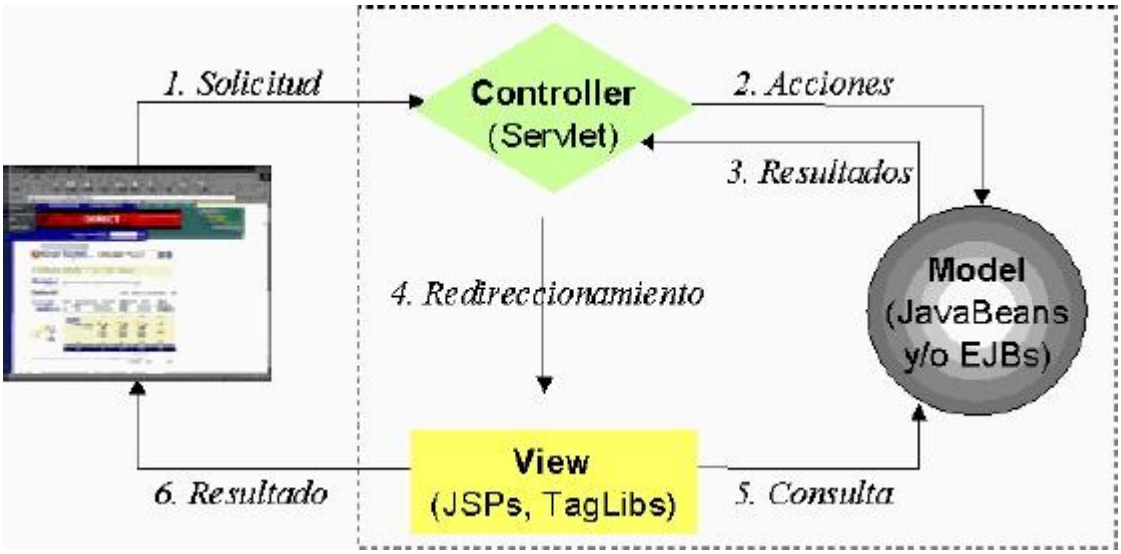


Figura 7: Interacció enre capes del MVC Struts

El navegador genera una sol·licitud que és atesa pel controlador (un *Servlet* especialitzat). Aquest s'encarrega d'analitzar la sol·licitud, seguir la configuració que se li ha programat en el seu XML i cridar al *Action* corresponent passant-li els paràmetres enviats. L'*Action* instanciarà i utilitzarà els objectes de negoci per a concretar la tasca. Segons el resultat que retorni el *Action*, el Controlador derivarà la generació de la interfície a una o més *JSPs*, les qual podran consultar els objectes del Model per mostrar informació d'aquests. Aquest gràfic ens proveeix una visió més detallada del funcionament de *Struts*.

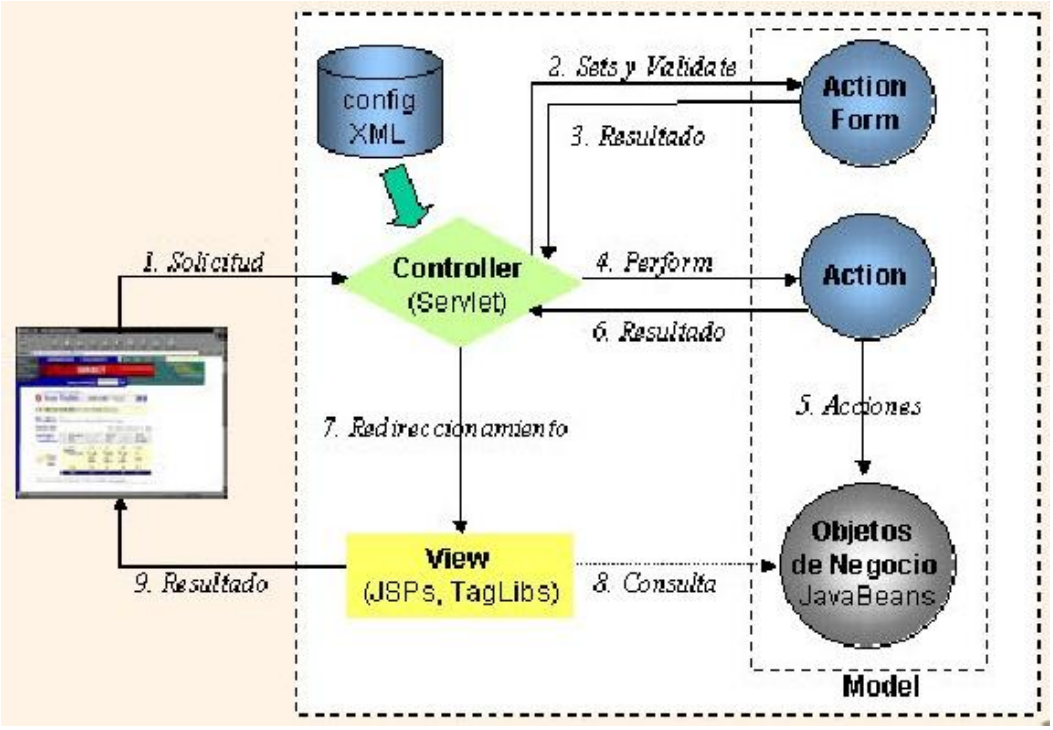


Figura 8: Funcionalitat més detallada de les diferents capes en Struts

Components

- **Model**

Els components del model proporcionen un model de la lògica de negoci darrera de *Struts*. Proporcionen interfícies a la base de dades. Els components del model són generalment classes Java. No hi ha cap format específic pel model, això permet reutilitzar codi ja escrit per altres projectes. Els models normalment s'elegeixen d'acord als requeriments del client.

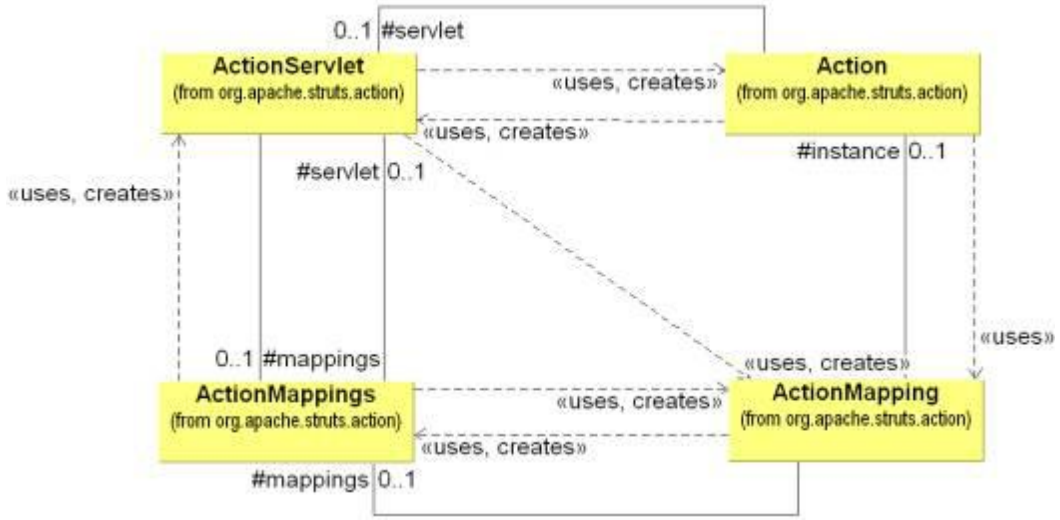


Figura 9: Relacions entre les accions de la capa lógica de negoci de Struts

- **Vista**

Els components de la vista són responsables de presentar la informació als usuaris i acceptar les entrades de informació d'aquestos. Són els responsables de mostrar la informació subministrada pel model. Principalment s'usen *JSP* per la presentació de la vista encara que existeixen altres opcions com usar plantilles de *Velocity*. Per ampliar les capacitats de la vista es poden usar etiquetes, *JavaScript*, etc ...

- **Controlador**

Quan un usuari realitza una petició, aquesta es gestionada pel *Servlet Struts ActionServlet*. Quan el *ActionServlet* rep la petició, intercepta la URL i basant-se en els fitxers de configuració de *Struts*, gestiona la petició mitjançant la classe *Action*. Aquesta classe és part del controlador i és responsable de comunicar-se amb la capa del model. El *struts-config.xml* determina quina classe *Action* ha de ser cridada pel controlador. La configuració del *struts-config.xml* es traduïda en un conjunt de *ActionMapping*, els quals són posats dintre d'un contenidor de *ActionMappings*.

Els *ActionMappings* contenen el coneixement de com un esdeveniment específic mapeija a cadascú dels *Actions*. L'*ActionServlet* passa l'*ActionMapping* a la classe mitjançant el mètode *perform()* o *execute()*. Això permet al *Action* accedir a la informació de control de flux.

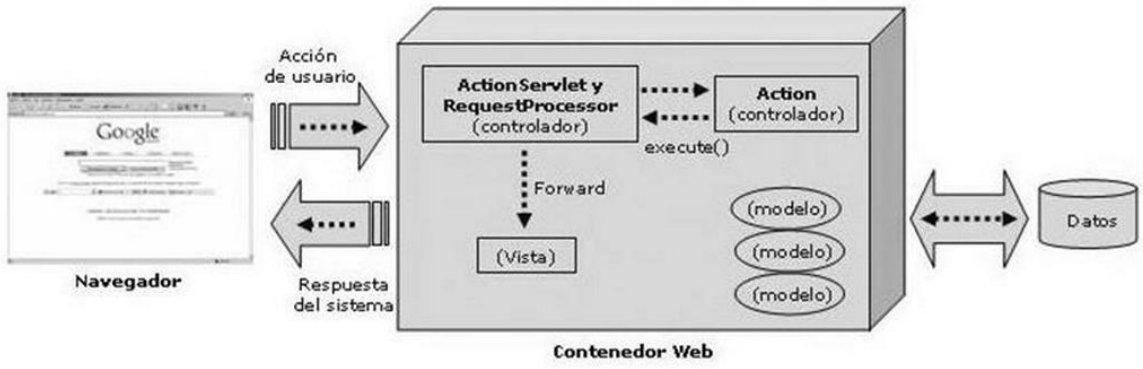


Figura 10: ActionServlet i RequestProcessor dintre del MVC.

2.3.1.2 Struts 2

Amb el pas dels anys, *Struts* s'ha convertit en el *framework* MVC per J2EE per excel·lència. No obstant això, també s'ha quedat un poc antiquat. Com va néixer a un altra època, el disseny original de *Struts* no respecta els principis que avui es consideren bàsics en Java, com per exemple l'ús d'*APIS* "no invasives" que minimitzen les dependències del nostre codi de les APIS del *frameworks*, o la facilitat per fer proves unitàries.

Struts 2 va més enllà d'una simple revisió de Struts1. La relació entre *Struts* 1 i *Struts* 2 és més filosòfica que basada en codi. *Struts* 1 estava orientat a accions que implementa un MVC amb la separació d'interessos en la seva arquitectura. *Struts* 2 és una nova implementació d'aquests principis del *framework* MVC orientat a accions, aquest nou framework conté diferències substancials respecte al seu predecessor.

Per tant podem considerar el *Struts* 2 un *framework* completament nou basat en l'arquitectura obtinguda del *framework WebWork*. El disseny a alt nivell segueix el patró Model-Vista-Controlador. Com ja s'ha comentat, aquest patró permet la separació de responsabilitats que aplica al domini d'una aplicació web. Aquesta separació de responsabilitats permet gestionar la complexitat de grans sistemes de software dividint-les en components d'alt nivell. El patró MVC identifica tres responsabilitats diferents: model, vista i controlador. En *struts* 2 aquestes responsabilitats estan implementades per l'*Action, result i FilterDispatcher* respectivament. La següent figura mostra la implementació del patró MVC que gestiona el flux de l'aplicació web.

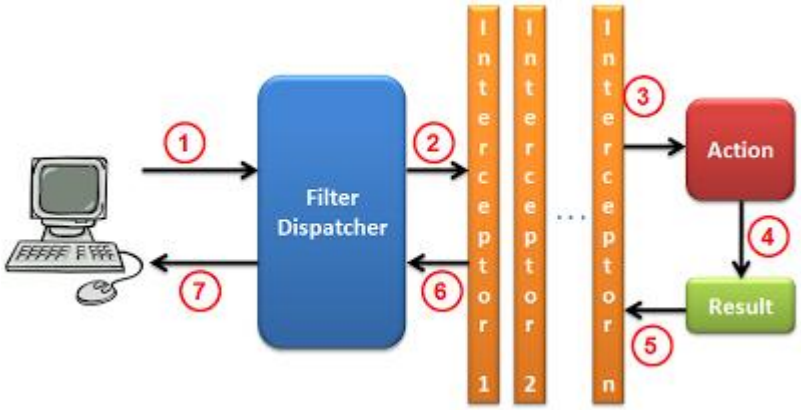


Figura 11: Funcionament bàsic Struts2

A. Inicialment la pàgina realitza la petició *HTTP* al *Filter Dispatcher.*

B. El *Filter Dispatcher* envia les dades als interceptors per a que aquests pugin realitzar les respectives validacions respecte a les dades que enviem.

C. Els interceptors un cop hagin validat la informació enviada invocaran el *Action*.

D. S'invoca el Action respectiu segons la configuració en el *Filter Dispactcher* el qual també s'encarregarà de realitzar la lògica respectiva (processar una informació o retornar alguna dada necessària).

E. Un cop finalitzat el treball del *Action* retornarà la informació requerida al *result* per a que seguidament el *Filter Dispatcher* sàpiga quin serà el següent pas (un *JSP* o un altre *Action*).

F. Els resultats seran validats pels interceptors.

G. Els interceptors envien el resultat al *Filter Dispatcher*.

H. El *Filter Dispatcher* envia el resultat per a que l'usuari ho visualitzi o ho redirigirà a una altra acció**.**

Tot seguit és fa una breu descripció de cadascuna de les capes del model:

- ✓ **Model o *Actions***

En la imatge anterior podem comprovar que en *Struts* 2 el model el realitzen les accions. Una acció de *Struts* 2 té dos papers. Primer és una encapsulació per a les peticions de la lògica de negoci en una unitat simple de treball. En segon lloc

l'acció serveix com lloc de transferència de dades. Es pot dir que una aplicació té un cert nombre d'accions per gestionar qualsevol conjunt de comandes que se li proposen al client. El controlador, després de rebre la petició, ha de consultar entre els seus mapeijos i determinar quina de les accions ha de gestionar la petició. Un cop que es troba l'acció apropiada, el controlador passa el control de la gestió del processador de la petició a l'*Action* invocant-lo. Aquest procés d'invocació dirigit pel *framework*, prepara les dades necessàries i executa la lògica de negoci de l'acció. Quan l'acció finalitza el seu propòsit mostra la vista (resultat).

- ✓ **Vista o *Result***

Un dels aspectes més interessant de *Struts* 2 és com la seva arquitectura clara simplifica el camí a noves tecnologies i tècniques. El component *Result* és una bona mostra d'això. El *Result* proporciona una encapsulació neta de processat de la entrega d'informació de control a un altre objecte que escriurà la resposta al client. Això ho converteix en fàcil per tenir diferents alternatives de resposta, com ara *XML, AJAX* o transformacions *XSLT* i poder ser integrades dintre del *framework*.

L'acció és responsable de l'elecció del quin resultat renderitzarà la resposta. L'acció pot escollir un ampli número de resultats amb els quals està associada. L'elecció més comú serà èxit o error. *Struts* 2 proporciona soport per usar les capes de presentació amb més èxit de l'actualitat com ara *JSP*, *Velocity*, *FreeMarker* i *XSLT*. A més a més la estructura clara de la seva arquitectura assegura que la majoria de tipus de resultats puguin ser fàcilment construïts gestionant els nous tipus de resposta.

- ✓ **Controlador o *FilterDispatcher***

El paper del controlador en *Struts* 2 l'interpreta el *FilterDispatcher*. Aquest objecte tan important és un *servlet* filter que inspecciona cada petició entrant i determina quina acció de *Struts* 2 deuria gestionar la petició. El *framework* gestiona tot el treball del controlador per nosaltres. Només es necessari informar al *framework* de quins mapeijos de peticions de *URLs* es refereixen a quines accions. Això es fa amb un *XML* base de configuració o bé mitjançant anotacions Java.

Funcionament

El *framework* consta de més components que els que simplement proporciona el patró MVC. *Struts* 2 proporciona una implementació més clara de MVC, i aquesta claredat només és possible gràcies a l'ajuda d'altres components arquitectònics clau en el processament de cada una de les peticions. Els principals serien els interceptors, *OGNL* i *ValueStack*. La següent imatge mostra el flux de processament d'una petició.

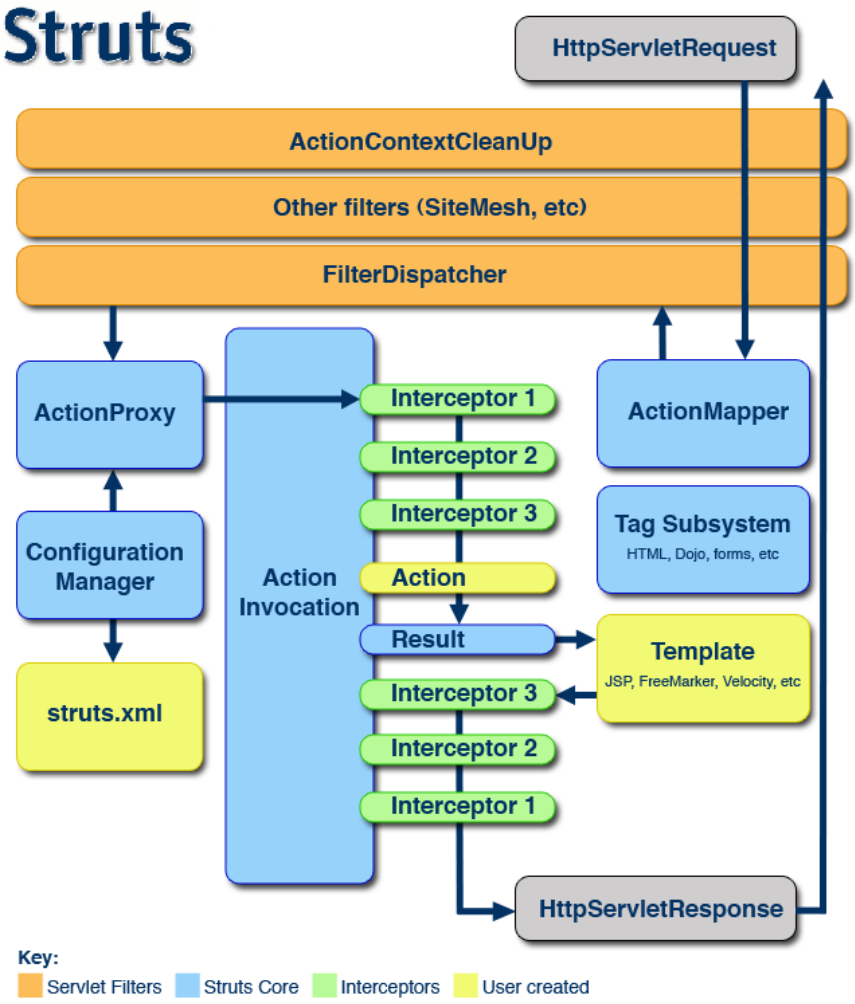


Figura 12: Funcionament detallat dels diferents components que participen a Struts2

Aquesta figura assumeix que el *FilterDispatcher* ha de realitzar el seu treball de controlador. En aquest punt el *framework* ja ha seleccionat una acció i s'ha iniciat el procés d'invocació. La primera cosa a considerar és que el flux de treball d'aquest diagrama encara obeeix la vista simple de MVC. Una acció ha estat seleccionada pel controlador, i s'executarà, i llavors es seleccionarà el resultat apropiat. La resta de diagrama serveix per veure d'una manera més clara el flux bàsic del MVC i veure de forma gràfica quins

elements són els que intervenen dins del processament d'una petició *HTTP*, diferenciant-los quins son elements estàndards de J2EE, quins són elements propis de *Struts* 2 i quins són creats per l'usuari. També és mostren els components de *Struts* 2: interceptors, *ValueStack* i *OGNL* que tot seguit es fa una breu descripció.

Interceptors

Hi ha una pila de interceptors en front de cada acció. Això és una part principal del *framework*, cada acció té una pila d'interceptors associada a ella. Aquests interceptors són invocats abans i després de cada acció. Els interceptors s'activen abans de l'acció i després del resultat. Encara que cal destacar que no és necessari que tinguin alguna cosa que fer en ambdues ocasions, hi ha cops el control simplement passa a través d'aquestos. Alguns interceptors només funcionen abans que l'acció s'hagi executat, i altres només després. Depèn de la funció que se'ls hi hagi assignat. Els interceptors permeten definir tasques comuns i clares que es poden reutilitzar fora de l'àmbit de les accions. Proporcionen un component arquitectònic que defineix diversos flux i tasques per a que puguin ser fàcilment reutilitzades també fora de l'arquitectura principal

La ValueStack i OGNL

Segons s'acaba de veure els interceptors no s'enduen molt de temps de la part de desenvolupament, no obstant això la *ValueStack* i *OGNL* si que es tenen que tenir constantment en compte. La *ValueStack* és simplement l'àrea d'emmagatzemament que manté totes les dades del domini de l'aplicació associades amb el processament de la petició. En definitiva és un lloc on el *framework* fa els seus deures per solucionar els problemes de processament de peticions. En canvi, *OGNL* és un llenguatge que permet referenciar i manipular la *ValueStack*.

Per tant, podem dir que la *ValueStack* és on les dades són emmagatzemades mentrestant que es treballa amb elles, i el OGNL és el llenguatge que el desenvolupador i el *framework* usen per tal d'agafar a aquestes dades des de diverses parts del cicle de processat de peticions.

2.3.1.3 Spring

El primer que s'ha de deixar clar és que *Spring* no és un framework de desenvolupament web. Efectivament, disposa de mòduls per al desenvolupament web com *Spring MVC* i *Spring Web Flow* que estan entre els millor, però per damunt d'això *Spring* és un *framework* per al desenvolupament d'aplicacions empresarials Java, siguin web o no. La idea és que *Spring* s'encarregui de les tasques d'infraestructura per a que els desenvolupadors es centrin en resoldre els problemes del domini.

Spring framework no obliga a usar un model de programació en particular, no obstant això s'ha popularitzat en la comunitat de programadors en Java, esdevenint l'alternativa del model de *Enterprice JavaBean*. El disseny del *framework* ofereix molta llibertat als desenvolupadors en Java i solucions molt ben documentades i fàcils d'usar. Encara que les característiques fonamentals d'aquest *framework* poden utilitzar en qualsevol aplicació fet en Java, existeixen moltes extensions i millores per construir aplicacions basades en web amb la plataforma empresarial J2EE.

Algunes de les principals característiques de *Spring* són:

- ✓ **Flexible**: puc usar qualsevol component o mòdul de *Spring* tenint sempre la possibilitat d'integració amb altes *frameworks* molt populars sense problemes. És a dir, l'eina no ens obliga a usar-lo en tota la seva magnitud sinó que podem utilitzar només el que ens agrada. Per exemple, si utilitzo el seu suport a base de dades no estic obligat a usar el seu MVC i a l'inrevés.
- ✓ **Dèbilment acoblat**: no hem força a utilitzar el *framework* sinó que puc usar els seus projectes com els necessiti.
- ✓ **Altament Cohesiu**: cada mòdul és centra en lo que té que fer de manera dedicada, és a dir, els subprojectes com JDBC són específics per aquesta funció, també per a *webservice* o MVC. És atòmic en les seves funcions, convertint-lo en reutilitzable en del desenvolupament en qualsevol fase.

Spring basa tota la seva filosofia en aquests tres principis:

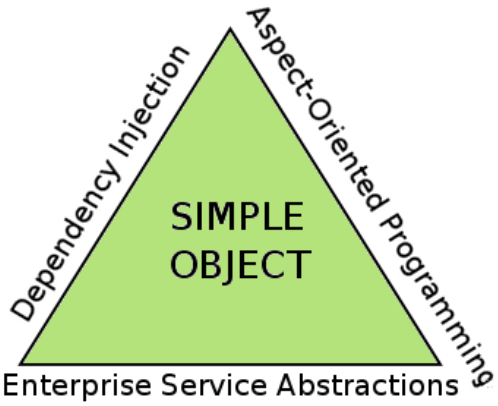


Figura 13: Els tres principis que es basa Spring

✓ **Injecció de dependències (o Inversió de control)**

L'objectiu és aconseguir un baix acoblament entre els objectes de la nostra aplicació. Amb aquest patró de disseny, els objectes no creen o busquen les seves dependències sinó que aquestes són donades a l'objecte. El contenidor és l'encarregat de realitzar aquest treballa al moment d'instanciar l'objecte. S'inverteix la responsabilitat en quan a la forma en que un objecte obté la referència a l'altre objecte.

D'aquesta forma, els objectes coneixen les seves dependències per la seva interfície. Així la dependència pot ser intercanviada per diferents implementacions a través del contenidor. En resum, es programa orientat a interfícies i s'injecten les implementacions a través del contenidor.

✓ **Programació Orientada a Aspectes**

Es tracta d'un paradigma de programació que intenta separar les funcionalitats secundaries de la lògica de negoci. Per exemple, la seguretat, la gestió de transaccions, etc., són funcionalitats que travessen el nostre programa en diverses abstraccions d'aquest. Per tant correm el ris de caure en la repetició del codi i el acoblament entre la nostra lògica de negoci i la implementació de les funcionalitats transversals a les aplicacions.

Al treballar amb AOP (*Aspect-Oriented-Programing* estem centralitzant el codi que implementa estes funcionalitats i configurant *Spring* per a que les executi on vulguem, en lloc de tenir que fer-ho en cada moment que faci ús d'aquestes.

- ✓ **Abstracció de servicis de l'empresa**

Spring s'encarrega de fer transparent moltes tasques que hi ha que realitzar al treballar amb serveis utilitzats habitualment en aplicacions empresarials i que solen resultar molt repetitives. Per exemple al accedir a base de dades mitjançant *JDBC*, llavors *Spring* s'encarrega de gestionar les connexions, unificar excepcions, etc ...

Els mòduls que integren *Spring* poden treballar independentment uns dels altres. Spring està format més o menys per uns 20 mòduls on tot seguit resumirem els més importants.

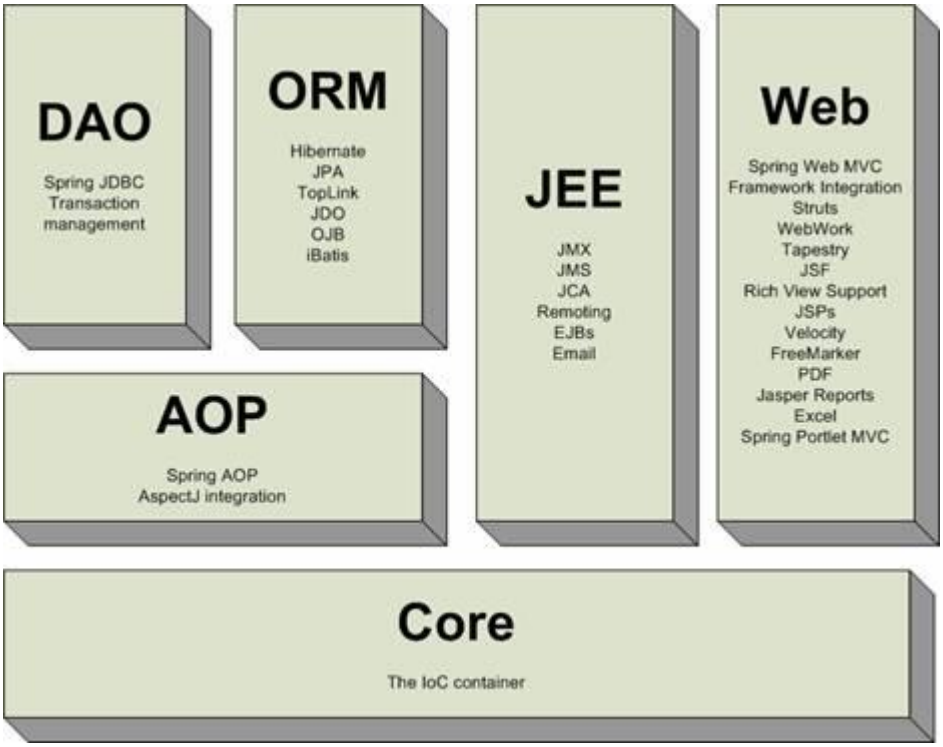


Figura 14: Mòduls que integren el framework Spring

La part fonamental de *Spring*, és el seu mòdul *Core*, el qual s'encarrega de la Injecció de dependencies. El concepte d'aquest mòdul és el *BeanFactory*, que implementa el patró de disseny *Factory*. Dalt del *Core*, està el mòdul Context que s'encarrega de brindar les eines per accedir als *beans*.

- ✓ **DAO:** Proporciona una capa d'abstracció sobre *JDBC*, abstreu el codi d'accés a dades d'una manera simple. Compta amb una capa d'excepcions sobre els missatges d'errors proporcionats per cada servidor específic de base de dades.

- ✓ **AOP:** Aquesta capa desacobla el codi d'una manera neta, implementant funcionalitats que per lògica i claredat deurien estar separades. Usant metadades a nivell de codi font es poden incorporar diversos tipus de informació i comportament al codi.
- ✓ **ORM:** Proveeix capes d'integració per APIs de mapeig objecte – relacional, incloent, *JDO, Hibernate i myBatis*. Usant el paquet ORM és possible usar aquests mapejadors conjuntament amb altres característiques que *Spring* ofereix.
- ✓ **WEB:** Proporciona característiques bàsiques d'integració orientades a la web, com ara inicialització de contexts mitjançant *servlet listeners* o un context d'aplicacions orientat a la web. Quan s'usa *Spring* conjuntament amb *Struts*, aquest és el paquet que permet una integració senzilla.
- ✓ **J2EE:** Proveeix integració amb aplicacions *Java Enterprise Edition* així com serveis *JMX, JMS, EJB*, etc.

Spring MVC

Spring MVC és un dels mòduls del *framework Spring*, i aquest proveeix un exhaustiu suport per al patró MVC, així com també proveeix suport d'altres característiques, una d'aquestes és facilitar la implementació de la capa de presentació. *Spring* proporciona un MVC per a web bastant flexible i configurable, però això no li lleva senzillés ja que pot realitzar aplicacions senzilles sense tenir que configurar moltes opcions.

El MVC Web de *Spring* té algunes característiques principals:

- ✓ Clara divisió entre controladors, models web i vistes.
- ✓ Està basat en interfícies i és bastant flexible.
- ✓ Proveeix interceptors al igual que controladors.
- ✓ No obliga a usar JSP com a única tecnologia a la capa de la vista.
- ✓ Els controladors són configurats com els altres objectes, mitjançant de *IoC*.

En *Spring* MVC el nostre *DispatcherServlet* funciona sobre el patró *Front Controller*. El patró front *controller* ens dona un punt d'entrada únic a les nostres peticions. De manera que totes elles van a passar per un mateix *servlet*, en el cas de *Spring MVC*, es tracta de *DispatcherServlet*. Aquest *servlet* s'encarrega de gestionar tota la lògica a la nostra aplicació.

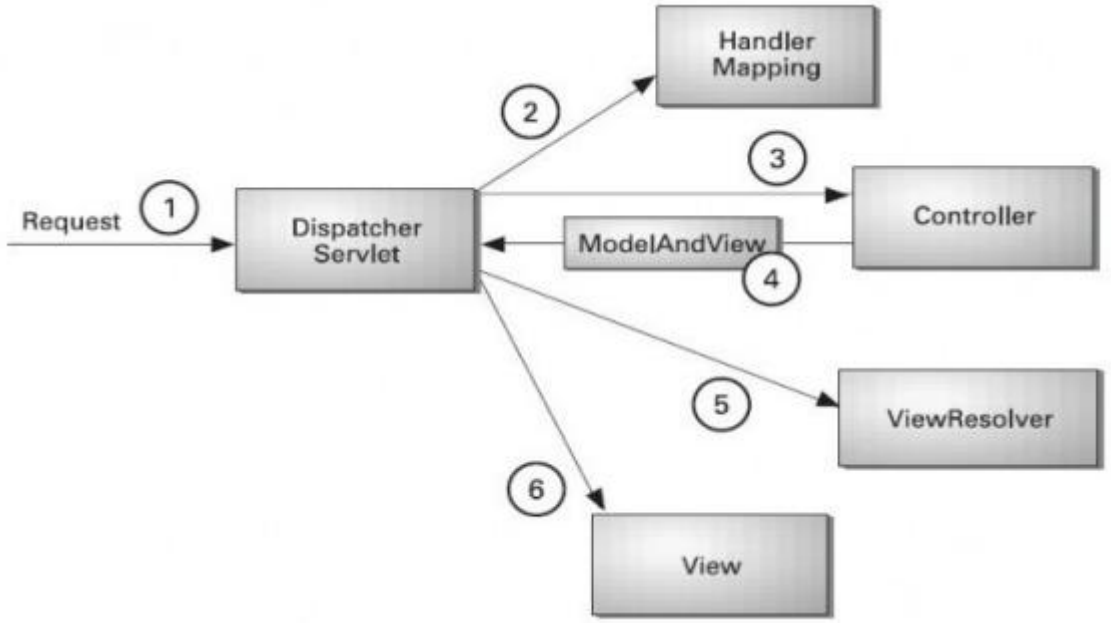


Figura 15: Funcionalitat básica en una aplicación Spring MVC

El flux bàsic en una aplicació sobre *Spring MVC* és el següent:.

- ✓ La petició arriba al *DispatcherServlet* (1).

- ✓ El *DispatcherServlet* tindrà que trobar quin controlador va a gestionar la petició. Per a fer-ho, el *DispatcherServlet* té que trobar el maneijador associat a la *URL* de la petició. Tot això es realitza en la fase de *HandlerMapping* (2).

- ✓ Un cop trobat aquest Controller, el DispatcherServlet li deixara gestionar al *Controller* la petició (3). Al controlador s'ha de realitzar tota la lògica de negoci de l'aplicació, és a dir, aquí cridarem a la capa de serveis. El controlador retornarà al *Dispatcher* un objecte de tipus *ModelAndView* (4). On el Model seran els valors que obtindrem de la capa de serveis i *View* serà el nom de la vista en la que volem mostrar la informació que va dintre del Model.

- ✓ Un cop passat aquest *ModelAndView* al *DispatcherServlet*, serà aquest el que tindrà que associar el nom de la vista retornada pel controlador a una vista concreta (pàgina *JSP, JSF,...*). Aquest procés ve indicat en la imatge com *ViewResolver* (4).

- ✓ Finalment i un cop resolta la vista, el *DispatcherServlet* tindrà que passar el Model (els valors recollit en el controlador a través de la capa de serveis) a la vista concreta *View* (5).

- ✓ En els passos de *HandlerMapping*, selecció de *Controller* i *ViewResolver* podem indicar-li al *DispatcherServlet* quina estratègia seguir.

2.3.1.4 JSF

Introducció

JSF és un *framework* MVC basat en el API de *Servets* que proporciona un conjunt de components en forma d'etiquetes definides en pàgines XHTML mitjançant el *framework Facelets. Facelets* es defineix en la especificació 2 de JSF com un element fonamental de JSF que proporciona característiques de plantilles i de creació de components compostos. Abans de la especificació actual s'utilitzava JSP per composar les pàgines JSF.

Entrant en el l'escenari MVC, JFS utilitza les pàgines *Facelets* com vista, objectes *JavaBean* com models i mètodes d'aquests objectes com controladors. El *servlet FacesServlet* realitza la tasca més important, la de processar les peticions HTTP, obtenir les dades d'entrada, validar-los i convertir-los, col·locar-los en els objectes del model, invocar les accions del controlador i renderitzar la resposta utilitzant l'arbre de components. Tot seguit veurem algunes de les característiques més destacables del MVC de JSF.

- ✓ Navegació entre vistes.
- ✓ Defineix les interfícies mitjançant vistes que agrupen components gràfics.
- ✓ Conversió de dades i validació automàtica de l'entrada de l'usuari.
- ✓ Suporta internacionalització i accessibilitat.

- ✓ A partir de la especificació 2.0 un model estàndard de comunicació *AJAX* entre la vista i el servidor.
- ✓ Connexió dels components gràfics amb les dades de l'aplicació mitjançant els anomenats *beans* gestionats.

JSF MVC

El model de *JSF* orientat a esdeveniments permet a les aplicacions estar menys condicionades a detalls *HTTP* i simplifica l'esforça necessari per al seu desenvolupament. *JFS* també millora l'arquitectura tradicional del MVC fent més fàcil moure la capa de negoci i presentació fora del controlador, i treu fora la lògica de negoci de les pàgines *JSP*.

En la implementació del MVC per part de *JSF*, els *beans* controladors de mapeijos interactuen entre la vista i el model. A causa d'això és important imitar la lògica de negoci i la lògica de persistència en els *bean* de control que estan lligat als *JSF*. Una alternativa comú és delegar la lògica de negoci al model de l'aplicació. En aquest cas, els *bean* de control també mapeigen el mapa d'objectes del model on la vista pot ser-los mostrada. Es tendeix a separar els *beans* en dos categories:

- ✓ *Bean* de control que estan lligats a les *JSF* (controladors).
- ✓ *Bean* de control no lligats a les *JSF* (objectes de motel).

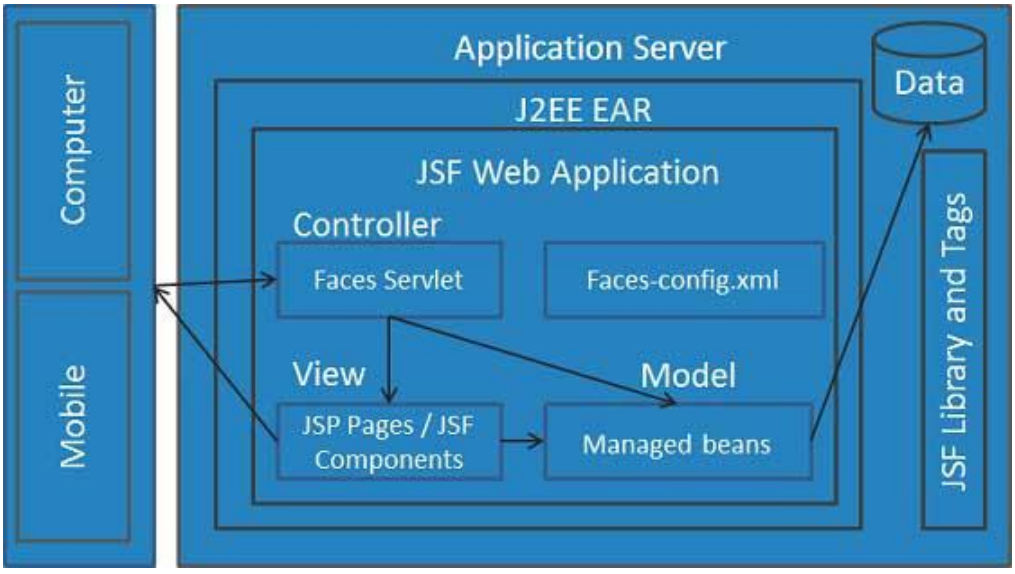


Figura 16: Arquitectura MVC de JSF

Capes

- ✓ **Model**

En *JSF* s'introdueix el concepte de *bean* administrat. El *bean* administrat és el pegament de la lògica de l'aplicació. Els *beans* administrats són definits en el fitxer *faces-config.xml* i proporcionen al desenvolupador de l'aplicació un total accés als mètodes mapejats dels *beans*. Aquest concepte de *IoC*, com ja hem comentat anteriorment, es usat de forma molt existosa en Spring. La característica de beans administrats és responsable de crear *beans* de backup i altres *beans* com Data Access Objects (DAO).

Un *bean* de backup és un POJO sense dependencies d'una implementació específica en classes o interfícies. EL controlador de *JSF* el *FacesServlet* no és conscient de quina acció ha agafat, només és conscient del resultat d'una acció i usarà aquest resultat per decidir sobre la navegació. Aquest *servlet* és el component que és conscient de quina acció o mètode hi ha que cridar en cada esdeveniment de l'usuari.

- ✓ **Vista**

Aquesta capa descriu el disseny, comportament i *rendering* de l'aplicació. El *UIComponent* és una de les peces claus d'una aplicació *JSF*. Els seus components són fonamentals per la capa de la vista i representen el comportament i estructura de l'aplicació. Un desenvolupador usaria aquest components per a construir una aplicació ajuntant components dins dels altres. Aquesta estructura, en temps d'execució, serà presentada com un component jeràrquic.

La implementació de la vista en *JSF* és un model de component amb estat, a diferència que la tecnologia *JSP*. La vista en JSF està composada per dos peces: la vista arrel i les pàgines *JSP*. La vista arrel és una col·lecció de components UI que mantenen l'estat UI. Els components *JSF* usen el patró de disseny *Composite* per gestionar un arbre de components. La pàgina *JSP* sol·licita components UI a altres pàgines *JSP* i permet que enllaci camps de components a propietats dels *beans*, i també botons a manejadors d'esdeveniments i mètodes d'accions.

- ✓ **Controlador**

Faceservlet és el controlador que té *JSF*. El *FaceServlet* actua com el "guarda", controlant el flux de navegació i gestionant les peticions a la pàgina *JSF* apropiada.

Cicle de vida d'un JSF

El cicle de vida d'una pàgina *JSF* és similar al de una *JSP*. El client fa una petició *HTTP* de la pàgina i el servidor respon amb la pàgina traduïda a *HTML*. No obstant això, degut a les característiques extres que ofereix aquesta tecnologia, el cicle de vida proporciona alguns serveis addicionals mitjançant l'execució d'alguns passos extres.

Les passos del cicle de vida s'executen depenent de si la petició s'ha originat o no des de una aplicació *JSF* i si la resposta és o no generada amb la fase de renderitzat del cicle de vida de *JSF*.

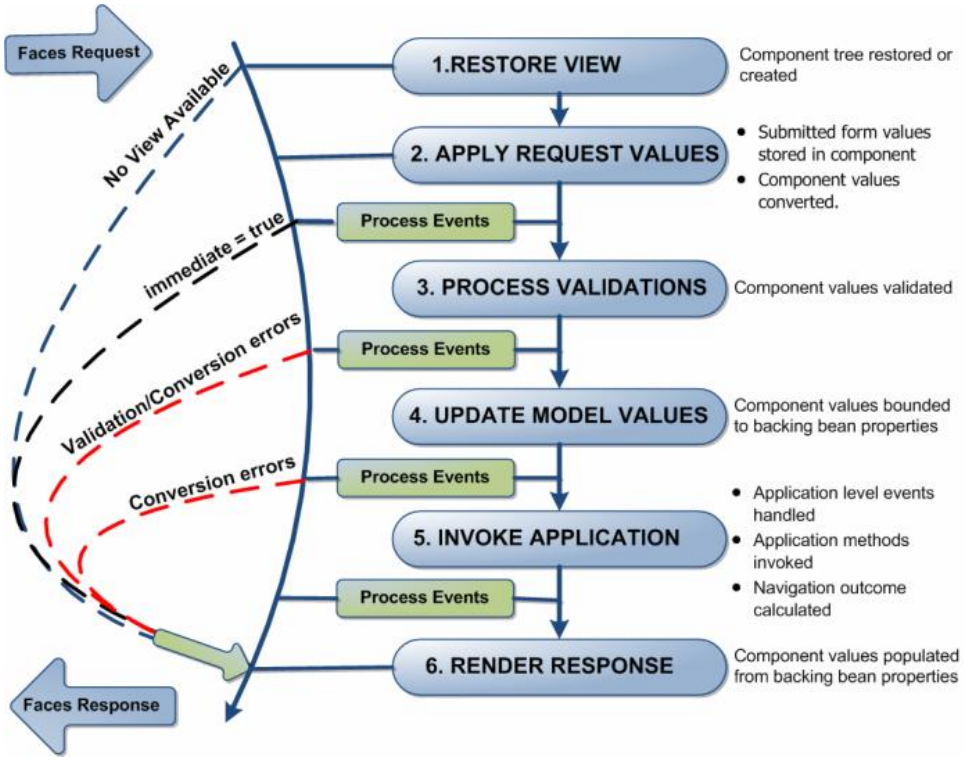


Figura 17: Flux bàsic de JSF dintre del MVC.

1. **Restaurar la vista:** En aquest pas s'obté l'arbre de components corresponent a la vista JSF de la petició. Si s'ha generat abans es recupera, i si és la primera vegada que l'usuari visita la pàgina, es genera a partir de la descripció *JSF*.
2. **Aplicar els valors de la petició:** Un cop obtingut l'arbre de components, es processen tots els valors associats a aquestos. Es converteixen totes les dades de la petició a tipus de dades Java i, per aquells que tenen la propietat *inmediate* a certa, es validen, avançant-se a la següent fase.
3. **Processar les validacions:** Es validen totes les dades. Si existeix algun error, es posa un missatge d'error i s'acaba el cicle de vida, saltant a l'últim pas.
4. **Actualitzar els valors del model:** Aquí ja s'han processat i validat tots els valors. S'actualitzen llavors les propietats del *beans* gestionats associats als components.
5. **Invocar a l'aplicació:** En aquesta fase totes les propietats dels *beans* associats a components d'entrada s'han actualitzat. Llavors es crida a l'acció seleccionada per l'usuari.
6. **Renderitzar la resposta:** invoca als atributs de codificació dels components i dibuixa els components de l'arbre de components guardat en el *FacesContext*.

2.3.1.5 Resum

- ✓ Struts
  - Avantatges
    - Estable: molt consolidat i provat.
    - Versió molt estesa.
    - Molta documentació, exemples, llibres, etc.
  - Inconvenients
    - La gestió de *Action* com a fils d'execució pot generar problemes.
    - Si el IDE no el suporta, pot resultar molt difícil mantenir la configuració.
    - Obsolet: actualment hi ha millors alternatives.
    - Per a una sola pantalla hem de crear almenys 2 classes (*ActionForm* i *Action*) .
- ✓ Struts2

- Avantatges
  - Suport per AJAX.
  - Millores en els *tags.*
  - Accions POJO (*Plain Old Java Objects*), no fa falta estendre de *Action.*
  - Formularis POJO, ja no s'usa els *ActionForm*.
  - Integració d'eines de *debbugins* i *profiling*.
- Inconvenients
  - No hi ha *feedback* per les propietats que no s'han especificat o per a les expressions *OGNL* no vàlides.
  - La documentació no està molt ben organitzada.

✓ Spring

- Avantatges
  - Ofereix divisió clara entre *Controllers*, Models (JavaBeans) i Views
  - No obliga a usar *JSP*. Permet usar *XLST, Velocity, FreeMaker* o implementar el teu propi llenguatge.
  - Molt flexible: implementació mitjançant interfícies.
  - Els controladors es configuren mitjançant *IoC* com els altres objectes (fàcilment testejables i integrables).
  - Ofereix un *framework* per a totes les capes de l'aplicació.
  - S'enllaça directament amb els *beans* de negoci, no existeix *ActionForms.*
  - Quantitat de codi testejable, les validacions no depenen de la API de *servlets.*
- Inconvenients
  - Configuració complexa: molts d'arxius *XMLs*.
  - Alguns cops massa flexible: no disposa d'un Controlador comú.
  - Corba d'aprenentatge llarga.

✓ JSF

- Avantatges
  - Molts de components.
  - Bon suport per diversos *IDEs.*
  - Ràpid i fàcil per començar a treballar.
  - Java EE Standard.
  - Moltes llibreries i eines.

        - El codi JavaScript s'incrusta com a part de components.
    - Inconvenients
        - Quan alguna cosa no funciona com s'espera, és difícil depurar.
        - Seguir estàndards fa que l'evolució de *JSF* no sigui tan ràpida com altres *frameworks*.
        - Pesat, tan en CPU com en memòria.

Informe d'ús de frameworks J2EE

S'ha realitzat un informe que han elaborat la gent de ZeroTurnAround on pot ser molt útil a l'hora de prendre una decisió sobre el *framework* a escollir. Aquest informe és molt complet i exhaustiu, i analitza els frameworks: Spring MVC, Grails, Vaadin, GWT, Wicket, Play, Struts, JSF. L'informe ha estat realitzat al 2012 i analitza els *frameworks* més populars del mercat, basant-se en 1800 responsables de desenvolupament.

Es comparen els *frameworks* entre ells, basant-se en les següents 8 categories:

1. Rapid application prototyping
2. Framework Complexity
3. Ease of Use
4. Documentation & Community
5. Framework Ecosystem
6. Throughput/Scalability
7. Code Maintenance/Updates
8. UX, Look and feel

Figura 18: Les diferents categories que han usat per comparar els frameworks

El gràfic següent mostra les puntuacions obtingudes de cada *framework*, basant-se en les característiques anteriors analitzades i comparades dels diferents *frameworks*.

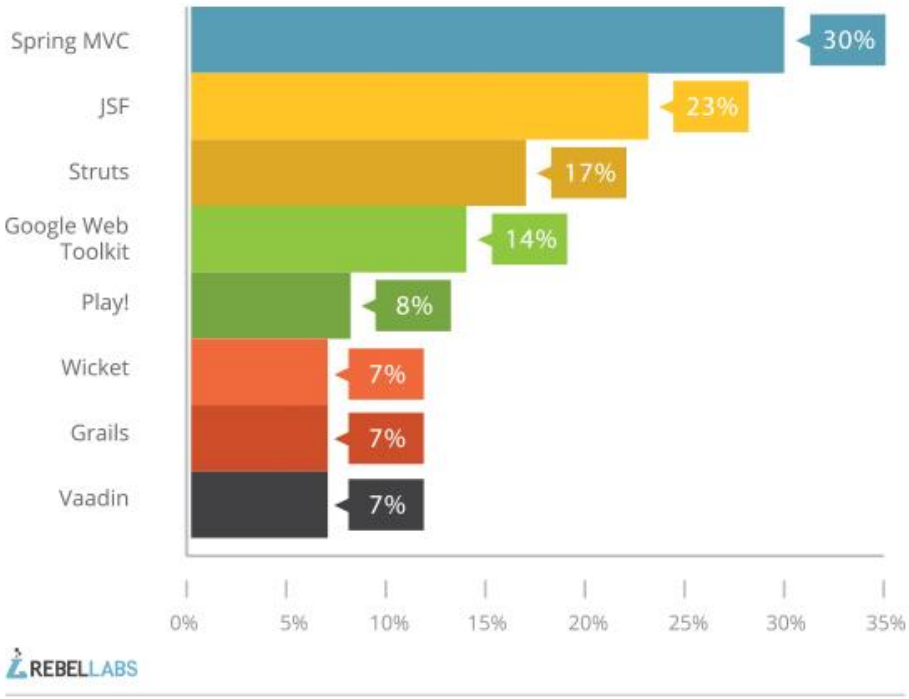


Figura 19: Gràfic que mostra les puntuacions obtingudes de cadascun dels frameworks

2.3.2 Frameworks de persistència

Tot seguit s'analitzaran dos dels dels *frameworks* de persistència més populars que hi ha avui en dia al mercat, els quals estan situats entre la capa de codi de l'aplicació Java i la de l'API JDBC:

- ✓ Hibernate
- ✓ MyBatis

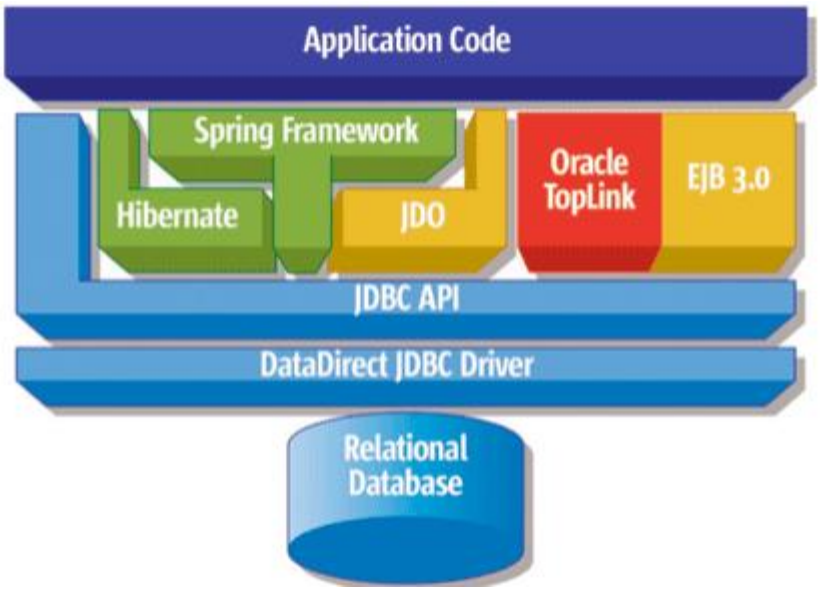


Figura 20: Arquitectura on s'inclou la relació entre persistència i base de dades

2.3.2.1 Hibernate

Hibernate és una eina ORM completa que ha aconseguit en un temps record una excel·lent reputació en la comunitat de desenvolupament es posiciona clarament com el producte *OpenSource* líder d'aquest camp gràcies a les seves prestacions, bona documentació i escalabilitat.

Hibernate parteix d'una filosofia de mapeig d'objectes Java, també conegut com POJOs (Plain Old Java Objects), no contempla la possibilitat d'automatitzar directament la persistència de *Entity Bean* tipus *BMP* (és a dir, generar automàticament aquests tipus d'objectes), encara que si és possible combinar *Hibernate* amb aquests tipus de *beans* utilitzant els coneguts patrons per a la delegació de persistència en POJOs.

Tot seguit es mencionen algunes de les característiques principals:

- ✓ **Simplicitat y flexibilidat**: necessita un únic fitxer de configuració en temps d'execució i un document de mapeig per cada aplicació. Aquest fitxer pot ser l'estàndard de Java o un fitxer XML. L'ús de *frameworks* de persistència, com ara EJBs fa que l'aplicació depengui del *framework*. *Hibernate* no crear aquesta dependència addicional. Els objectes persistents en l'aplicació no tenen que heretar d'una classe d'*Hibernate* o obeir a una semàntica específica. Tampoc necessita un contenidor per funcionar.

- ✓ **Prestacions**: una de les grans confusions que apareixen al usar aquest tipus de *frameworks* és creure que els prestacions de l'aplicació se'n ressentiran. Aquest no és el cas d'*Hibernate*. La clau d'aquest tipus de situacions és si es realitzen el número mínim de consultes a la base de dades. Molts de *frameworks* de persistència actualitzen les dades dels objectes inclús quan no ha canviat el seu estat. El cacheig d'objectes juga un paper molt important en la millora de les prestacions de l'aplicació. *Hibernate* accepta diferents productes cachejats, tan de codi lliure com comercial.

- ✓ **Complet**: ofereix totes les característiques d'orientació a objectes, incloent la herència, tipus d'usuari i les col·leccions. A més a més també proporciona una capa

d'abstracció SQL anomenada HQL. Les sentencies HQL són compilades pel *framework* de *Hibernate* i cachejades per a la seva possible reutilització.

La següent figura mostra els rols de les interfícies *Hibernate* més important en les capes de persistència i de negoci d'una aplicació J2EE. La capa de negoci està situada sobre la capa de persistència, ja que la capa de negoci actua com un client de la capa de persistència. A més a més, *Hibernate* fa ús de APIs de Java, com ara JDBC, JTA (Java Transaction Api i JNDI (Java Naming Directory Interface).

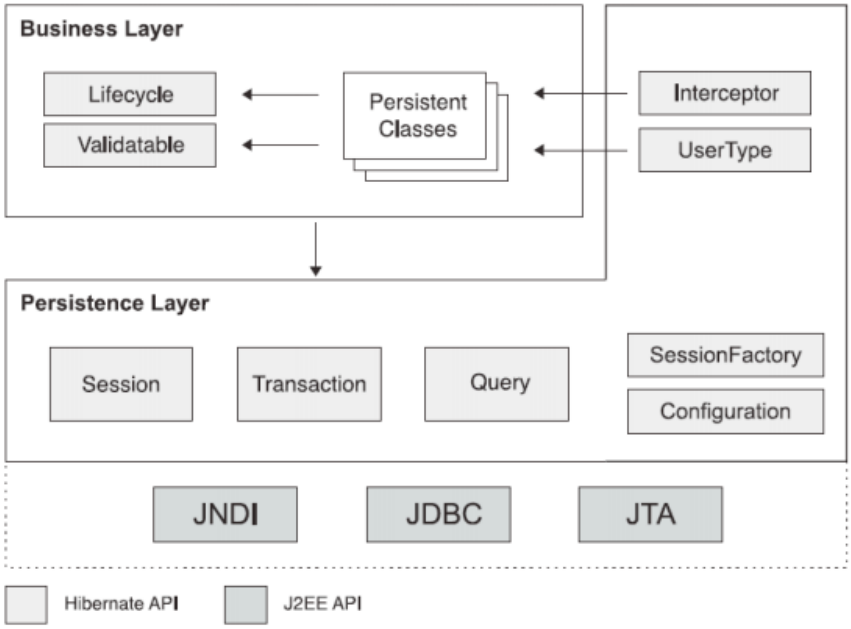


Figura 21: Rols de les interfícies Hibernate entre capes persistencia i lógica de negoci

2.3.2.2 MyBatis

MyBatis és un *framework* de persistència que suporta SQL, procediments emmagatzemats i mapeig avançats. *MyBatis* elimina casi tot el codi JDBC, l'establiment manual dels paràmetres i l'obtenció de resultats. *MyBatis* pot configurar-se amb XML o anotacions, i permet mapeijar mapes i POJOs amb registres de base de dades. Per tant podem dir que la seva funció principal és la de proporcionar un API per gestionar la capa de Persistència de dades en la Base de Dades (estaria situada entre la capa de Negoci i la Base de Dades). Es podria dir que *MyBatis* és la unió de dos *frameworks* independents, però que s'usen junts, els DAO i els sqlMaps.

- ✓ **DAO** (Data Access Objects): com ja s'ha comentat en l'apartat tipus de patrons de disseny, DAO és un patró de disseny d'objectes que subministra una interfície comú entre l'aplicació i els dispositius d'emmagatzemament de dades, com ara una Base de Dades o un arxiu.

- ✓ **sqlMaps**: són fitxers XML que s'encarreguen de simplificar la persistència d'objectes en base de dades relacionals permetent significativament la codificació en Java per a una aplicació que gestiona base de dades relacionals.

Tot seguit podem veure l'arquitectura que presenta el *framework* MyBatis i on és situa la capa de persistència.

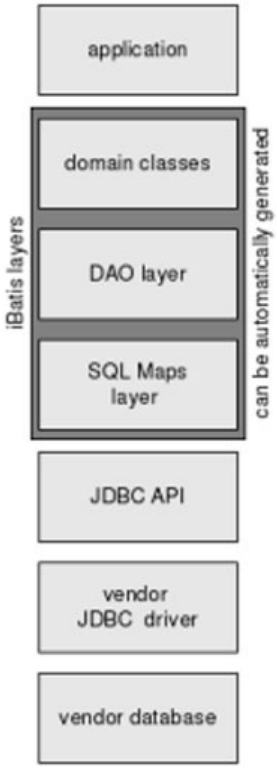


Figura 22: Arquitectura framework MyBatis

Per tant tenim que el *framework MyBatis* està compost per un conjunt de codificació basat en fitxers XML Map d'arxius de SQL (sqlMap), on existeix un fitxer per cadascuna de les taules de la base de dades, cadascun d'aquests XML posseeix plantilles de SQL que executen les consultes tal i com és van definir, a més dels resultats esperats per les classes de Java del domini. En el codi de l'aplicació, tenim els objectes d'accés a

MyBatis a través dels DAO, aquests actuen com l'API que executa les plantilles de mapes de SQL i assigna el resultat de les classes Java del domini corresponent.

Les funcions principals que dur a terme el *framework MyBatis* són:

- ✓ Executar els SQL escrits mitjançant JDBC (Java DataBase Connectivity), per tant ens oblidem dels *try/catch*.
- ✓ Mapeijar les propietats dels objectes a paràmetres per a les *PreparedStatement* (sentencies SQL parametritzades).
- ✓ Mapeijar els resultats d'una consulta a un objecte o una llista d'objectes

2.3.2.3 Resum

- ✓ Hibernate
  - Avantatges
    - Potent llenguatge de consulta (HQL).
    - Transaccions, cache, associacions, polimorfisme, herència, *lazy loading*, persistència transitiva, etc ...
    - Molta documentació, exemples, llibres, etc.
    - No intrusiu (estil POJO).
    - Comunitat activa amb molt usuaris.
  - Inconvenients
    - Major complexitat de disseny.
    - Corba d'aprenentatge més lenta.
    - Més sobrecarrega que les consultes directes SQL.
    - No apta per a aplicacions de gran gestió de dades.
- ✓ MyBatis
  - Avantatges
    - Simplicitat (fàcil d'usar si sabem SQL).
    - Abstreu JDBC.
    - Permet mapeig directe a XML.
    - Corba d'aprenentatge ràpida.
    - Control complet sobre les SQL.
    - *Batches, lazi loading*, transaccions, paginació.

- Inconvenients
  - Les bases de dades han de ser exclusivament relacionals.
  - No es totalment transparent (hi ha que programar SQL).
  - Perd funcionalitat si casi totes les sentencies SQL són construïdes dinàmicament.

Per tant *MyBatis* i *Hibernate* són dos mecanismes diferents de persistència de dades en una base de dades relacional. Cadascú té les seves avantatges i limitacions. MyBatis no proporciona una solució ORM completa, i no proporciona ninguna assignació directa dels objectes i models relacionals. No obstant això *MyBatis* proporciona un control complet sobre les consultes. *Hibernate* proporciona una solució ORM completa, però en canvi no ofereix un control sobre les consultes.

Tot seguit és mostra una comparativa dels dos *frameworks* basant-se en les següents característiques.

Característiques	**MyBatis**	**Hibernate**
Simplicitat	Molt bo	Bo
Solució completa ORM	Millorable	Molt bo
Adaptabilitat a canvis en el model de dades	Bo	Millorable
Complexitat	Molt bo	Millorable
La dependència de SQL	Bo	Millorable
Rendiment	Bo	Molt bo
Portabilitat a través de diferents bases de dades relacionals	Millorable	Molt bo
Portabilitat a les plataformes de no-Java	Molt bo	Bo
Comunitat de suport y documentació	Millorable	Molt bo

3 Arquitectura Liferay

En aquest apartat definirem el *CMS Liferay* a nivell global, on es donarà una visió general de com funciona el portal, de les seves característiques principals i dels seus components.

En segon lloc, i entrant una mica més a fons, es detallaran els diferents components sobre els que s'ha treballat en el prototipus.

Finalment es mostra l'entorn de desenvolupament per a que es pugui veure a grans trets com estan enllaçades totes aquestes tecnologies i components.

3.1 Introducció Portal Liferay

Nosaltres necessitem pal·liar el problema de la descentralització de la informació buscant un portal web per a que sigui un únic punt d'accés a informació, dades, contingut, aplicacions, etc.... *Liferay Portal* és una eina de gestió de continguts, líder en la comunitat Java, amb una orientació Web 2.0 i utilitza llicència *Open Source*. És un gran gestor de continguts ja que té una arquitectura flexible i modular, té multitud d'eines per la gestió de continguts i estalvia temps i recursos per crear pàgines webs.

Algunes de les característiques principals:

- ✓ Reconeixement internacional com a plataforma de portals més segura del mercat.
- ✓ Funciona en tots els sistemes operatius, servidors d'aplicacions, base de dades (en més de 700 combinacions de desplegaments).
- ✓ Basat en estàndards: *JSR 127* (JSF), *JSR 168* (Porlet Specification), *JSR 286* (Porlet 2.0 Specification), *JSR 170* (Content Repository), *JSR 208* (Java Business Integration), *AJAX, Spring, Struts, Tiles, Velocity, WSRP.*
- ✓ Potent sistema de gestió d'organitzacions, usuaris i rols.
- ✓ Incorpora una suite de eines col·laboratives (wikis, calendaris, blogs, fòrums,...).
- ✓ Autenticació i *Single Sign-On* (SSO) *: LDAP, Facebook, OpenID*, etc ...
- ✓ Integració amb sistemes de terces com ara el gestor documental *Alfresco*.

En la següent imatge podem veure tot el que engloba Arquitectura Lògica de Liferay.

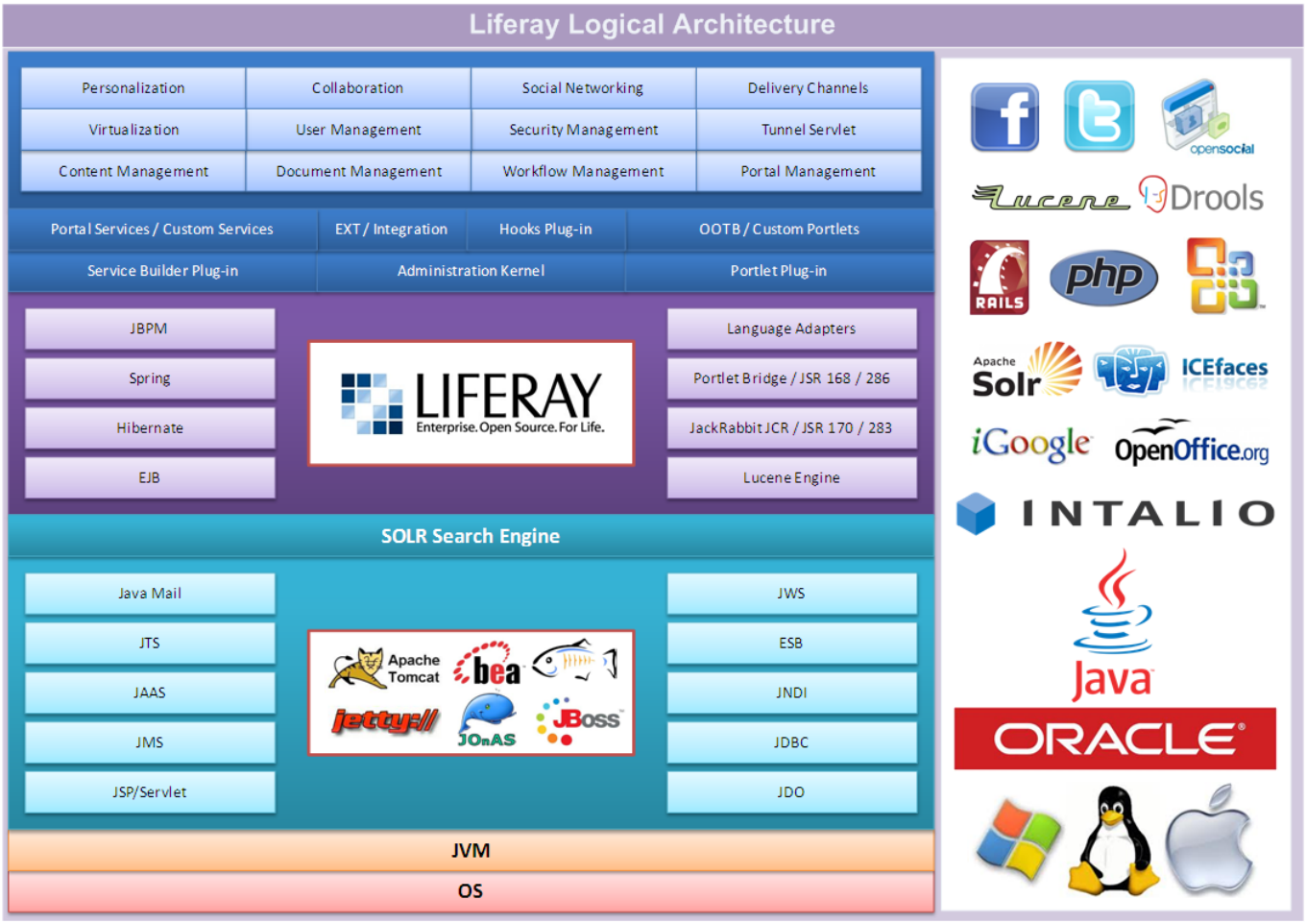


Figura 23: Arquitectura lógica de Liferay

3.2 Arquitectura dels components

En *Liferay* hi intervenen molts de tipus de components com podem veure a la imatge següent on mostra l'arquitectura de components que té el sistema:

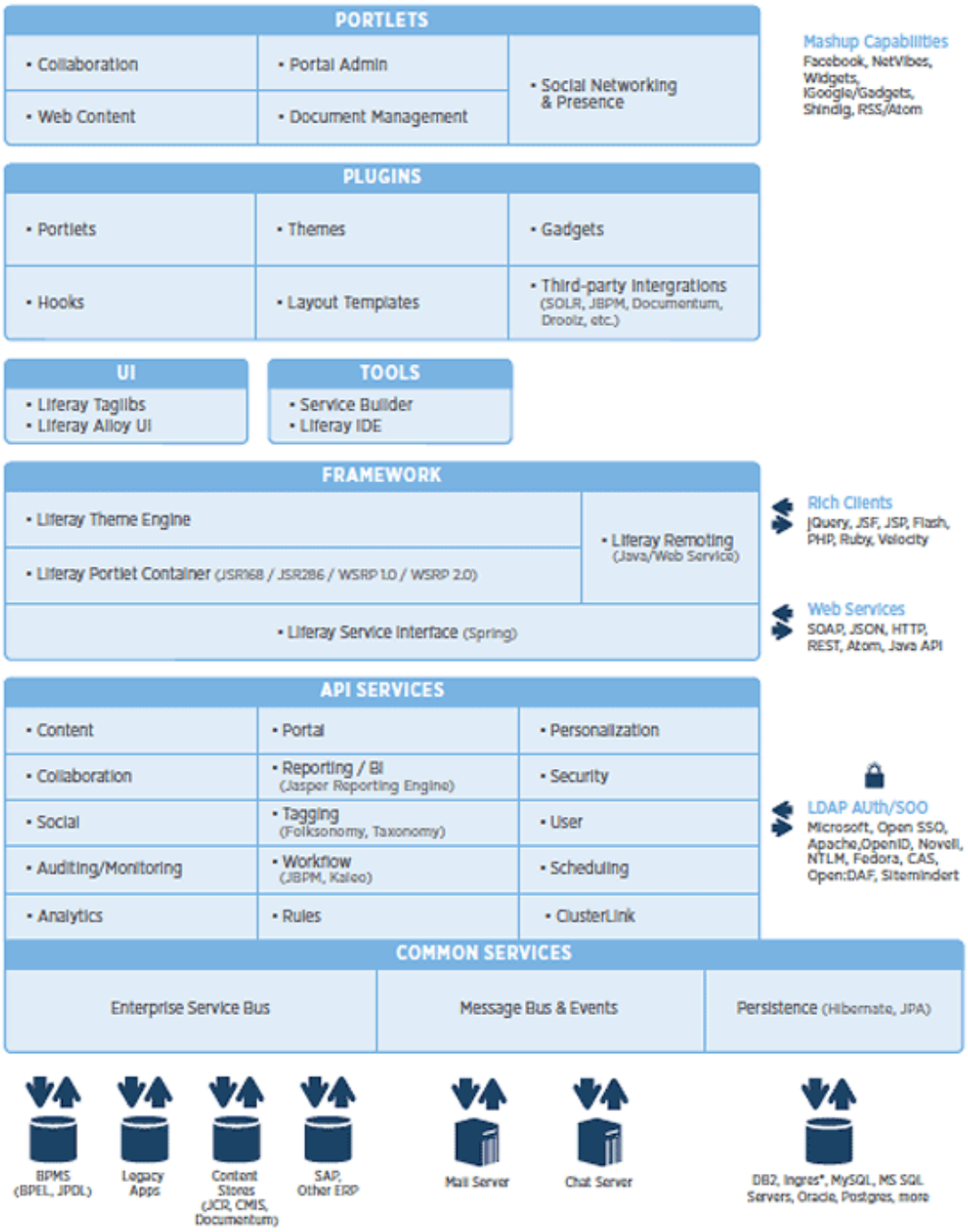


Figura 24: Arquitectura dels diferents components que disposa Liferay

Nosaltres ens centrarem sobretot en els *plugins*, ja que usarem un d'ells per tal de desenvolupar una part de l'aplicació. Existeixen 4 tipus principals de *plugins*:

✓ **Theme**

Un Theme o Tema d'aparença és un *plugin* que permet modificar l'estil o disseny d'un portal web o d'una pàgina concreta. Cada Tema d'Aparença pot disposar d'una sèries d'esquemes de color que permeten fer diverses variants del

mateix Tema d'Aparença i mantenir-ho tot en un mateix *plugin*. Per exemple, podem fer que un Tema de Aparença disposi de 2 variants de tonalitats, una en verd i l'altra en groc.

- ✓ **Layout**

Una *Layout* o Plantilla és un *plugin* que permet definir una disposició de pàgina diferent per a cadascuna de les pàgines. Són sobre aquestes les que es col·loquen els *porlets* i afecten únicament i exclusivament al contingut central de la pàgina. La capçalera, navegació principal i *footer* es definiran directament en el *theme*. Per exemple, podem tenir plantilles de 2 columnes al 40%, 3 columnes al 33%, 1 fila al 100% i 2 columnes al 60%, etc.

- ✓ **Hook:**

A diferència dels *porlets*, aquest tipus de *plugin* permet modificar el codi natiu del portal. S'utilitzen, sobretot, per modificar el codi d'algun *porlet* natiu del portal, definir variables d'idioma o modificar alguna funcionalitat del poral.

Existeixen 4 tipus de *hooks* principals:

- *Custom JSPs*: Permeten modificar el codi d'un o diversos *porlets* mitjançant la modificació dels seus *JSPs*.
- *Portal properties*: Permeten modificar les propietat del portal o definir-ne de noves.
- *Services*: Permeten modificar els serveis del portal directament o afegir-ne de nous.
- *Language properties*: Permeten declarar noves variables d'idiomes per facilitar la internacionalització del portal. Aquestes variables després seran usades en diferents punt del portal com, per exemple, per a traduir algun text d'un Tema d'Aparença o d'un *porlet*.

✓ **Porlet:**

Els *porlets* són el component principal de programació de *Liferay*. Són components modulars d'una interfície d'usuari que proporciona contingut específic, el qual pot ser un servei o dades aprovisionades per un sistema d'informació. Aquest és genera mitjançant el llenguatge de programació orientat a objectes Java i *JSP*, el qual presenta el seu contingut als usuaris del portal, que és el sistema on aquests són executats i es visualitzen com una col·lecció de finestres de *porlet* que no es solapen, on en cada finestra es mostra un *porlet*.

Aquests *porlets* són components modulars ja que cadascun d'ells s'encarrega de generar la seva pròpia interfície d'usuari al portal, ja que és independent d'aquest, i a causa d'això és un component portable. Per tant funcionen com a mòduls independents i cadascú té una funcionalitat diferent. Qualsevol codi creat dintre d'un *porlet* no afectarà mai al codi natiu del portal.

Com ja s'ha comentat anteriorment *Liferay* disposa de molts de *porlets* natius per a ser usats però també es poden crear *porlets* nous mitjançant els estàndards establerts. Per exemple l'estàndard *JSF-2864*, que serà l'escolli't per a desenvolupar el *porlet* d'aquest projecte. Podem destacar les dues fases principals que té:

- **Render:** es crida quan el *porlet* necessita repintar la pàgina, és a dir, és l'acció que es fa per carregar la pàgina.
- **Action Phase:** és quan el *JSP* del *porlet* crida una 'ActionURL', fa el procés assignat a aquesta 'ActionURL' i després torna a fer la fase render.

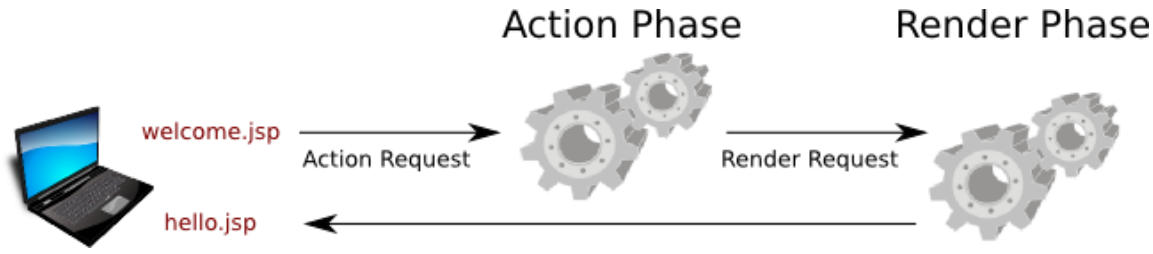


Figura 25: Flux bàsic entre les pàgines JSP i les dues fases principals del portlet

Una vegada vistes les fases que pot tenir un *portlet* (si més no, les més importants), és important conèixer també el seu cicle de vida.

El cicle de vida d'un *portlet* consta de quatre fases:

- **init():** mètode per a inicialitzar el *portlet*. Es crida una sola vegada amb la seva creació.
- **processAction():** mètode cridat cada vegada que es realitza una acció (o ActionURL) sobre el *portlet*.
- **render():** mètode per a realitzar el *renderitzar* (o pintat) del *portlet*.
- **destroy():** mètode que s'executa només una vegada, al eliminar el *portlet*. Bàsicament serveix per a alliberar possibles recursos que utilitzi el *portlet*.

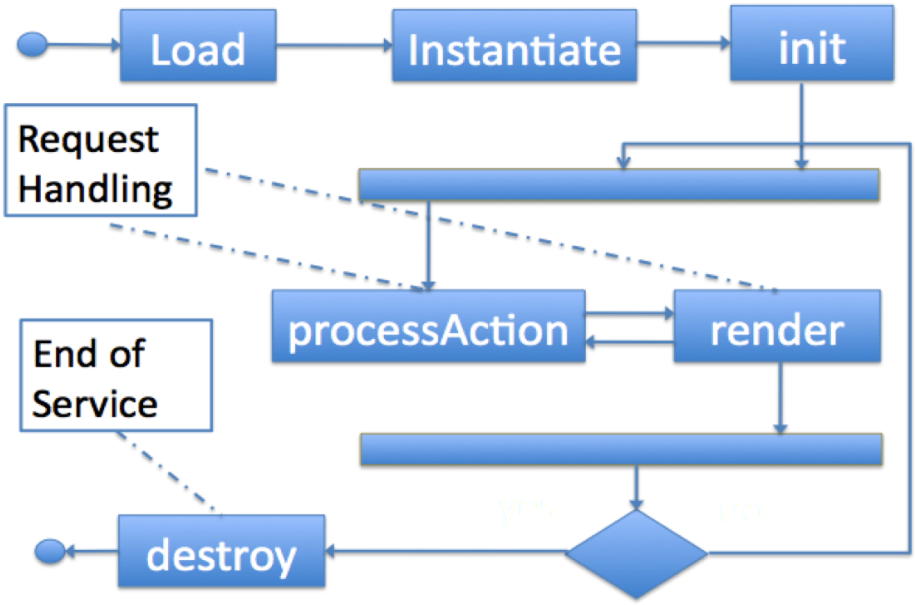


Figura 26: Cicle de vida d'un portlet

3.3 Entorn de desenvolupament

Un cop vist els principals components que intervenen en *Liferay* ens centrarem en l'entorn que necessita *Liferay* per a ser desplegat. Fent èmfasi a grans trets en les diverses integracions a aplicacions externes i a tots els sistemes implicats en el seu funcionament.

En la imatge següent es mostra el servidor *Liferay* en un entorn corporatiu, col·locat a la LAN interna i intercomunicat amb tots els altres sistemes per tal que els seus components puguin interactuar amb tots els elements i serveis definits anteriorment.

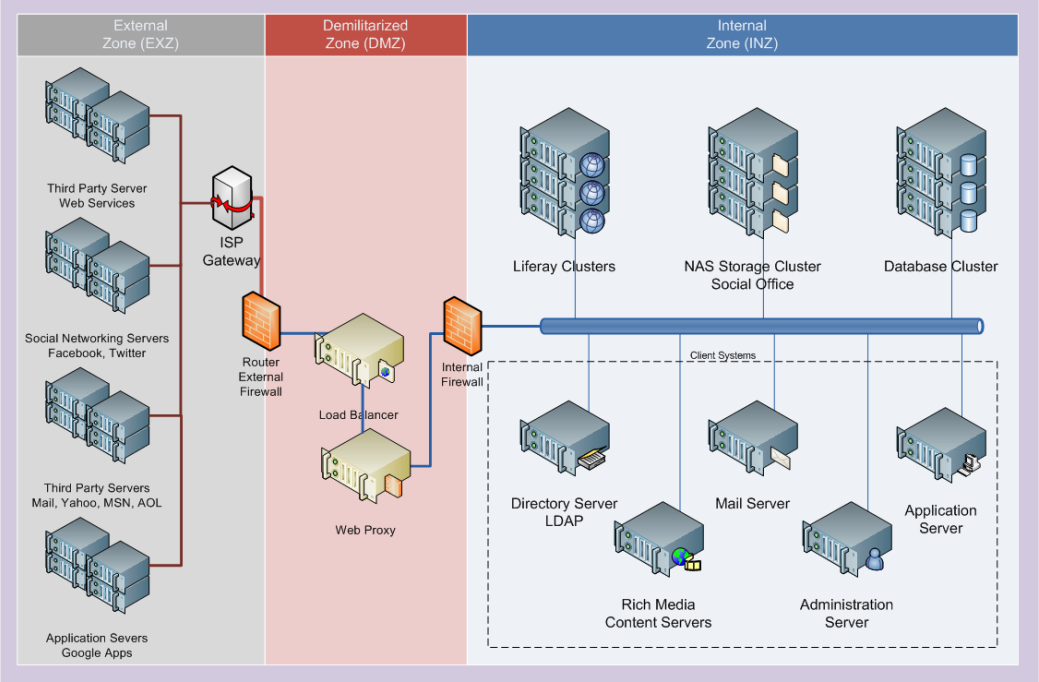


Figura 27: Exemple d'entorn de desplegament dintre d'un entorn de producció

Com hem comentat anteriorment, *Liferay* desataca per la seva compatibilitat i fàcil integració amb els sistemes, tecnologies i eines més populars del mercat d'avui en dia.

Tot seguit farem un resum de les característiques de compatibilitat en el seu desplegament:

- ✓ **Sistemes operatius**
  - Linux (CentOS, RHES, SUSE, Ubuntu y otros)
  - Unix (AIX, HP-UX, Mac OS X, Solaris y otros)
  - Windows
- ✓ **Contenedors de Servlets**
  - Jetty
  - Resin
  - Tomcat
- ✓ **Servidors d'aplicacions**
  - Geronimo
  - GlassFish
  - JBoss
  - OracleAS
  - WebLogic

- WebSphere

✓ **Java Runtimes**

- Java Standard & Enterprise Edition (SE/EE) 5
- Java Standard & Enterprise Edition (SE/EE) 6
- Java Standard & Enterprise Edition (SE/EE) 7

✓ **Base de dades**

- IBM DB2
- MySQL
- Oracle
- PostgresSQL
- SQL Server

✓ **Entorns de Cloud Computing**

- *Liferay* està preparat per ser desplegat en el núvol i en entorns virtualitzats, incloent EC2 i VMWare.

Per últim cal destacar que *Liferay* ha recollit en els últims anys premis i reconeixements qualificats com a solució tecnològica no només per ser el millor entorn *Open Source* per a portals, sinó pel seu caràcter innovador i visionari, situant-se com a *Leader* en l'anomenat quadrant màgic de *Gartner*.

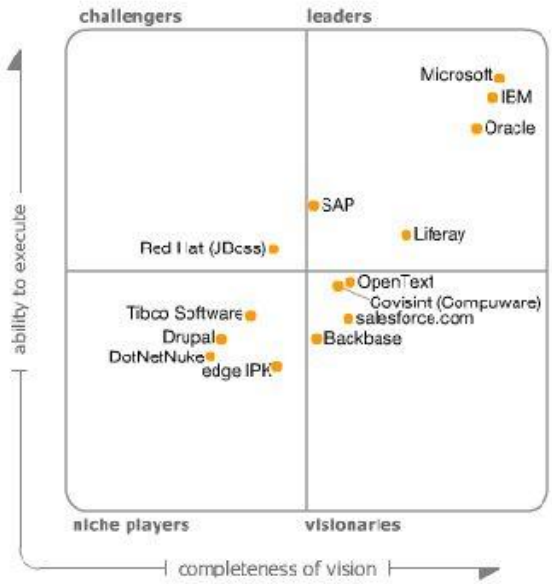


Figura 28: Liferay dintre del quadrant màgic de Gartner

4 Anàlisis funcional

En primer lloc es descriuran les funcionalitats de l'aplicació i es resumiran els principals requisits funcionals que ha de tenir la nostra aplicació.

Tot seguit descriurem els actors implicats en l'aplicació i mostrarem els diagrames de casos d'ús juntament amb les seves especificacions.

4.1 Requisits funcionals

El projecte a desenvolupar consisteix en crear una intranet corporativa per als empleats de l'ajuntament de Vinaròs. La idea principal és dotar d'una aplicació web accessible per a tots els usuaris de l'ajuntament on puguin trobar les aplicacions i eines més còmodament, és a dir, aquesta hauria de convertir-se en una eina bàsica per als usuaris per tal de centralitzar tota la informació que necessiten accedir dia a dia.

A nivell d'aplicacions implementarem inicialment dues, una wiki corporativa i una aplicació que facilitarà l'accés a altres aplicacions als usuaris. Amb la idea de deixar la intranet per poder estendre més les funcionalitats d'aquesta i introduir més informació centralitzada per als usuaris un cop finalitzat el projecte, per tal de tenir continuïtat.

La *Wiki* s'encarregarà d'emmagatzemar diferents temes interessants tant pel departament d'informàtica com per als usuaris. Inicialment aquesta *Wiki* està pensada per a que el departament d'informàtica pugui anar documentant casos i situacions que es van produint dia a dia per tal que puguin ser els propis usuaris, mitjançant la consulta de la *Wiki*, qui resolguin algunes de les problemàtiques o dubtes que els sorgeixen. A més a més, aquesta eina la pot usar el departament d'informàtica per tal de documentar instal·lacions d'aplicacions, problemes resolts, procediments interns, etc..., és a dir, informació només rellevant per al departament d'informàtica per tal de poder documentar alguns temes importants.

Per una altra banda, hi haurà una aplicació que enllaçarà les aplicacions que té disponible cada usuari, és a dir, vincularà els usuaris que hi ha a l'ajuntament i les aplicacions que utilitza cadascú dels usuaris. Mostrant-li els enllaços disponibles per a que pugui accedir. També ens servirà al departament d'informàtica per poder tenir informació útil de cadascuna de les aplicacions.

Requisits funcionals:

- ✓ Els usuaris han de poder accedir des de qualsevol navegador.
- ✓ L'autenticació ha d'estar integrada en el Directori Actiu.
- ✓ Ha d'haver diferents rols en l'aplicació.
- ✓ Els usuaris han de poder accedir només a les seves aplicacions.
- ✓ L'administrador ha de poder crear, modificar, eliminar les aplicacions.
- ✓ L'administrador gestionarà la *Wiki*.
- ✓ Els usuaris podran accedir als temes de la *Wiki* que tingui permís.
- ✓ L'administrador gestionarà el portal.

4.2 Diagrama de casos d'us

Podem considerar un cas d'ús com la descripció dels passos que s'hauran de realitzar per dur a terme algun procés. Per tant definirem els nostres casos d'ús de la nostra aplicació definint els actors i les tasques que realitzaran aquests.

Com s'ha anomenat als requeriments funcionals només disposarem de dos tipus d'usuaris i per tant només tindrem dos actors que interaccionaran amb el sistema, que seran l'usuari i l'administrador de l'aplicació (el departament TIC).

- ✓ **Diagrama de casos d'ús 1: Accés a l'aplicació**

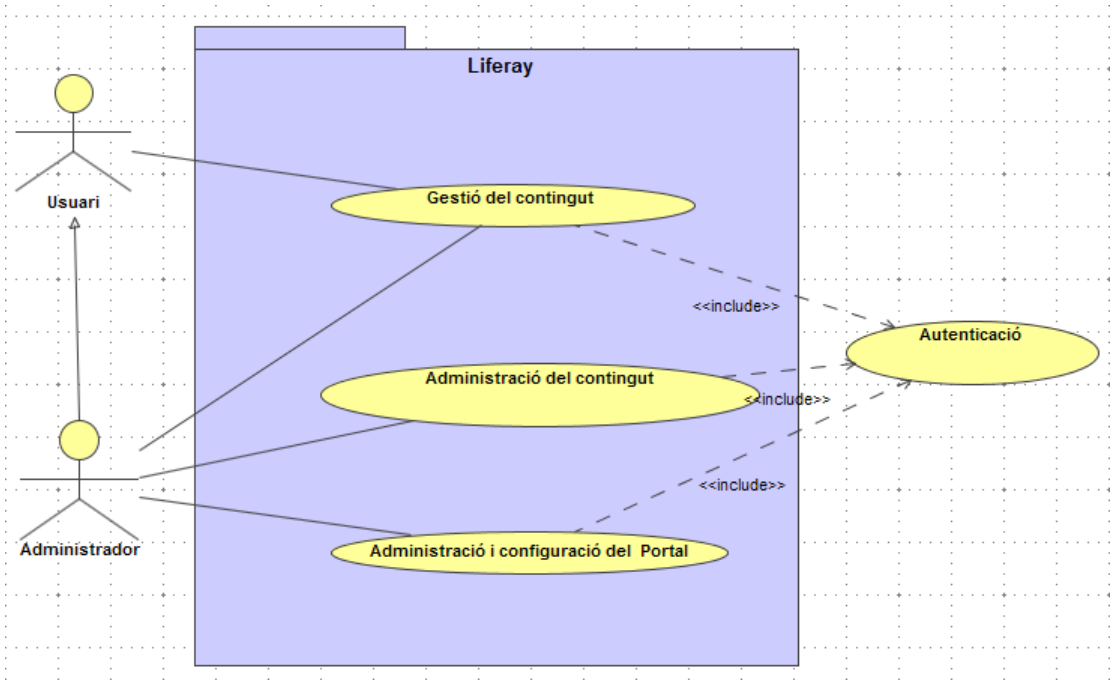


Figura 29: Diagrama de casos d'ús 1 – Accés a l'aplicació

Cas d'ús 1: Gestió del contingut

Descripció	Accedir a l'aplicació. Primer l'usuari s'autentica al sistema i pot accedir a totes les aplicacions que té accessibles. Un cop autenticat al sistema podem veure les diferents funcions que pot tenir l'usuari en el segon diagrama de casos d'ús.
Actors implicats	Administrador i Usuari
Casos d'ús relacionats	Autenticació
Precondició	Estar identificat correctament al sistema
Postcondició	Veure totes les aplicacions disponibles per l'usuari
Alternatives de procés i excepcions	Sinó existeix l'usuari, l'administrador l'hauria de donar d'alta al Directori Actiu.

Cas d'ús 2: Administració del contingut

Descripció	Accés al portal per a poder administrar les diferents característiques de les entitats (wiki, porlet activitats, etc ..)
Actors implicats	Administrador
Casos d'ús relacionats	Autenticació
Precondició	Estar identificat correctament al sistema
Postcondició	Poder administrar les diferents característiques de les entitats
Alternatives de procés i excepcions	Sinó existeix l'usuari, l'administrador l'hauria de donar d'alta al Directori Actiu.

Cas d'us 3: Administració i configuració del Portal

Descripció	Control sobre el portal i totes les seves organitzacions. Possibilitat de modificar paràmetres bàsics, usuaris, carregar components (*portlets*, temes i altres).
Actors implicats	Administrador
Casos d'ús relacionats	Autenticació

Precondició	Estar identificat correctament al sistema
Postcondició	Poder administrar les diferents característiques de les entitats
Alternatives de procés i excepcions	Sinó existeix l'usuari, l'administrador l'hauria de donar d'alta al Directori Actiu.

Cas d'ús 4: Autenticació

Descripció	Permet a l'usuari identificar-se al sistema. Una vegada identificat, obtindrà permisos sobre diferents accions en funció del seu perfil.
Actors implicats	Administrador i Usuari
Casos d'ús relacionats	
Precondició	Estar identificat correctament al sistema
Postcondició	Accedir a l'aplicació
Alternatives de procés i excepcions	Sinó existeix l'usuari, l'administrador l'hauria de donar d'alta al Directori Actiu.

- ✓ **Diagrama de casos d'ús 2: Gestió d'aplicacions i Wiki**

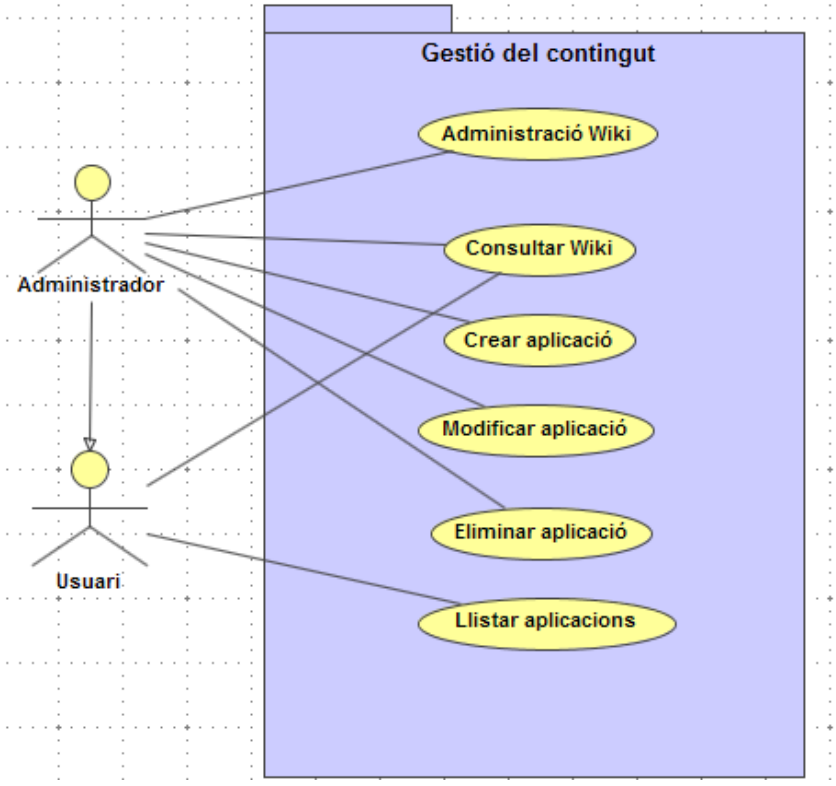


Figura 30: Diagrama de casos d'ús 2 – Gestió d'apliacions i Wiki

Cas d'ús 5: Administració Wiki

Descripció	Permet a l'administrador poder gestionar i administrar el *porlet* de la *Wiki*, tant a nivell de continguts com a nivell de permisos.
Actors implicats	Administrador
Casos d'ús relacionats	
Precondició	Estar identificat correctament al sistema i ser administrador
Postcondició	Accedir a la gestió i administració del *porlet Wiki*
Alternatives de procés i excepcions	

Cas d'ús 6: Consultar Wiki

Descripció	Permet a l'usuari com l'administrador poder consultar els diferents temes que hi ha a la *Wiki* segons els permisos.
Actors implicats	Administrador i Usuari
Casos d'ús relacionats	
Precondició	Estar identificat correctament al sistema
Postcondició	Accedir als diferents temes de la *Wiki*
Alternatives de procés i excepcions	

Cas d'ús 7: Crear aplicació

Descripció	Permet crear a l'administrador una aplicació amb les seves dades i vincular-la a un grup del Directori Actiu, d'aquesta manera es donarà permisos d'accés a l'aplicació.
Actors implicats	Administrador
Casos d'ús relacionats	
Precondició	Estar identificat correctament al sistema i ser administrador
Postcondició	Afegir una nova aplicació

Alternatives de procés i excepcions	

Cas d'ús 8: Modificar aplicació

Descripció	Permet modificar a l'administrador una aplicació ja existent.
Actors implicats	Administrador
Casos d'ús relacionats	
Precondició	Estar identificat correctament al sistema i ser administrador
Postcondició	Modificar una aplicació ja existent
Alternatives de procés i excepcions	

Cas d'ús 9: Eliminar aplicació

Descripció	Permet eliminar a l'administrador una aplicació ja existent.
Actors implicats	Administrador
Casos d'ús relacionats	
Precondició	Estar identificat correctament al sistema i ser administrador
Postcondició	Eliminar una aplicació ja existent
Alternatives de procés i excepcions	

Cas d'ús 10: Llistar aplicacions

Descripció	Permet llistar les aplicacions disponibles que té l'usuari segons els permisos atorgats.
Actors implicats	Administrador i Usuari
Casos d'ús relacionats	
Precondició	Estar identificat correctament al sistema

Postcondició	Visualitzar totes les aplicacions disponibles
Alternatives de procés i excepcions	

5 Disseny

5.1 Disseny arquitectònic

5.1.1 Entorn de desplegament

Un cop vistes les tecnologies a utilitzar comencem a veure com serà l'entorn de desplegament de la nostra aplicació.

El nostre entorn de producció serà molt semblant al descrit en l'apartat de desenvolupament *Liferay*. El servidor *Liferay* estarà situat a la *LAN* interna i intercomunicat amb tots els altres sistemes per tal que els seus components puguin interactuar amb tots els elements i serveis definits anteriorment.

Principalment ens centrarem en la integració del *Liferay* en el Director Actiu LDAP i la base de dades corporativa. Disposem de dos controladors de domini (*Màster -Slave*) amb Windows que contenen el Directori Actiu. Per altra banda tenim un servidor de *OracleRAC* amb alta disponibilitat, un servidor físic a cada *CPD* connectat mitjançant fibra òptica, per tal de garantir la disponibilitat del servei en cas de fallada d'un node.

Disposem de sistemes de virtualització de servidors i de clients, tant *VMware* com *Citrix*. Per tant usarem aquests sistemes per tal de muntar l'entorn de desenvolupament de *Liferay*. Concretament usarem la virtualització amb *VMware*, creant una màquina virtual amb una versió *SUSE Linux Enterprice Server* per a *VMware*, aprofitant les avantatges multiplataforma que ens garanteix el *Liferay*.

Un cop definit l'entorn s'haurà d'elegir el servidor d'aplicacions o contenidor web de *Liferay* que s'adapti millor a les nostres necessitats. Ens hem decantat per usar el servidor *Tomcat* ja que és el més senzill d'utilitzar i és el més extens. També per a les nostres necessitats és el que més s'ajusta. En la fase d'implementació ja es veurà quines versions del producte s'utilitzaran i quin entorn de treball serà l'escollit.

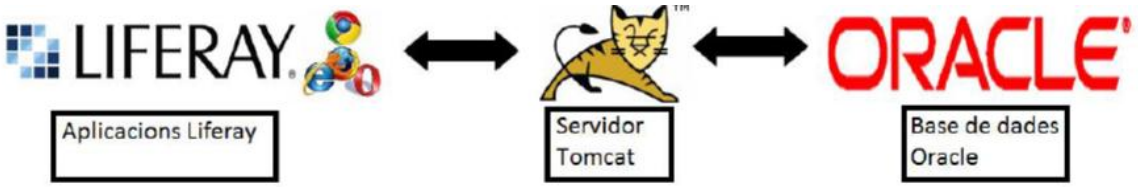


Figura 31: Interacció entre tecnologies de l'entorn de desplegament

Degut al gran abast i versatibilitat del *Liferay*, aquest sistema es pot anar ampliant i acabar integrant-se a molts de sistemes més del nostre entorn corporatiu. Com podria ser per exemple en el gestor documental *Alfresco*, en les xarxes socials corporatives, en el nostre servidor de correu intern, en la *NAS* de l'empresa, etc ...

5.1.2 Entorn de desenvolupament

Un cop definit l'entorn el qual és desplegarà la nostra aplicació anem a definir quin entorn de desenvolupament utilitzarem per a crear el nostre *porlet* d'aplicacions. S'ha escollit l'eina *Eclipse* degut a ser un dels *IDE's* més populars del mercat i a més a més destacar la seva capacitat d'integrar-se en eines, *SDK's*, etc.. *Liferay* no és la excepció i la versió 6 d'aquest incorpora *plugins* de desenvolupament amb *Eclipse*, facilitant el desenvolupament de *porlets*, *hooks* i *ext-plugins*. En la següent imatge es pot veure on quedaria situat el Liferay en aquesta arquitectura.

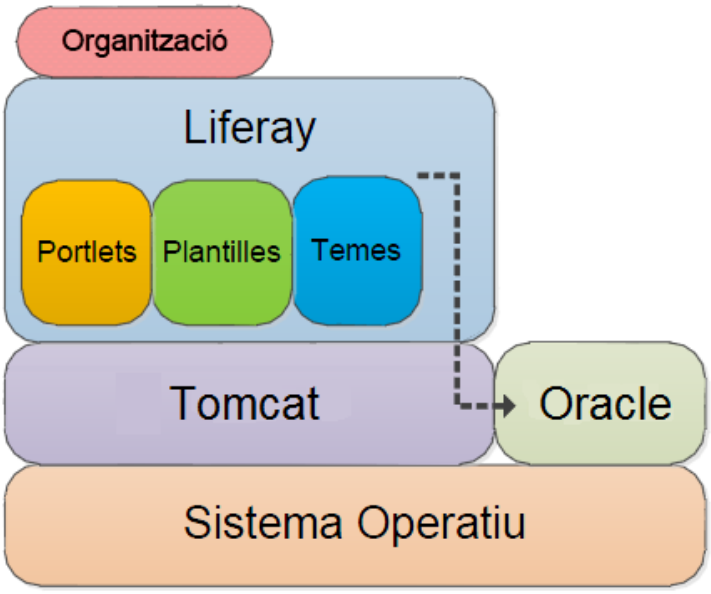


Figura 32: Arquitectura de l'entorn de desplegament

Per tant per a poder construir el nostre *porlet* necessitarem usar els *plugins* SDK que ens proporciona *Liferay* i els afegirem al *Eclipse*, juntament amb l'eina Ant, la qual realitzarà les tasques de compilació i acoblament. Aquesta imatge ens mostra la interacció del *Eclipse* amb *Java, Plugins SDK i ant* i amb el *plugins Liferay* que es poden desenvolupar.

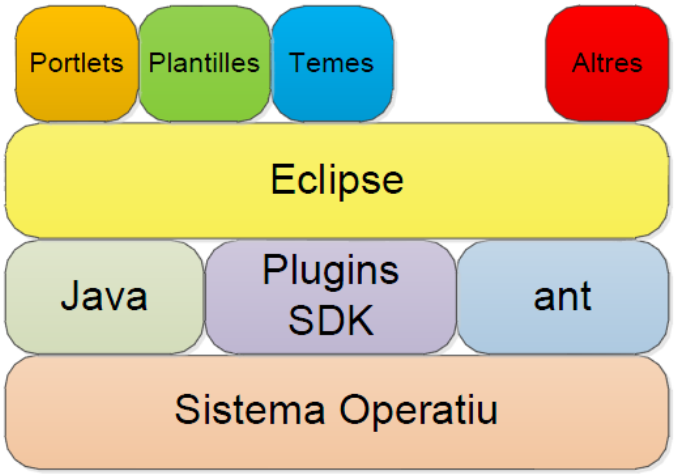


Figura 33: Arquitectura de l'entorn de desenvolupament

5.1.3 MVC del porlet a desenvolupar

Per a fer més robusta la nostra aplicació desenvoluparem el *porlet* usant un *framework MVC* combinat amb un de persistència. En l'apartat d'anàlisis de *frameworks* hem tractat diferents *frameworks* mostrant la seva potencialitat. En aquest projecte s'ha escollit usar-ne dos que combinats són molt potents: el *framework* web *Spring MVC* i el *framework* de persistència *Hibernate*. És una de les combinacions més usades per als desenvolupadors web avui en dia.

Serà *Hibernate* qui enllaci el objectes Java amb les taules de la base de dades i d'aquesta manera és *Hibernate* qui s'encarrega de la persistència de les dades. Aquest *framework* és complementa amb *Spring MVC* el qual permet injectar objectes ja instanciats dins d'altres objectes. En *Spring* la connexió a la base de dades ja està inclosa en objectes mapeijats per *Hibernate*.

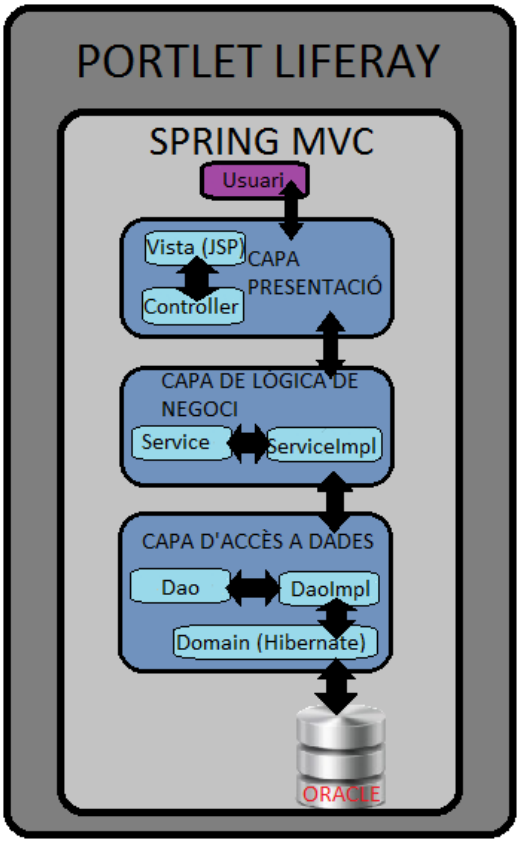


Figura 34: Arquitectura Spring MVC del nostre portlet

Com veiem aquest model *Spring MVC* amb *Hibernate* també es pot dividir amb tres capes:

- ✓ **Model:** és la capa encarregada d"encapsular tota la lògica de negoci de la nostra aplicació, la qual es pot subdividir en dues capes:
  - **Lògica de negoci:** Conté les classes *Java* per construir la capa de presentació, per tant és la que gestiona les peticions del controlador per donar una resposta d'acord amb la sol·licitud.
  - **Capa d'accés a dades (DAO):** són les classes *Java* que s'encarreguen de gestionar tota la interconnexió amb el sistema gestor de base de dades, també conté un *ORM* (tècnica de programació per convertir les dades dels llenguatges orientats a objectes en la seva representació en bases de dades relacionals). El motor de persistència *Hibernate* usa el *Domain* per a guardar la informació d'un objecte de manera permanent. Aquesta capa només es comunica amb la lògica de negoci.

- ✓ **Vista:** és la pàgina *HTML*, construïda amb *JSP* i usa tecnologies *Javascript* com ara *Jquery*, el codi proveeix de dades dinàmiques la pàgina. Aquest enviarà esdeveniments al controlador que els gestionarà. Per tant podem dir que és la resposta del controlador i el que se li presenta al usuari.
- ✓ **Controlador:** és l'eix central de la nostra arquitectura, s'encarrega de gestionar totes les peticions, validar els inputs rebuts i dirigir aquestes a la capa de lògica de negoci. Només es comunica amb la lògica de negoci i respon a través de la vista.

5.2 Diagrama de classes

En el següent diagrama de classes podem observar les quatre entitats que intervenen en la nostra aplicació:

- ✓ Aplicacio: Guarda les aplicacions definides per l'administrador.
- ✓ Group_: Estan definits els grups que hi ha dintre del *Liferay* (importats des de el Directori Actiu).
- ✓ User_: Estan definits els usuaris que hi ha dintre del *Liferay* (importats des del Directori Actiu).
- ✓ Users_Groups: entitat que associa els usuaris amb els grups.

Només mostrarem el diagrama de la classe Aplicacions amb la seva totalitat i les que afecten a ella sense especificar tots els atributs, s'entén que no cal ja que no estan realitzades per aquest treball al aprofitar-se de les de *Liferay*.

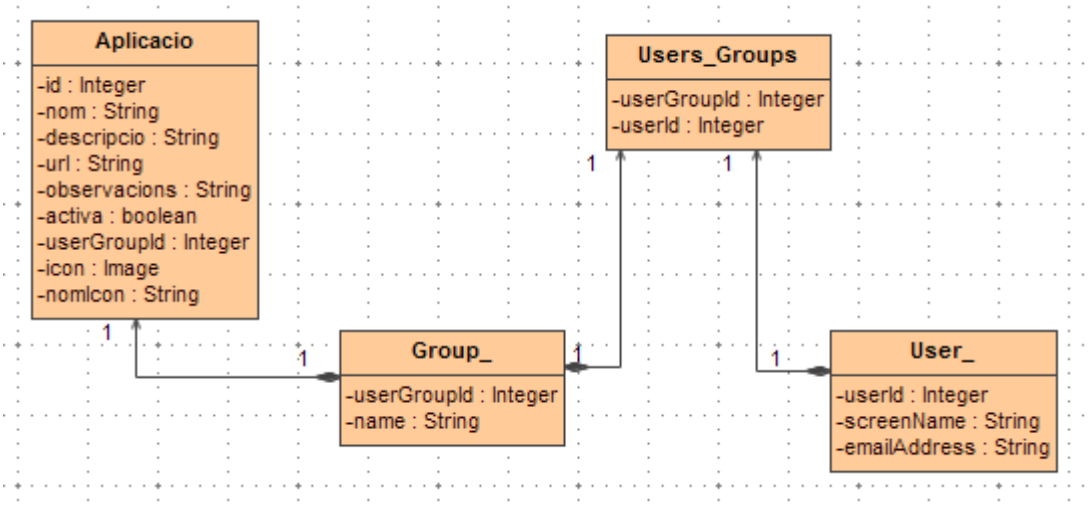


Figura 35: Diagrama de classes

Per tant l'única entitat creada per la nostra aplicació serà la classe Aplicacions, les altres 3 classes que ja existeixen al *Liferay* s'usaran per determinar qui té accés a les aplicacions. Un aplicació sempre pertany a un grup, el qual determinarà els usuaris que poden accedir a l'aplicació concreta. Els grups s'importaran del Directori Actiu, llavors la gestió dels permisos s'haurà de fer des de el mateix controlador del domini.

6 Implementació

6.1 Portlet Aplicacions

Com ja hem comentat anteriorment hem usat Spring Porlet MVC per tal de desenvolupar el porltlet aplicacions per al nostre portal. Quan s'usa Spring portlet en Liferay és pot implementar seguint una de les dues següents opcions:

- ✓ Spring porlet MVC + Liferay service Builder.
- ✓ Spring porlet MVC + implentació Spring DAO amb Hibernate.

Actualment a *Liferay* existeix una eina anomenada *'Service Builder'* que et permet delegar la creació de classes i mètodes necessaris per desenvolupar la lògica d'accés a les dades de forma automàtica, definint les entitats i els tipus de dades de la base de dades. Però en aquest desenvolupament no s'ha utilitzat ja que es perd el control d'aquesta lògica, perquè al ser *Liferay* qui et crea els arxius automàticament fa menys manejable les interaccions amb la base de dades. En canvi amb l'altra forma queda més clara la comunicació entre les capes del model vista controlador que és el que es pretenia.

Per tant s'ha escollit usar la segona opció, usar Spring Porlet MVC juntament amb la implementació Spring DAO amb Hibernate. A més a més aprofitarem l'avantatge que ens proporciona Spring al usar anotacions, llavors desenvoluparem el portlet utilitzant aquesta funcionalitat.

En resum, usarem les següents coses per tal de desenvolupar el nostre portlet:

- ✓ **Spring Portlet MVC**
- ✓ **Spring Portlet DAO amb Hibernate**
- ✓ **Anotació d'Hibernate i l'Anotació de Spring**
- ✓ **Java Persistence API (JPA)**
- ✓ **Jquery JGrid**

Tot seguit podem veure l'estructura de carpetes i fitxers del desenvolupament del nostre portlet:

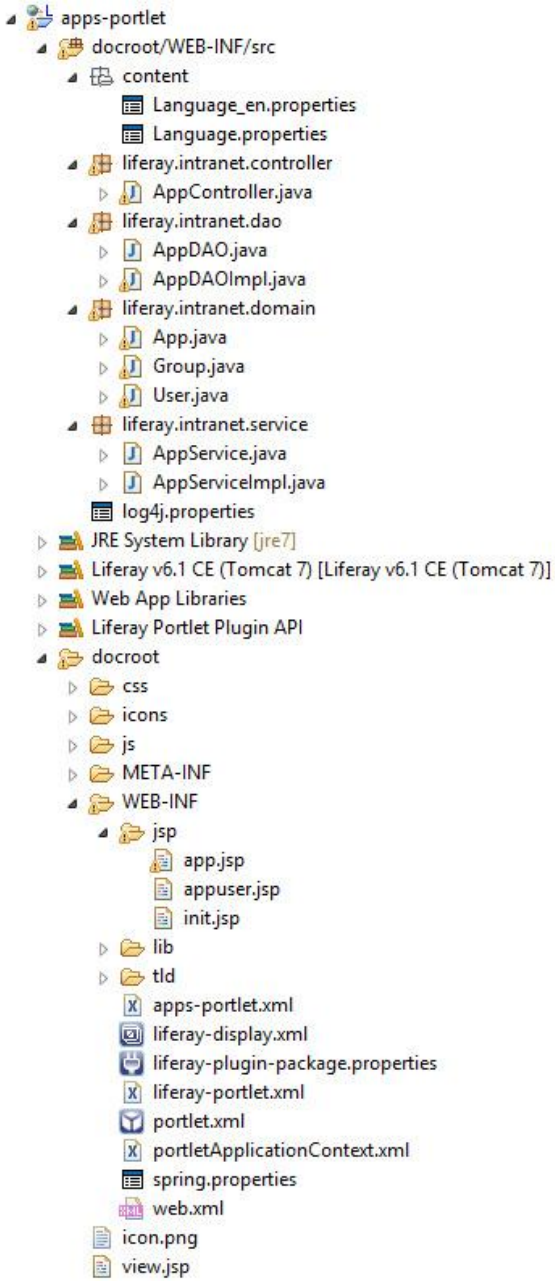


Figura 36: Estructura de directoris del nostre portlet

6.1.1 Configuració

El portal Liferay disposa d'arxius jar dels frameworks Spring i Hibernate, per tant o bé s'utilitzen els propis jars que ja et proporciona Liferay o bé es poden buscar els últims Spring jars i usar-los. S'ha optat per usar les pròpies llibreries que ja et proporciona el portal Liferay ja que el tamany de l'aplicació esdevindrà més petit que usant llibreries externes.

Per afegir els arxius jar del portal Liferay al nostre porlet ho hem fet a través de les propietats de l'arxiu *liferay-plugin-package.properties*:

liferay-plugin-package.properties

```
name=App
module-group-id=liferay
module-incremental-version=1
tags=
short-description=
change-log=
page-url=http://www.liferay.com
author=Liferay, Inc.
licenses=LGPL
portal-dependency-jars=\
    jstl-api.jar,\
    jstl-impl.jar,\
    commons-fileupload.jar,\
    commons-io.jar,\
    antlr3.jar,\
    aopalliance.jar,\
    c3p0.jar,\
    cglib.jar,\
    commons-beanutils.jar,\
    commons-collections.jar,\
    commons-digester.jar,\
    dom4j.jar,\
    hibernate3.jar,\
    javassist.jar,\
    slf4j-api.jar,\
    saaj-api.jar,\
    spring-aop.jar,\
    spring-beans.jar,\
    spring-context.jar,\
    spring-core.jar,\
    spring-expression.jar,\
    spring-jdbc.jar,\
    spring-orm.jar,\
    spring-web-portlet.jar,\
    spring-web.jar,\
    xml-apis.jar,\
    spring-transaction.jar,\
    spring-web-servlet.jar,\
    spring-context-support.jar,\
    spring-asm.jar
portal-dependency-tlds=c.tld
long-description=
liferay-versions=6.2.0+
```

Figura 37: Llibreries utilitzades per al desenvolupament del portlet

Un cop ja tenim les llibreries necessàries es configuraran els arxius *XML* necessaris per al correcte desplegament del portlet.

- ✓ **web.xml:** Arxiu on es configuren les propietats del servidor, juntament amb els possibles recursos de la base de dades. A aquest fitxer s'han d'afegir les següents línies de codi XML, per establir on emmagatzemarem el fitxer de configuració del Spring del nostre portlet amb el paràmetre "*contextConfigLocation"* en aquest cas serà a la carpeta *'WEB-INF/context'* amb el nom de *"applicationContext.xml"*, arxiu que crearem a continuació. Després s'ha de crear el *"listener"* i el *"servlet"* per poder treballar amb *Spring MVC*, on el *servlet* serveix per representar el contingut, és a dir , representar la vista del *portlet* i el *listener* carregarà totes les configuracions 'context' del *portlet*.

```
<display-name>apps-portlet</display-name>

<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/portletApplicationContext.xml</param-value>
</context-param>
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<servlet>
    <servlet-name>ViewRendererServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.ViewRendererServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>ViewRendererServlet</servlet-name>
    <url-pattern>/WEB-INF/servlet/view</url-pattern>
</servlet-mapping>
```

- ✓ **spring.propierties:** arxiu de propietats fer la connexió amb la base de dades.

```
# Oracle
#database properties
jdbc.default.driverClassName=oracle.jdbc.driver.OracleDriver
jdbc.default.url=jdbc:oracle:thin:@localhost:1521:XE
jdbc.default.username=LIFERAY
jdbc.default.password=LIFERAY
#hibernate properties
hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
```

- ✓ **portletApplicationContext.xml** : arxiu que conté la configuració Spring de l'aplicació i que es referenciat des de l'arxiu *web.xml*. En aquest arxiu és configuren les següents parts:

    - *viewResolver* : gestor de les vistes de l'aplicació.

```
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass"
        value="org.springframework.web.servlet.view.JstlView" />
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
</bean>
```

    - *spring.properties*: es referència l'arxiu, abans mencionat, que té les propietats de connexió a la base de dades i s'agafen les dades necessàries.

```
<context:property-placeholder location="/WEB-INF/spring.properties" />
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName">
        <value>${jdbc.default.driverClassName}</value>
    </property>
    <property name="url">
        <value>${jdbc.default.url}</value>
    </property>
    <property name="username">
        <value>${jdbc.default.username}</value>
    </property>
    <property name="password">
        <value>${jdbc.default.password}</value>
    </property>
</bean>
```

    - Configurar *l'Hibernate* per a fer el mapeig d'atributs entre la base de dades relacional i els objectes de l'aplicació, s'establirà que la classe *Java* on s' ha de realitzar el mapeig s'ubicarà al paquet *'intranet.liferay.domain'*.

```
<bean id="sessionFactory"
    class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="packagesToScan">
        <list>
            <value>liferay.intranet.domain</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">${hibernate.dialect}</prop>
            <prop key="hibernate.show_sql">false</prop>
            <prop key="hibernate.hbm2ddl.auto">update</prop>
        </props>
    </property>
</bean>
```

- Referenciar les classes DAO i les classes Services

```
<context:component-scan base-package="liferay.intranet.dao" />
<context:component-scan base-package="liferay.intranet.service" />
```

- *portletMulipartResolver:* s'utilitza per a poder pujar les icones de les aplicacions.

```
<bean id="portletMultipartResolver"
    class="org.springframework.web.portlet.multipart.CommonsPortletMultipartResolver">
    <!-- one of the properties available; the maximum file size in bytes -->
    <property name="maxUploadSize" value="100000"/>
</bean>
```

- ✓ **portlet.xml** : arxiu del nostre portlet on es defineixen les seves característiques. Es necessita definir el nostre *FrontController*, que serà el *DispatcherPortlet*. I també li tenim que dir on tenim definit el nostre fitxer '*application context'*.

```
<portlet-name>apps</portlet-name>
<display-name>Apps</display-name>
<portlet-class>org.springframework.web.portlet.DispatcherPortlet</portlet-class>
<init-param>
    <name>contextConfigLocation</name>
    <value>/WEB-INF/apps-portlet.xml</value>
</init-param>
```

- ✓ **App-portlet.xml**: aquest arxiu s'ha referenciat al portlet.xml i contindrà els controladors que necessitem per configurar la gestió del portlet, per a que puguen invocar per la seva url.

```
<context:annotation-config />
<tx:annotation-driven />
<context:component-scan base-package="liferay.intranet.controller" />
<bean id="portletModeHandlerMapping"
    class="org.springframework.web.portlet.handler.PortletModeHandlerMapping">
    <property name="portletModeMap">
        <map>
            <entry key="view">
                <ref bean="appController" />
            </entry>
    <!--  -->

    <!--  -->
        </map>
    </property>
</bean>
```

- ✓ Configuració dels idiomes: arxius per definir les etiquetes dels diferents idiomes de l'aplicació. Es poden crear tants arxius d'idiomes com faci falta.
    - **Language_properties** i **Language_en_properties.**

```
label.id_app=Id App
label.nom=Nom
label.descripcio=Descripci&oacute
label.url=URL
label.observacions=Observacions
label.icon=Icona
label.nomicon=Nom icona
label.usergroupid=UserGroupId
label.actiu=Actiu
label.addapp=Add App
label.updateapp=Update App


label.menu=Menu
label.title=App Manager
label.footer=&copy; JordiTolosa
```

6.1.2 Capa d'accés a les dades

- ✓ **Domain** : Primer de tot s'haurà de definir les classes del objectes que volem realitzar el mapeig amb *Hibernate* anomenades *domain*. Aquestes usaran les anotacions JPA d'Hibernate ja esmentades anteriorment.

```java
@Entity //defeix la classe com una entitat
@Table(name="APPS")//nom de la taula a la base de dades
public class App implements Serializable {
    @Id //clau primària
    @Column(name="ID_APP") //nom de l'atribut
    @GeneratedValue//valor autonumèric
    private Integer id_app;

    @Column(name="NOM")
    private String nom;

    @Column(name="DESCRIPCIO")
    private String descripcio;

    @Column(name="URL")
    private String url;

    @Column(name="OBSERVACIONS")
    private String observacions;

    @Column(name="NOMICON")
    private String nomIcon;

    @Column(name="USERGROUPID")
    private Integer usergroupid;

    @Column(name="ICON")
    @Lob
    CommonsMultipartFile icon;

    @Type(type="yes_no")
    private boolean actiu;

    @OneToOne   //relació 1a1 entre una aplicació i un grup
    @JoinColumn(name = "USERGROUPID", insertable=false, updatable=false)
    private Group mgroup;
```

- ✓ **DAO**: Aquesta part s'encarregarà de fer les interaccions amb la base de dades mitjançant el mapeig del 'domain' definit anteriorment. Aquesta usa les anotacions del framework Spring.

```java
@Repository //indica que és una classe relacionada en la capa de persitència i usa patró Singleton
public class AppDAOImpl implements AppDAO {
    @Autowired //s'encarrega de cercar un bean de la classe corresponent
    private SessionFactory sessionFactory;

    public void addApp(App app) {
        sessionFactory.getCurrentSession().save(app);
    }
```

6.1.3 Capa Lògica de Negoci

En aquesta capa intermitja controlarem la lògica de negoci del nostre portlet, per fer-ho necessitem definir el *service* el qual realitzarà la comunicació entre el *controlador* i el *DAO*. Aquesta també usa les anotacions del *framework Spring*.

```
@Service //s'especifica que aquesta classe contindrà la lògica de negoci
public class AppServiceImpl implements AppService {
    @Autowired //s'encarrega de cercar un bean de la classe corresponent
    private AppDAO appDAO;

    @Transactional //s'especifica quins mètodes vols que s'executen en una sola transacció
    public boolean isAdmin(Integer idUser){
        return appDAO.isAdmin(idUser);
    }
```

6.1.4 Capa Presentació

- ✓ **Controlador**: S'implementa el controlador el qual gestiona les peticions del portlet. Al controlador cada mètode es invocat pel "*Request Parameter mapping*". Aquesta també usa les anotacions del framework Spring, on usem l'anotació *Autowired* per interactuar amb la base de dades.

```
@Controller("appController")//controlador gestionat per Spring el qual crea el bean associat al fitxer de configuració
@RequestMapping("VIEW")//el controlador s'encarrega de gestionar la vista del portlet
public class AppController  {

    @Autowired //s'encarrega de cercar un bean de la classe corresponent
    private AppService appService;

    @RenderMapping//indica a Spring que te que cercar un .jsp dins de la carpeta establerta amb aquest nom, per mostrar-lo a la vista del portlet
    public String listApps(Map<String, Object> map) throws com.liferay.portal.kernel.exception.PortalException, com.liferay.portal.kernel.exception.SystemException {
```

Només hi ha un mètode, listApps, el qual utilitza el *@RenderMapping*, el qual s'usarà per a refrescar la pàgina. Els altres mètodes usaran tots el *@ActionMapping* per realitzar les diferents accions. Alguns usaran *@ModelAttribute* per tal de mapejar l'objecte sencer (mètode *addApp*) i altres només usaran el *@RequestParam* per passar algun atribut (mètode *deleteApp*).

```
@ActionMapping(params = "action=add")//rep les peticions dels usuaris, executen la lògica associada i estableixen quin mètode Render serà el seguent en ejecutar-se
public void addApp(ActionRequest actionRequest,ActionResponse actionResponse, Model model, @ModelAttribute("app") App app,  BindingResult result) throws IOException,
        PortletException {
```

```
@ActionMapping(params = "action=delete")
public void deleteApp(@RequestParam("appId") Integer appId,ActionRequest actionRequest,ActionResponse actionResponse, Model model) throws IOException,
        PortletException {
    appService.removeApp(appId);
}
```

- ✓ **JSP**: ja finalment només ens quedaria crear la vista usant una pàgina *JSP*. S'ha usat les etiquetes de formularis les quals tenen la capacitat de recuperar les dades dels models d'objectes. A més també s'ha usat el "*portlet URL*" per invocar les diferents accions del controlador amb les etiquetes *JSTL*.

```
<!-- Es crea el conjunt d'accions d'Spring per enviar l'informació al Controlador -->
<portlet:actionURL var="addAppURL">
<portlet:param name="action" value="add"></portlet:param>
</portlet:actionURL>
<portlet:actionURL var="deleteAppURL">
<portlet:param name="action" value="delete"></portlet:param>
</portlet:actionURL>
<portlet:actionURL var="consultAppURL">
<portlet:param name="action" value="consult"></portlet:param>
</portlet:actionURL>
<portlet:actionURL var="updateAppURL">
<portlet:param name="action" value="update"></portlet:param>
</portlet:actionURL>
```

S'ha decidit usar *jqGrid* per tal de mostrar la llista d'aplicacions, el qual és un control basat en *jquery* per realitzar operacions bàsiques (CRUD). La principal avantatge de treballar amb aquest control, és que, al ser creat en *jquery*, s'executa a la part de client el qual el fa més eficient.

```
<!-- taula amb la llista de aplicacions -->
<h3>Gestió Aplicacions</h3>
<%{%>
<c:if  test="${!empty appList}">
    <!-- class="row-border hover order-column" -->
    <div id="customers">
    <table id="tablaFormat">
        <thead>
        <tr>
            <th>Id_app</th>
            <th>Nom</th>
            <th>Descripció</th>
            <th>Url</th>
            <th>Nom Grup</th>
            <th>Observacions</th>
            <th>Actiu</th>
            <th>#</th>
            <th>#</th>
        </tr>
        </thead>
        <tbody>
        <c:forEach items="${appList}" var="app">
            <tr>
                <td>${app.id_app}</td>
                <td>${app.nom}</td>
                <td>${app.descripcio}</td>
                <td>${app.url}</td>
                <td>${app.mgroup.name}</td>
                <td>${app.observacions}</td>
                <td>${app.actiu}</td>
                <td class="center"><a href="${deleteAppURL}&appId=${app.id_app}" onclick="return confirmar('Esta segur que vol eliminar el registre?')">
                <td class="center"><a href="${consultAppURL}&appId=${app.id_app}"><liferay-ui:icon image="edit" message="editar registre"/></a></td>
            </tr>
        </c:forEach>
        </tbody>
        <tfoot>
```

6.2 Portlet Wiki

Aquest porltet ja ve predefinit amb el portal Liferay, llavors l'únic que s'ha de fer és afegir-lo a una pàgina i després és pot configurar, tant a nivell d'opcions de funcionalitat de Wiki com a permisos d'ús dels diferents usuaris.

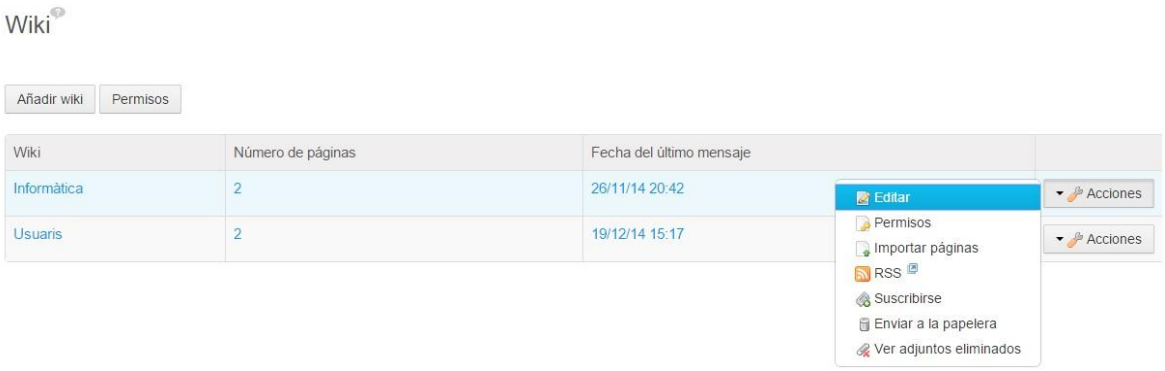


Figura 38: El nostre portlet Wiki al Liferay

Per tant, seguint la idea inicial del projecte, s'han creat dos Wikis diferents, una per al departament d'informàtica, només visible pels seus membres; i una altra d'usuaris, la qual serà visible per a tots els usuaris de l'ajuntament. Inicialment estarà enfocada per a temes estricaments d'informàtica, sent només el departament d'informàtica qui gestionarà els diferents fils i els usuaris només podran consultar els temes de la Wiki Usuaris, servint-los d'auto ajuda i dotant-los de més autonomia. Ja a l'apartat manual de l'usuari és podrà veure alguna captura de com es mostraria aquest portlet per pantalla. Més endavant, si es creu oportú es podrà dotar de permisos a usuaris per tal de poder afegir continguts a la wiki d'Usuaris.

6.3 Base de dades

A nivell de base de dades s'usa l'esquema definit per a Liferay ja que com en el nostre portlet només s'ha de crear una taula nova no s'ha considerat necessari crear un nou esquema per a la intranet. Per tant quan es fa el *deploy* del nostre portlet es crea la taula *Apps* dins d'aquest esquema i també es creen les diferents relacions definides dintre de les classes d*omain* amb les anotacions JPA d'*Hibernate*.

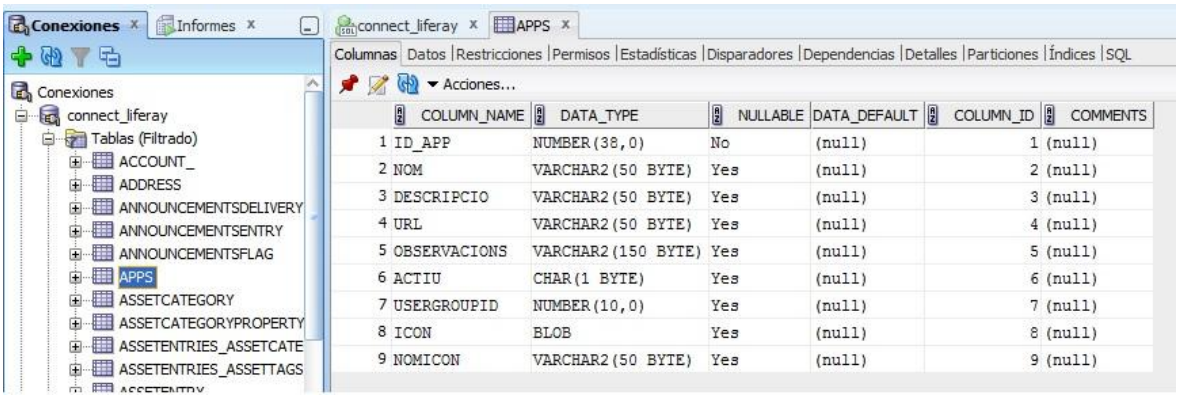


Figura 39: La taula Apps definida entre les taules del liferay, usant el SQLDeveloper

Tot seguit es mostra l'axiu sql que seria l'equivalent a aquesta taula.

CREATE TABLE apps(
id_app int,
nom varchar2(50),
descripcio varchar2(50),
url varchar2(50),
usergroupid int,
observacions varchar2(150),
icon BLOB,
nomicon varchar2(50),
actiu char(1) check(actiu in ('N', 'Y')),
primary key(id_app));

En els atributs d'hibernate, com ja hem comentat anteriorment, s'ha definit les diferents relacions entre taules.

- ✓ Relació Apps – Groups (One to One)

```
@OneToOne(mappedBy = "mgroup")
private App mapp;
```

```
@OneToOne   //relació 1a1 entre una aplicació i un grup
@JoinColumn(name = "USERGROUPID", insertable=false, updatable=false)
private Group mgroup;
```

En aquesta relació indiquem que una aplicació ha de pertànyer a un grup per tal de poder vincular quins usuaris tindran accés a aquesta aplicació.

- ✓ Relació Users – Groups (Many to Many)

```
@ManyToMany(mappedBy = "users")
private Set<Group> groups = new HashSet<Group>();


@ManyToMany(cascade = CascadeType.ALL)
@JoinTable(
        name = "USERS_USERGROUPS",
        joinColumns = @JoinColumn(name = "USERGROUPID"),
        inverseJoinColumns = @JoinColumn(name = "USERID")
)
private Set<User> users = new HashSet<User>();
```

Llavors tenim que un grup té una llista de usuaris i un usuari també pertany a un conjunt de grups. Aquesta relació no et crea cap taula nova, sinó que aprofita una que ja està definida al liferay que ja relaciona aquestes dos taules amb les seves claus primàries.

6.4 Integracions

6.4.1 LDAP

S'ha realitzat la integració amb el Directori Actiu de l'ajuntament, realitzant la importació dels grups necessaris per al portlet aplicacions i alguns dels usuaris que utilitzaran les aplicacions. També servirà per a tots els usuaris que usaran el portlet Wiki. La configuració s'ha pogut fer mitjançant el panel de control del mateix portal.

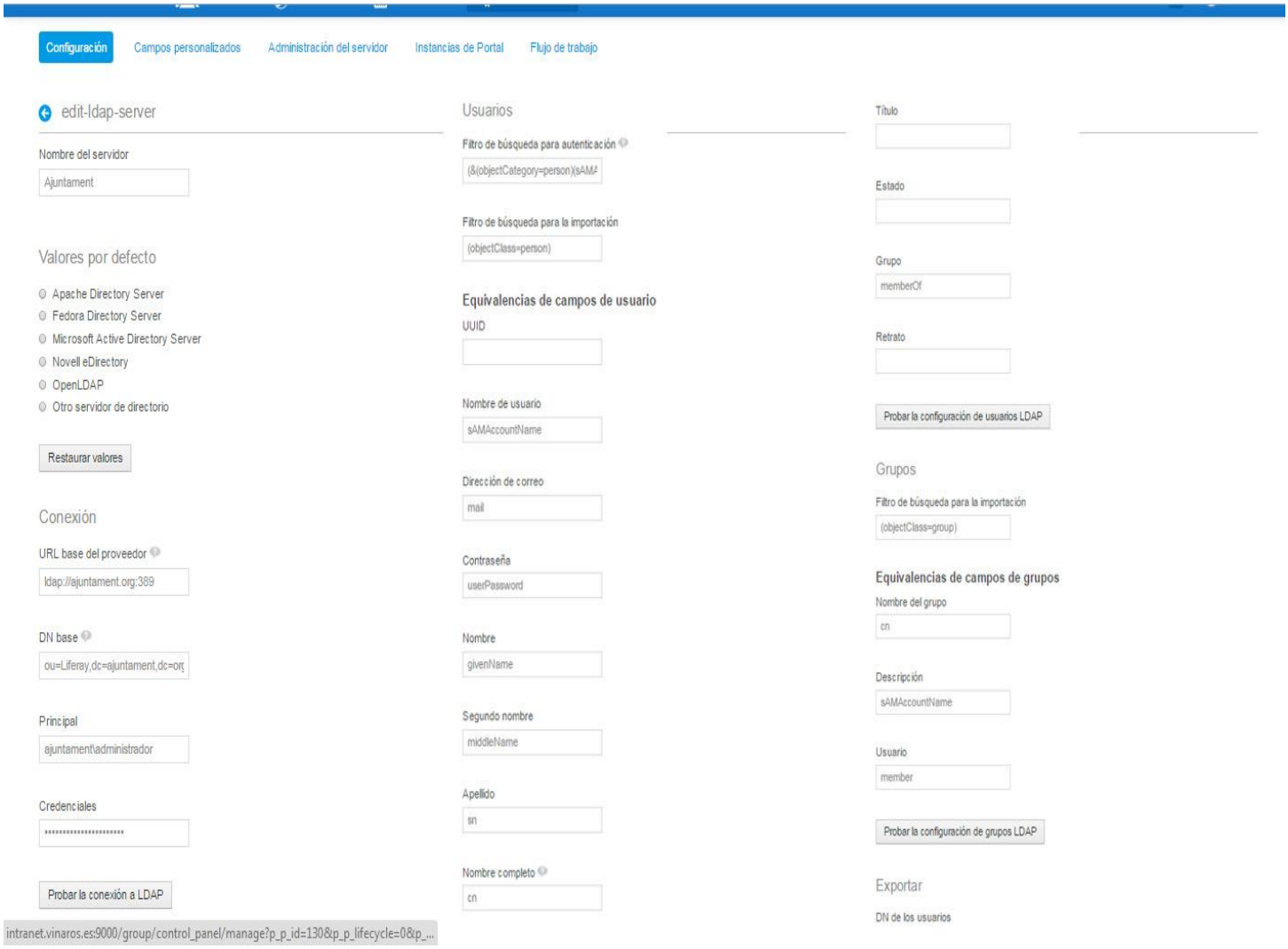


Figura 40: Configuració LDAP al panel de control per integrar en el Directori Actiu

Tot seguit podem veure els usuaris i grups que hi ha definits al Controlador de Domini dintre Directori Actiu els quals s'han usat per a la importació.

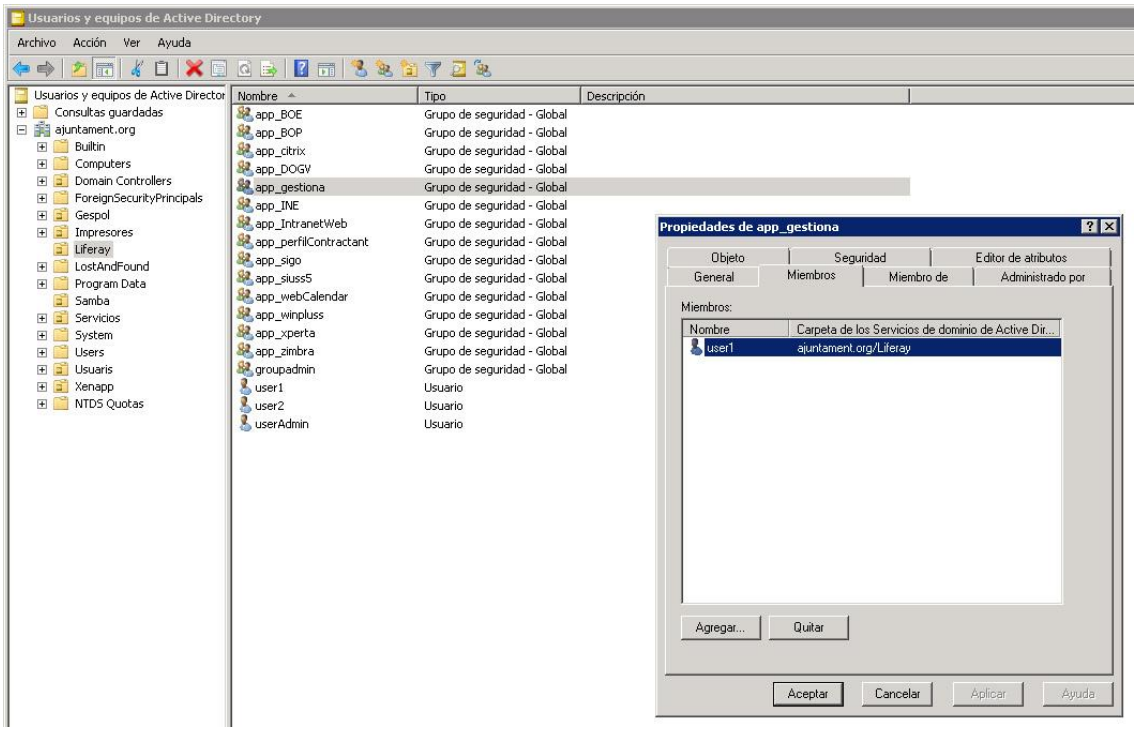


Figura 41: Els usuaris i grups definits al DC dintre del directori actiu

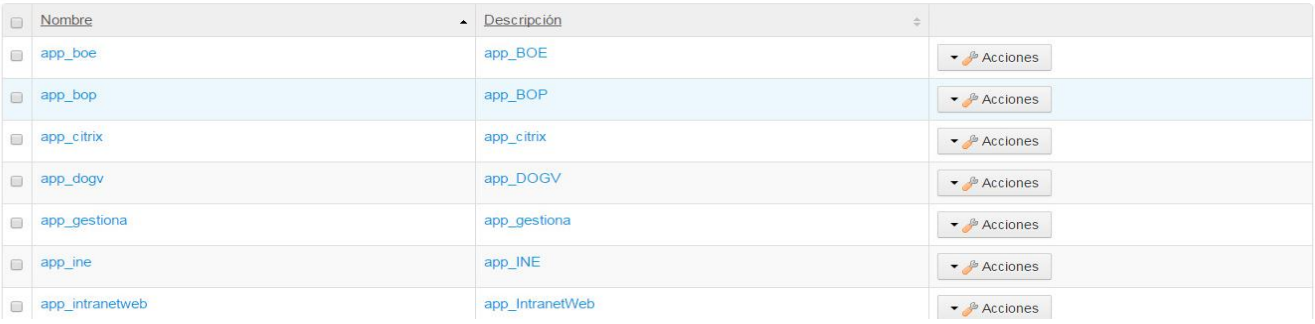
Un cop realitzada la importació dels usuaris i dels grups del LDAP es poden gestionar des del mateix panel de control del portal Liferay.

- ✓ Llistat d'usuaris

Nombre	Apellido	Nombre de usuario	Título	Organizaciones	Grupos de usuario	
zimbra	zimbra	zimbra				Acciones
userAdmin	userAdmin	useradmin			groupadmin	Acciones
user2	user2	user2			app_boe	Acciones
user1	user1	user1			app_boe, app_bop, app_dogv, app_citrix, app_xperta, app_perfilcontractant, app_intranetweb, app_ine, app_webcalendar, app_zimbra, app_gestiona, app_winpluss, app_siuss5, app_sigo	Acciones
Test	Test	test			groupadmin	Acciones

Figura 42: Resultat de la importació LDAP dels usuaris al Liferay

- ✓ Llistat de grups



Nombre	Descripción	
app_boe	app_BOE	Acciones
app_bop	app_BOP	Acciones
app_citrix	app_citrix	Acciones
app_dogv	app_DOGV	Acciones
app_gestiona	app_gestiona	Acciones
app_ine	app_INE	Acciones
app_intranetweb	app_IntranetWeb	Acciones

Figura 43: Resultat de la importació LDAP dels grups al Liferay

6.4.2 OracleRAC

Per a fer la connexió amb l'OracleRAC s'ha de canviar l'arxiu de connexió *url* del *jdbc*. Enlloc d'apuntar a l'oracle Express instal·lat a l'entorn de desenvolupament, s'ha d'usar l'altra connexió per a l'entorn de desplegament.

Llavors al fitxer de configuració **spring.properties** s'ha de canviar la següent línia:

jdbc.default.url =

jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=node1oraclerac)(PORT=1521))(ADDRESS=(PROTOCOL=TCP)(HOST=node2oraclerac)(PORT=1521))(LOAD_BALANCE=yes)(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=ORCL)))

7 Conclusions

En primer lloc el fet d'haver escollit desenvolupar un portlet amb Liferay m'ha permès endinsar-me en el món dels gestors de continguts i ampliar els coneixements que disposava sobretot de Liferay abans de la realització del present PFC. M'ha sorprès el gran potencial de Liferay, el qual comparteix posició de lideratge en el mercat de portals amb empreses com IBM o Oracle, però cobrint els requeriments del projecte amb una inversió molt menor.

Considero que el projecte era ambiciós des de un principi, ja que encara que tenia coneixements de Java no disposava d'experiència amb cap gestor de continguts, i el temps per desenvolupar un PFC és curt i necessitava una corba d'aprenentatge ràpida. Per aquest motiu desenvolupar el projecte partint des de zero no ha estat fàcil, inicialment posant èmfasis en integrar els coneixements de les últimes assignatures de la carrera, sobretot les d'enginyeria de programari, amb el portal Liferay per tal de desenvolupar una aplicació robusta i sobretot pràctica, que en acabat el projecte pugues entrar a un entorn de producció real. Aquest últim punt m'ha motivat bastant i aconseguit que tot plegat sigui una experiència molt gratificant per mi.

Els objectius plantejats inicialment s'han pogut aconseguir. Per una banda la part més enfocada al desenvolupament, la construcció d'un portlet a mida usant el patró de disseny J2EE Model Vista Controlador i també la configuració d'un portlet Wiki. Per l'altra banda la part enfocada al desplegament de l'entorn, on s'ha creat un entorn de producció dintre de l'ajuntament, s'ha desplegat el Liferay integrat-lo amb altres productes com són el OracleRAC i el Directory Actiu.

Per concloure espero que aquest projecte hagi estat la primera pedra de la intranet de l'ajuntament i que en un futur pròxim es continuen creant més aplicacions. Com ja hem comentat abans, Liferay s'integra en moltes tecnologies i productes existents al mercat, i com a treballs futurs es podria plantejar integrar-lo amb el servidor de correu Zimbra i per exemple amb el gestor documental Alfresco. A més a més té una altra part que considero molt atractiva, la del treball col·laboratiu, on existeix una suite anomenada SocialOffice que s'integra al Liferay i proporciona un conjunt d'eines per al treball col·laboratiu.

8 Bibliografia

[1] Liferay Portal 6.2 Developer's Guide: http://www.liferay.com/es/documentation/liferay-portal/6.2/development, consultat novembre 2014.

[2] Using Liferay Portal 6.2: http://www.liferay.com/es/documentation/liferay-portal/6.2/user-guide, consultat novembre 2014.

[3] Portlet Development - Development | Liferay. Available at: https://www.liferay.com/es/documentation/liferay-portal/6.0/development/-/ai/portlet-development, consultat novembre 2014.

[4] The Plugins SDK - Development | Liferay. Available at: http://www.liferay.com/es/documentation/liferay-portal/6.1/development/-/ai/the-plugins-s-3, consultat novembre 2014.

[5] Sezov, R. (2012). Liferay in action, consultat octubre 2014.

[6] Java Community Process. JSR-000286 Portlet Specification, (Final Release) http://jcp.org/aboutJava/communityprocess/final/jsr168/index.html, consultat novembre 2014.

[7] Java Community Process. JSR-000168 Portlet Specification, (Final Release) (Març-Abril 2013) http://jcp.org/aboutJava/communityprocess/final/jsr286/index.html, consultat novembre 2014.

[8] Bauer, C., & King, G. (2005). Hibernate in action, consultat novembre 2014.

[9] Hibernate ORM - Hibernate ORM. http://hibernate.org/orm/, consultat novembre 2014.

[10] Walls, C., & Breidenbach, R. (2005). Spring in action. Dreamtech Press, consultat octubre 2014.

[11] Spring. http://spring.io/ , consultat novembre 2014.

[12] jQuery. http://jquery.com/, consultat desembre 2014.

[13] Oracle JDBC Drivers release 10.2.0.1.0 Downloads. http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-10201-088211.html, consultat novembre 2014.

[14] Oracle SQL Developer. Available at: http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html, consultat novembre 2014.

A. Instal·lació i desplegament dels portlets

Primerament s'ha de configurar un entorn de desplegament per a que es puguin executar els dos portlets descrits a aquest projecte, el d'aplicacions i la Wiki. Per a fer-ho necessitarem inicialment uns requisits tècnics per poder desplegar aquests portlets. Tot seguit es detallaran els passos per poder configurar l'entorn de proves, instal·lant els productes necessaris i amb la configuració corresponent. Després configurarem el portal Liferay per a que pugui desplegar aquests dos portlets. I ja per acabar podrem desplegar-los i tenir-los disponibles per a fer proves.

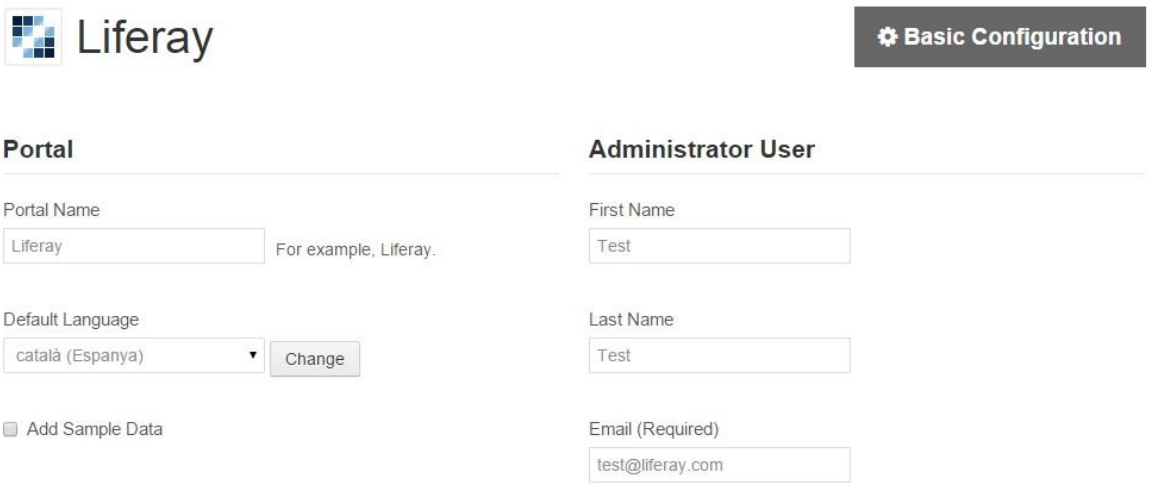
- ✓ **Requisits tècnics**
  - Java JDK 7
  - Oracle Express 10g (versió gratuïta)
  - Liferay 6.2 (en qualsevol servidor d'aplicacions)

- ✓ **Configuració de l'entorn de proves**
  - **Instal·lar Oracle Express 10g**: realitzar la instal·lació per defecte.
    - Configurar esquema Liferay: executarem el següent script connectats com a usuari System. Aquest script, *esquema_liferay.sql,* estarà ubicat dintre de la carpeta sql.

```
drop user LIFERAY cascade;
drop tablespace LIFERAYDAT including contents and datafiles;
drop tablespace LIFERAYIDX including contents and datafiles;


CREATE TABLESPACE "LIFERAYDAT" LOGGING
DATAFILE 'C:\oraclexe\oradata\XE\liferaydat.dbf' SIZE 1024M
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO;

CREATE TABLESPACE "LIFERAYIDX" LOGGING
DATAFILE 'C:\oraclexe\oradata\XE\liferayidx.dbf' SIZE 512M
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO;

CREATE USER "LIFERAY" PROFILE "DEFAULT" IDENTIFIED BY "LIFERAY"
DEFAULT TABLESPACE "LIFERAYDAT" TEMPORARY TABLESPACE "TEMP"
ACCOUNT UNLOCK;

GRANT "CONNECT" TO "LIFERAY";
GRANT "RESOURCE" TO "LIFERAY";

GRANT ALTER ANY INDEX TO "LIFERAY";
```

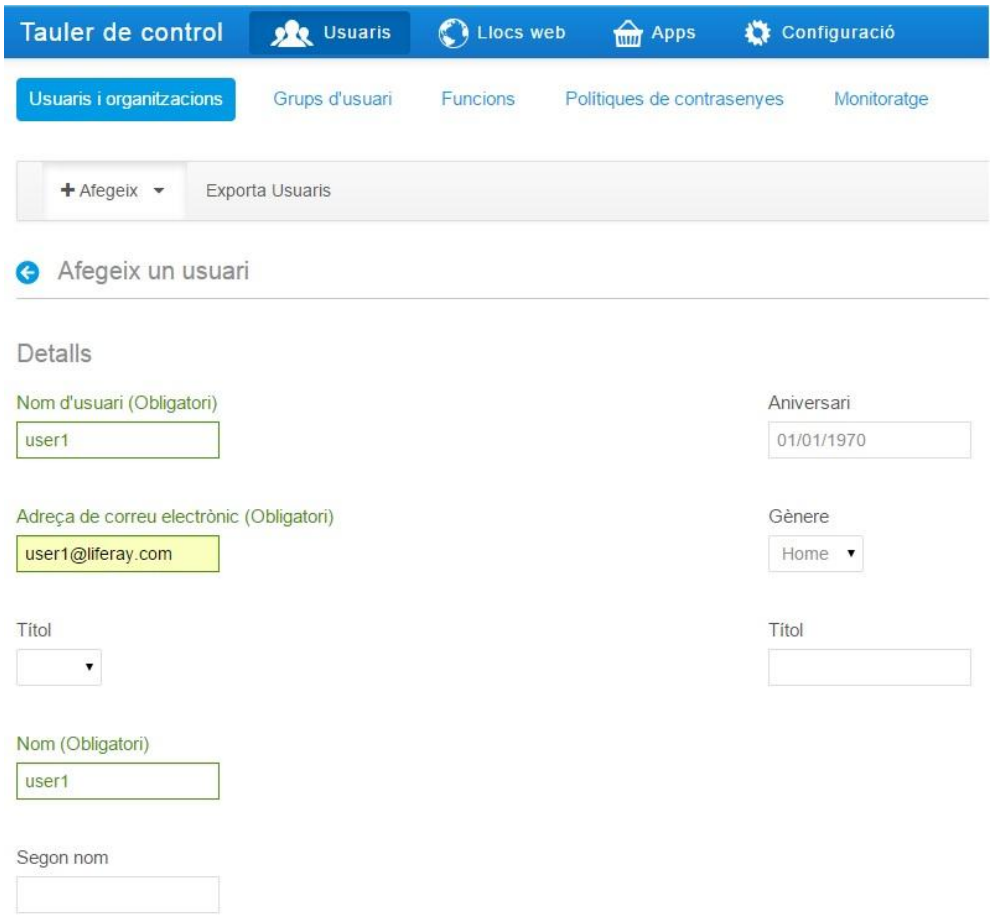
```
GRANT ALTER ANY SEQUENCE TO "LIFERAY";
GRANT ALTER ANY TABLE TO "LIFERAY";
GRANT ALTER ANY TRIGGER TO "LIFERAY";
GRANT CREATE ANY INDEX TO "LIFERAY";
GRANT CREATE ANY SEQUENCE TO "LIFERAY";
GRANT CREATE ANY SYNONYM TO "LIFERAY";
GRANT CREATE ANY TABLE TO "LIFERAY";
GRANT CREATE ANY TRIGGER TO "LIFERAY";
GRANT CREATE ANY VIEW TO "LIFERAY";
GRANT CREATE PROCEDURE TO "LIFERAY";
GRANT CREATE PUBLIC SYNONYM TO "LIFERAY";
GRANT CREATE TRIGGER TO "LIFERAY";
GRANT CREATE VIEW TO "LIFERAY";
GRANT DELETE ANY TABLE TO "LIFERAY";
GRANT DROP ANY INDEX TO "LIFERAY";
GRANT DROP ANY SEQUENCE TO "LIFERAY";
GRANT DROP ANY TABLE TO "LIFERAY";
GRANT DROP ANY TRIGGER TO "LIFERAY";
GRANT DROP ANY VIEW TO "LIFERAY";
GRANT INSERT ANY TABLE TO "LIFERAY";
GRANT QUERY REWRITE TO "LIFERAY";
GRANT SELECT ANY TABLE TO "LIFERAY";
GRANT UNLIMITED TABLESPACE TO "LIFERAY";


alter system set processes=300 scope=spfile;
```

- **Instal·lació Liferay 6.2**, per exemple usarem el *bundle with Tomcat*
  - Descargar-lo de la pàgina oficial de Liferay.
  - Descomprimir-lo.
  - Copiar **ojdbc14.jar** dins de la carpeta *tomcat/lib/ext*.
  - Engegar el Tomcat.
  - Obrir el navegador (localhost)
    - Seleccionar dades bàsiques.

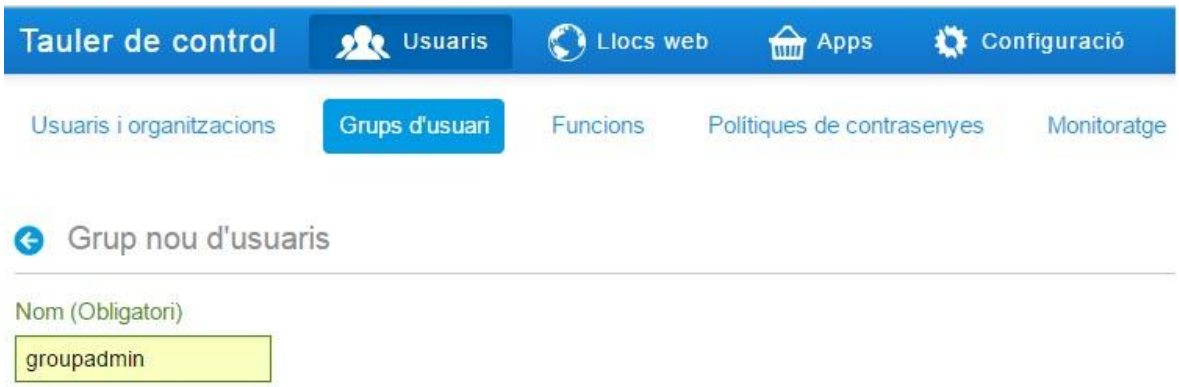


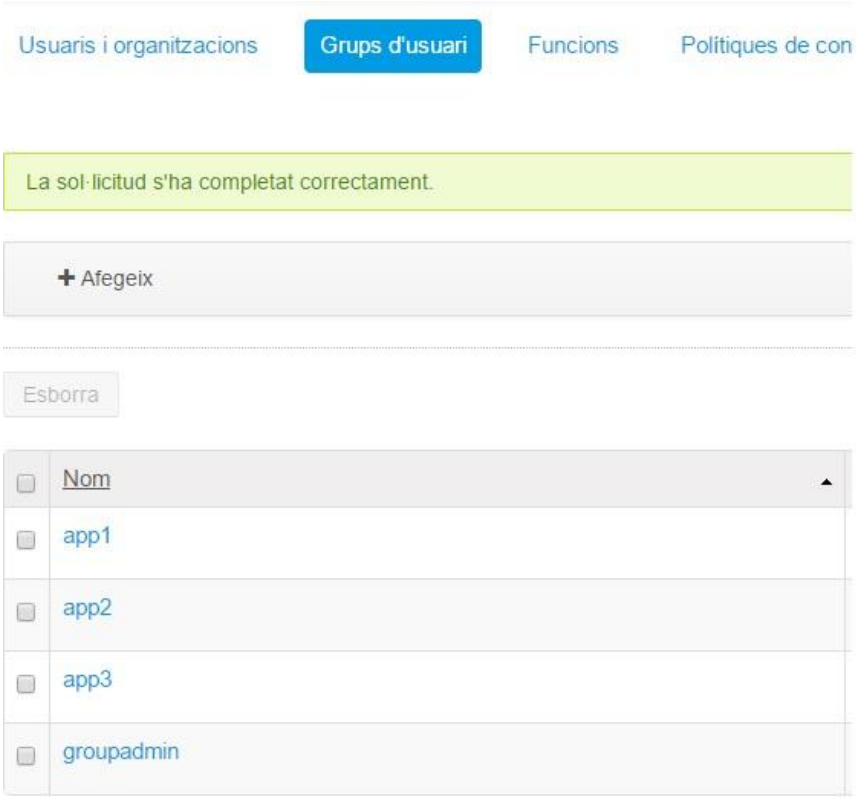
- Seleccionar la base de daes oracle amb l'usuari Liferay, per poder-lo vincular a l'esquema creat anteriorment.

- **Configuració del Portal Liferay**
  - **Creació d'usuaris:** Degut a que no podem importar els usuaris i grups del directori actiu, crearem aquests mitjançant el panel de control del Liferay. Per a l'administrador podem considerar l'usuari *test* definit per defecte a Liferay, tot seguit podem afegir per exemple l'usuari *user1* el qual no tindrà permisos d'administrador. D'aquesta manera ja tindrem els dos tipus d'usuaris per poder fer les diferents proves als portlets.

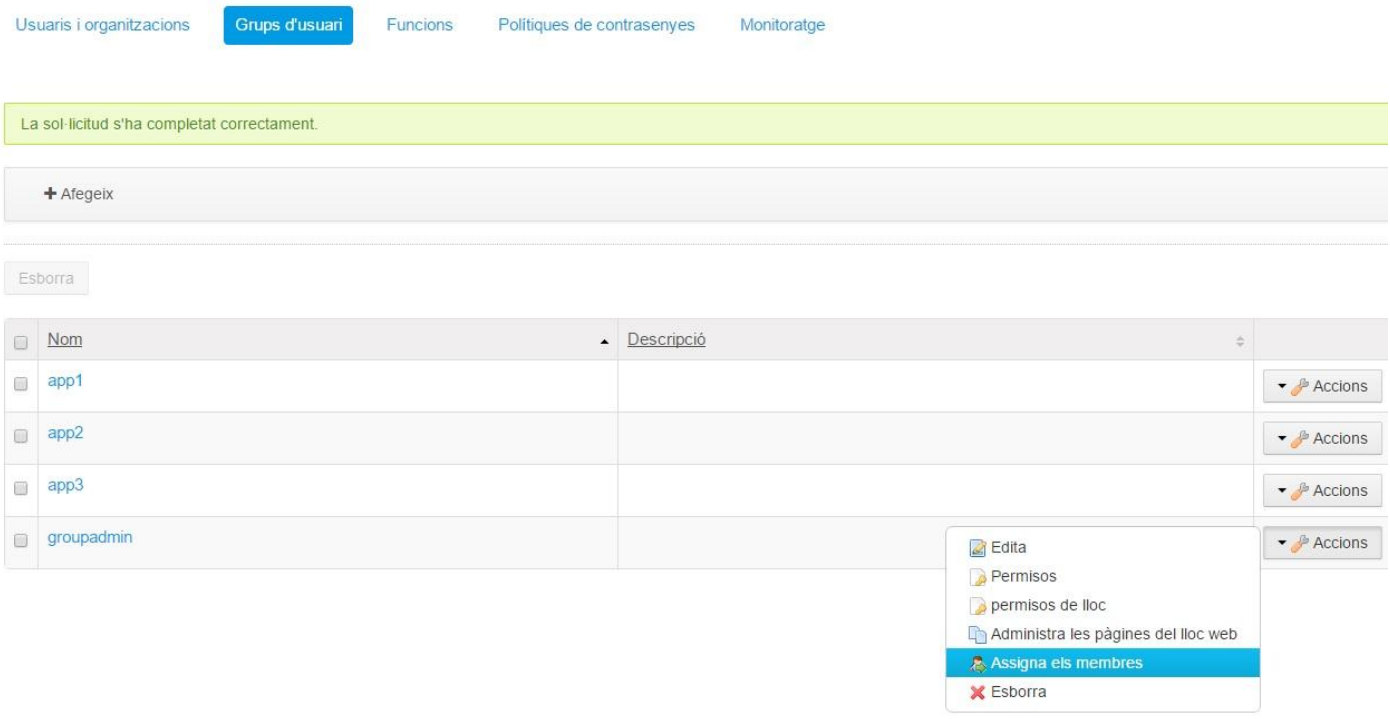


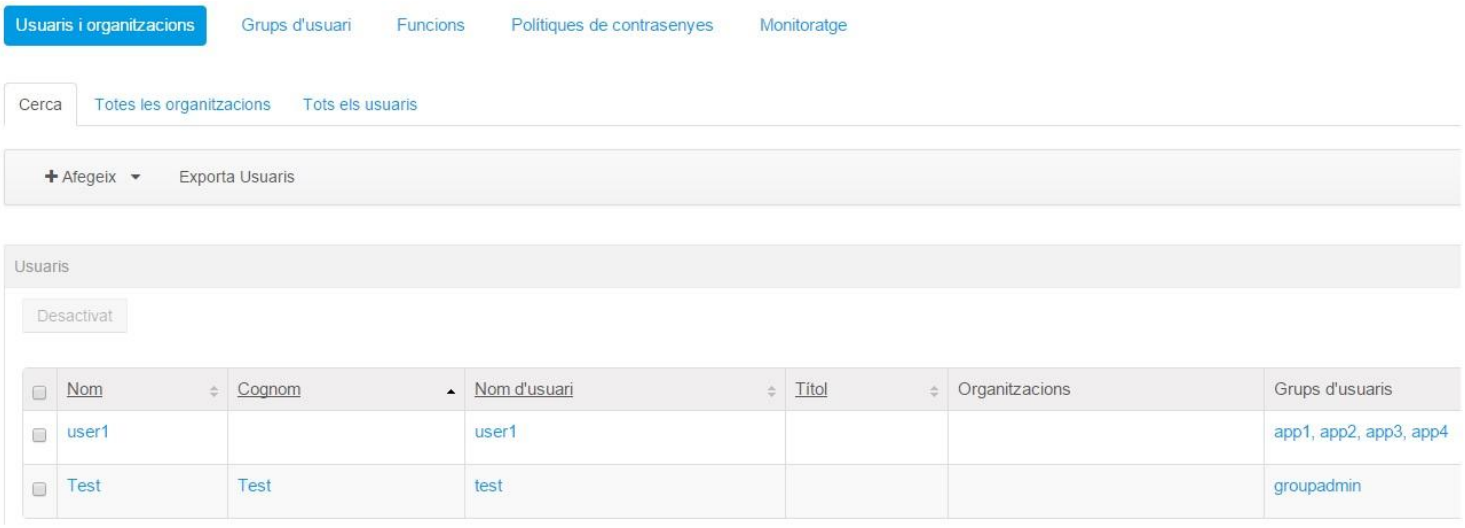
- **Creació de grups:** En aquest apartat es crearà el grup anomenat "groupadmin" el qual tindrà els permisos necessaris per poder administrar el portlet *aplicacions*. Per altra banda també necessitarem crear diferents grups d'aplicacions, on la seva funció serà determinar qui té accés a aquella aplicació.





- **Assignació d'usuaris als grups**: Primer de tot assignarem al grup "groupadmin" l'usuari que administrarà el portlet aplicacions, en el nostre cas podem aprofitar l'usuari *test*. Després assignarem a cadascú dels grups *app* l'usuari corresponent, en el nostre cas *user1*.





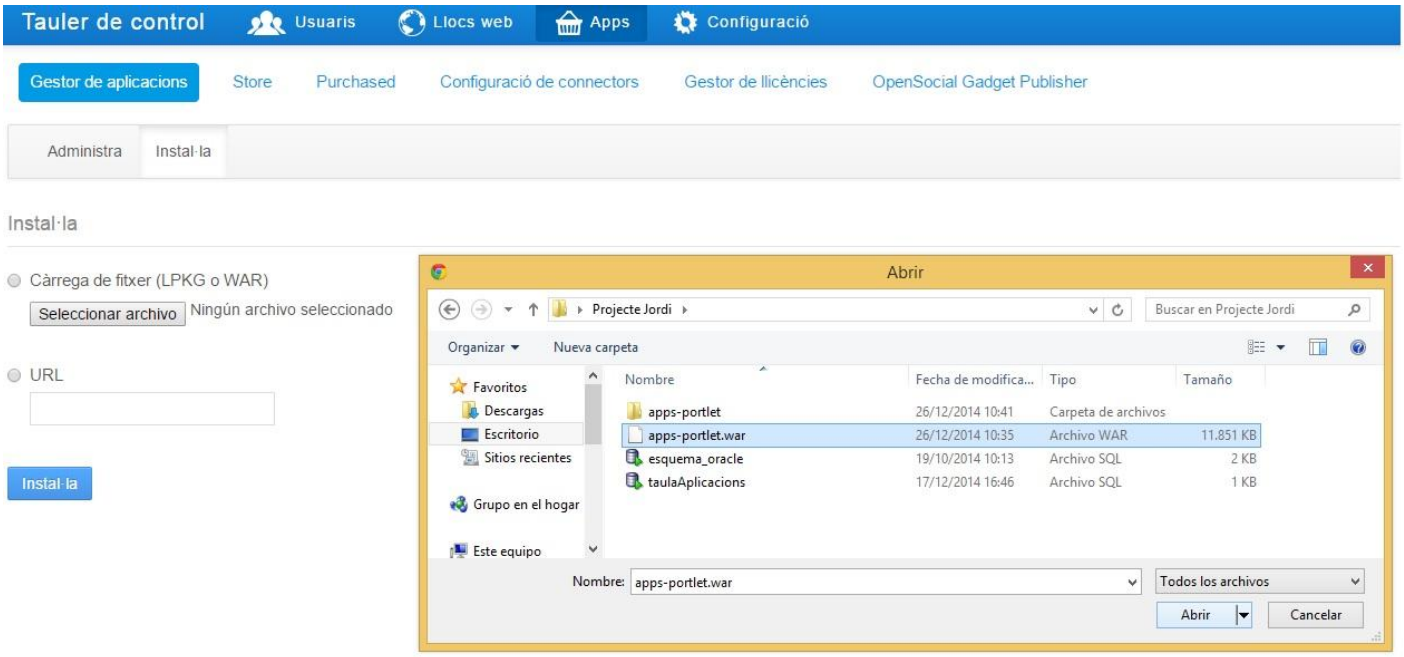
- ✓ **Deplegament portlet aplicacions**

  Un cop configurat l'entorn del portal Liferay carregarem el portlet aplicacions que s'ha desenvolupat. Tot es farà a través del panel de control del Liferay. Després afegirem una pàgina pública per tal de poder afegir aquest portlet.

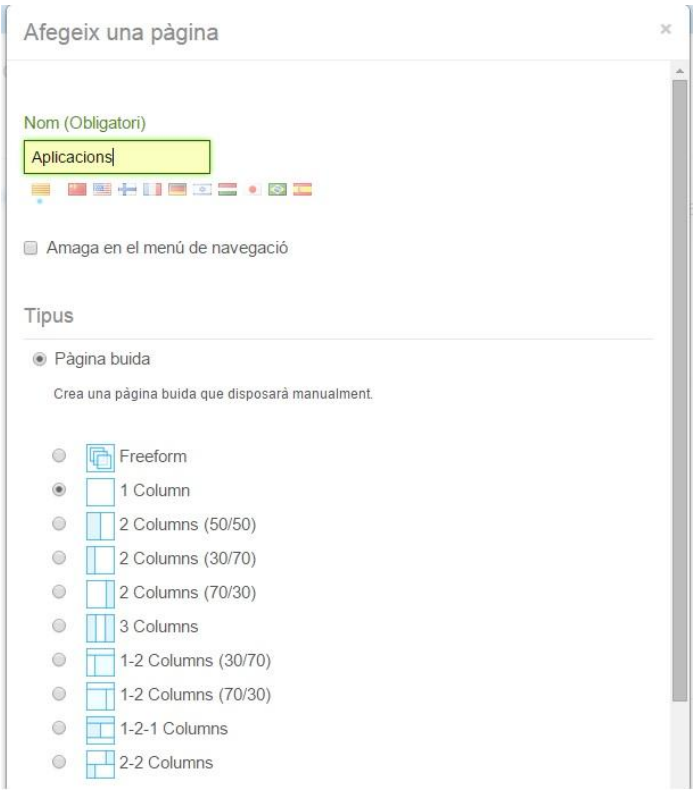
  - **Crear taula apps:** Executarem aquest script connectats a l'usuari Liferay. Aquest script *apps.sql* estarà ubicat dintre de la carpeta sql.

```
CREATE TABLE apps(
id_app int,
nom varchar2(50),
descripcio varchar2(50),
url varchar2(50),
usergroupid int,
observacions varchar2(150),
icon BLOB,
nomicon varchar2(50),
actiu char(1) check(actiu in ('N', 'Y')),
primary key(id_app));
```

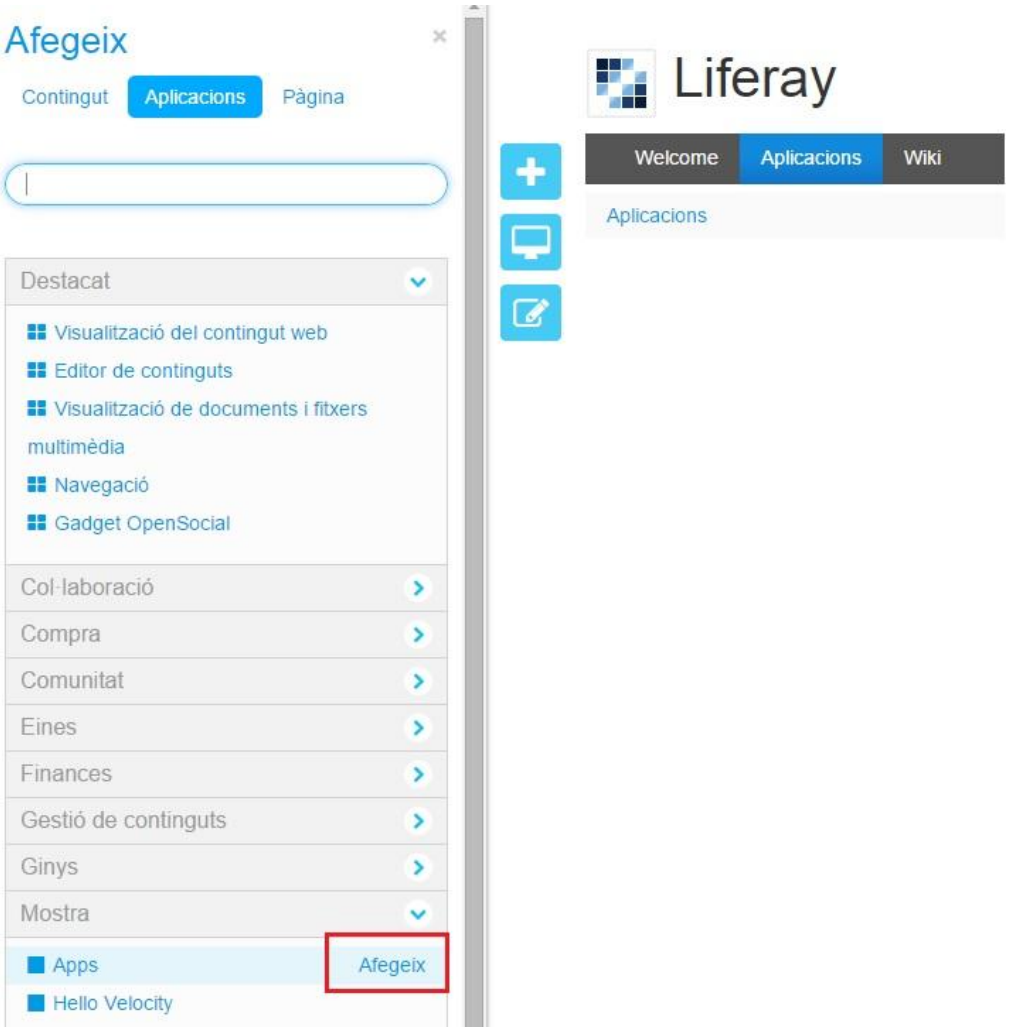
- **Carregar el portlet:** seleccionem l'arxiu *apps-portlet.war*.



- **Crear una nova pàgina**



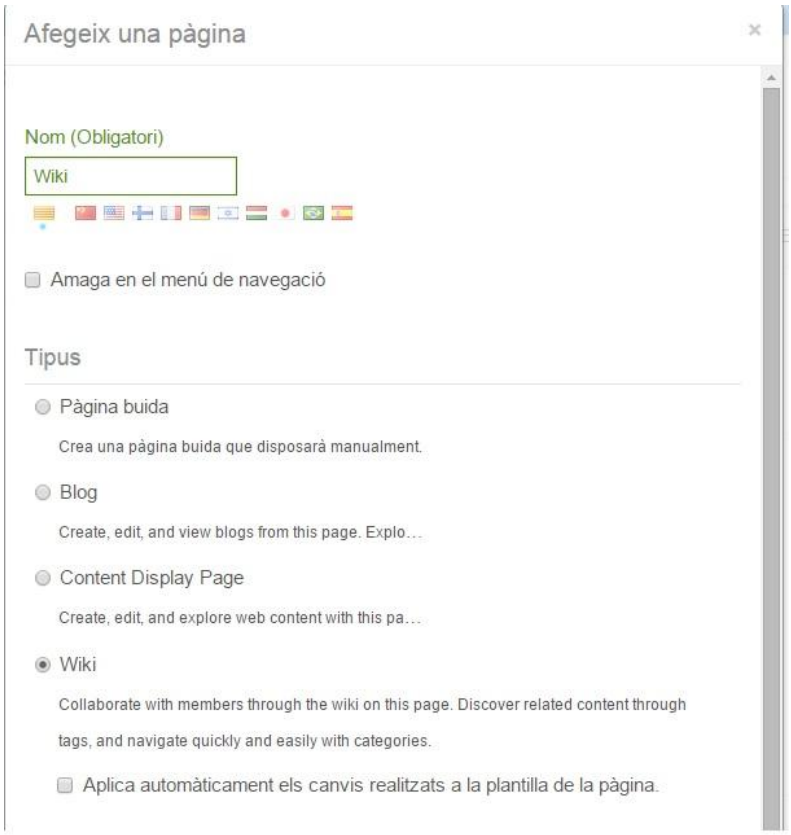
- **Afegir el portlet a la pàgina**



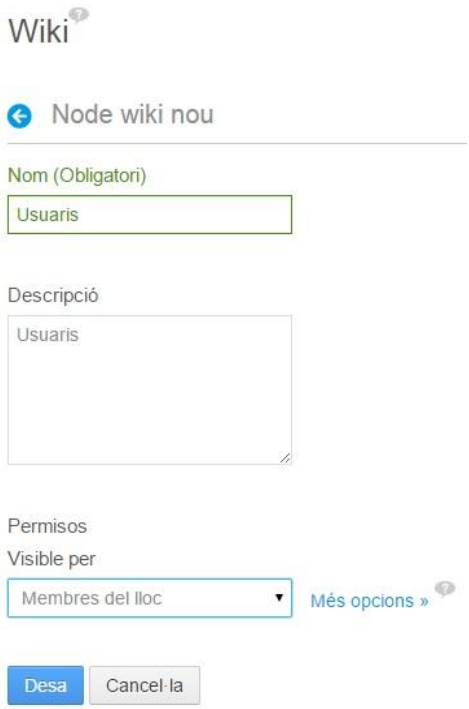
✓ **Desplegament portlet Wiki**

Aquest portlet ja el porta incorporat Liferay, per tant lúnic que s'haurà de fer es crear una pàgina ja definint que incorpora aquest tipus de portlet. I després a la configuració de la Wiki definir els dos tipus de Wikis que ens interessen, la d'Informàtica i la d'usuaris, amb els seus permisos corresponents.

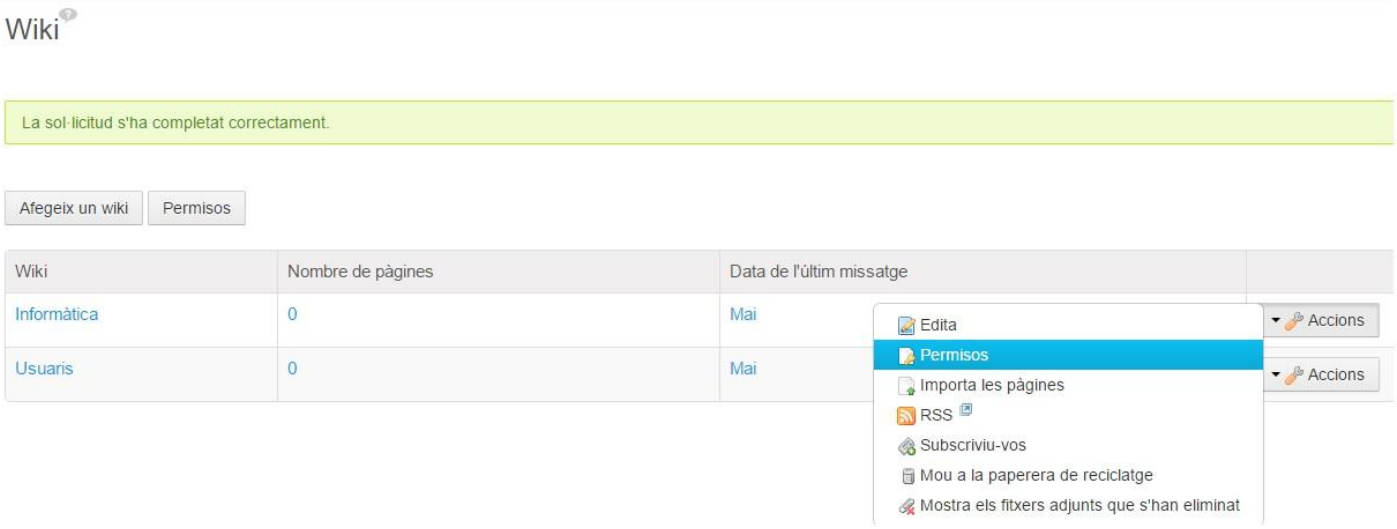
- **Crear una nova pàgina**



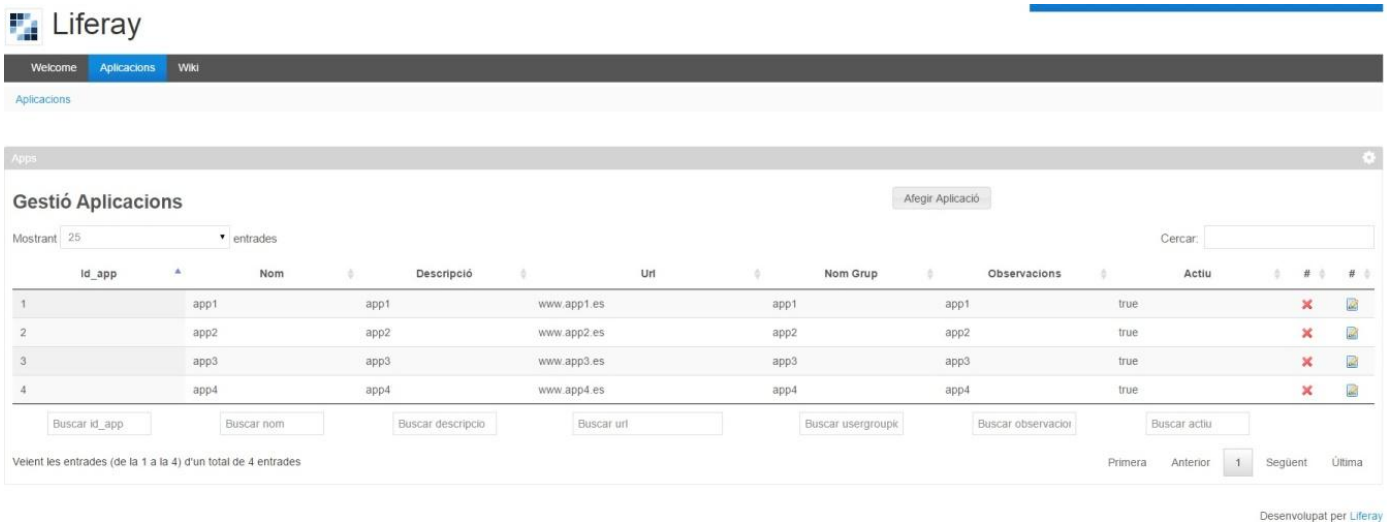
- **Configurar els dos tipus de Wiki, Usuaris i Informàtica.**

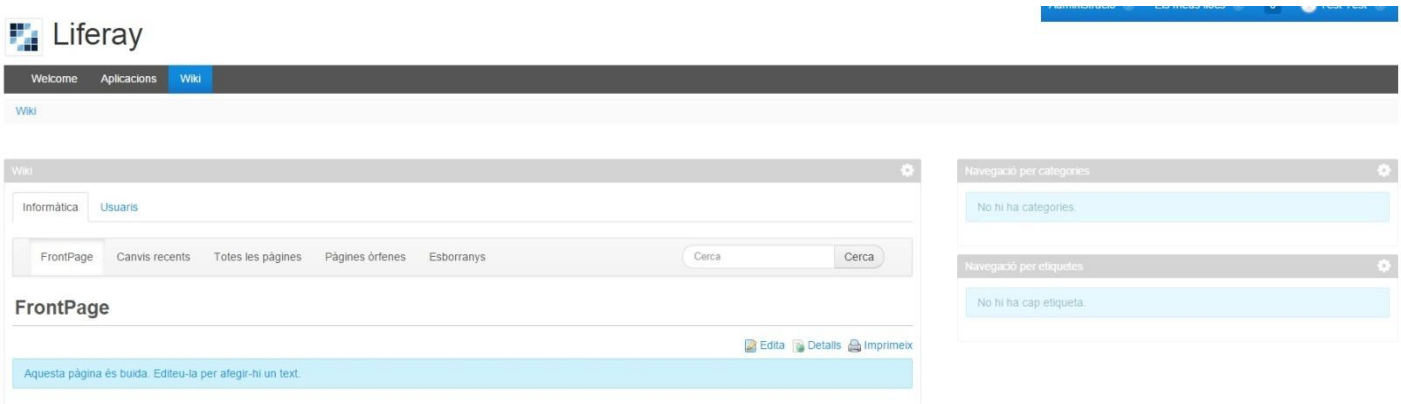


- **Configurar els permisos d'accés**



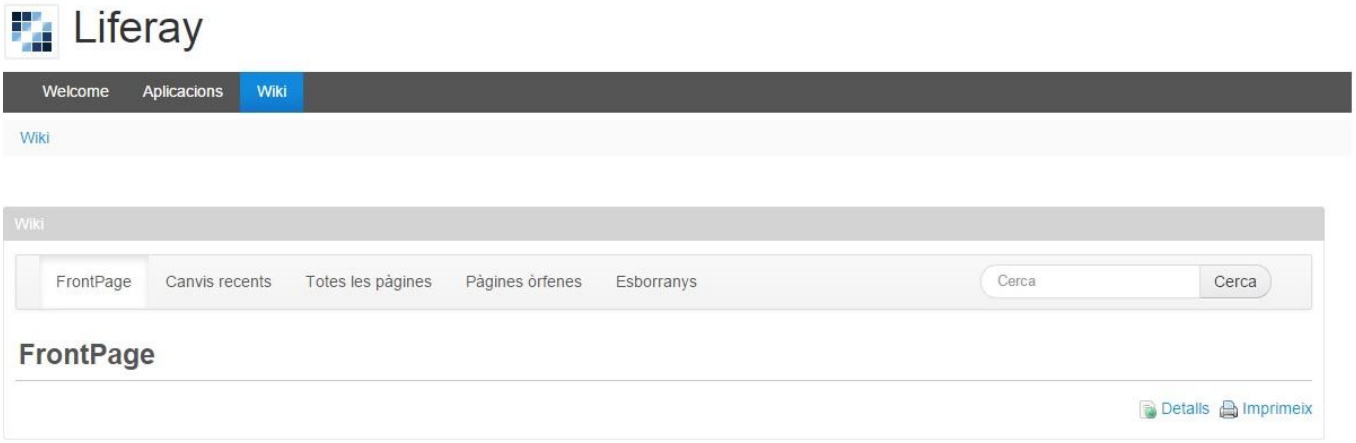
- ✓ **Portlets desplegats**
  - **Administrador**





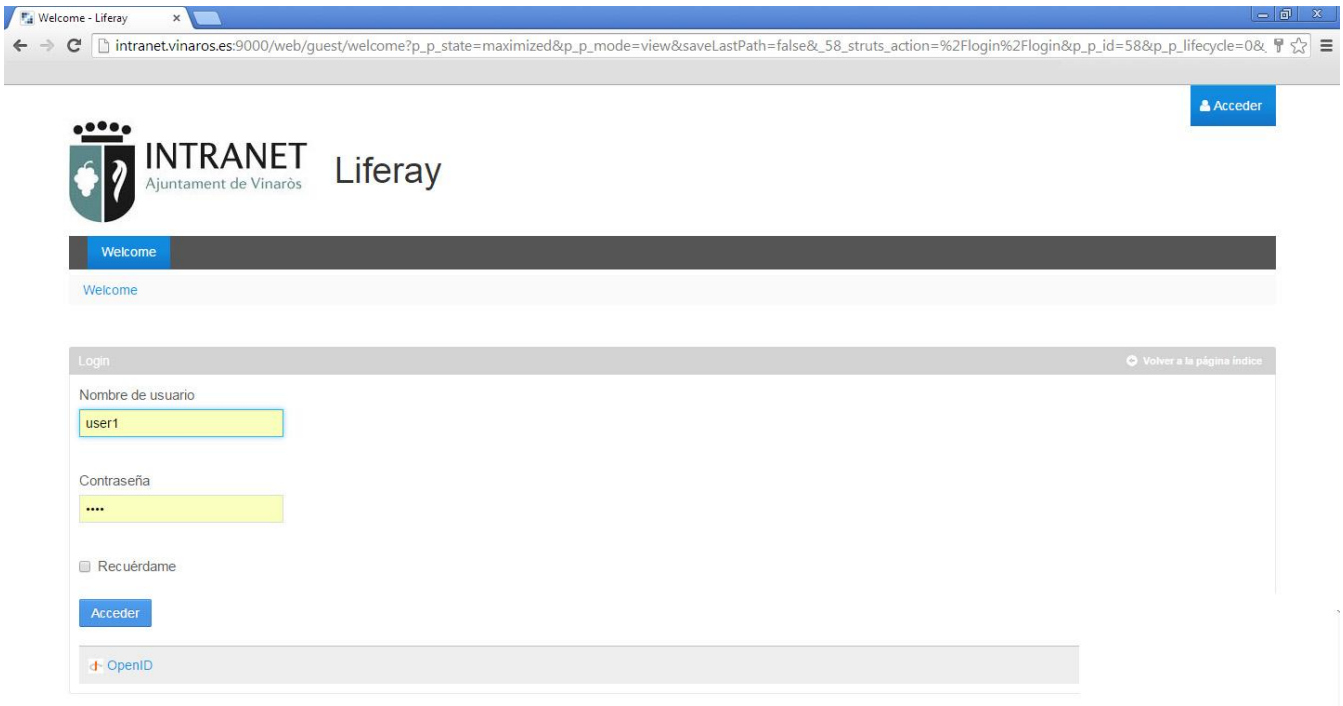
- **Usuari**


Liferay
0
user1
Welcome
Aplicacions
Wiki
Aplicacions
Apps
Aplicacions Usuari
app2
app1
app3
app4
Desenvolupat per Liferay


Liferay
Welcome
Aplicacions
Wiki
Wiki
Wiki
FrontPage
Canvis recents
Totes les pàgines
Pàgines òrfenes
Esborranys
Cerca
Cerca
FrontPage
Detalls
Imprimeix

B. Manual d'usuari

En aquest manual diferenciarem els dos rols definits per l'aplicació, l'administrador i l'usuari. Començarem per la pantalla d'identificació la qual determinarà quin tipus d'usuari és i a quines aplicacions pot accedir. L'usuari introduirà les seves credencials del domini i accedirà a l'aplicació.

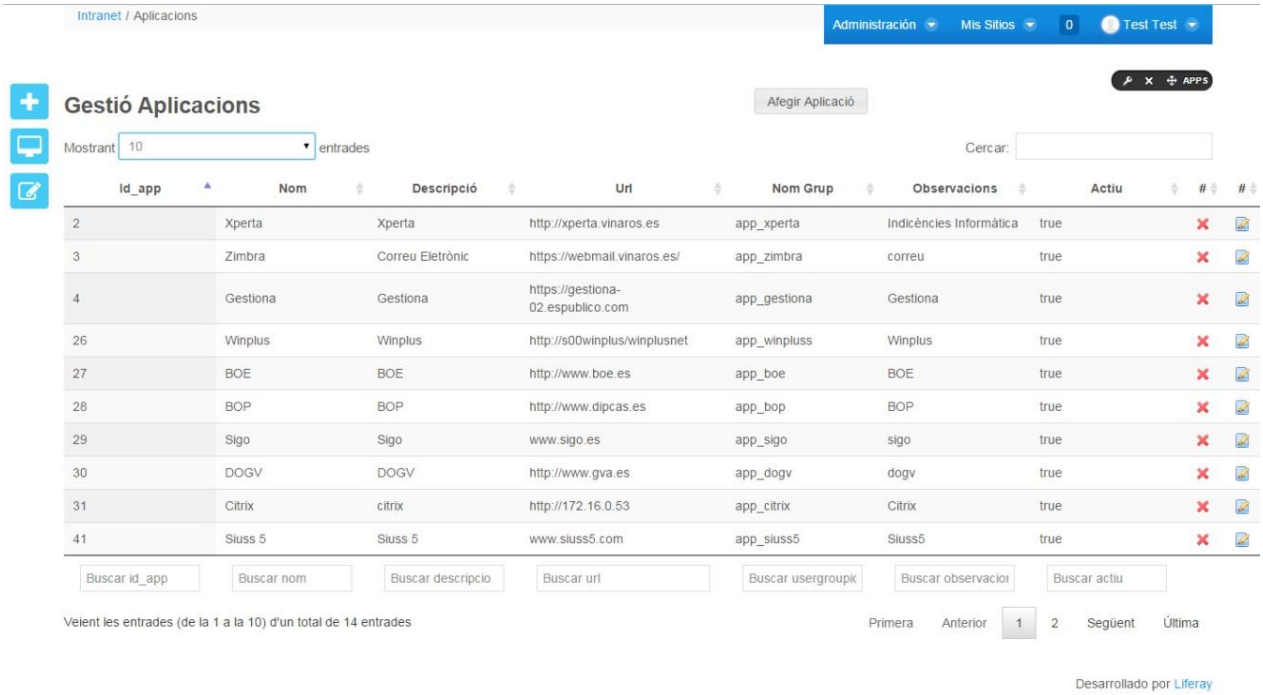


✓ Administrador

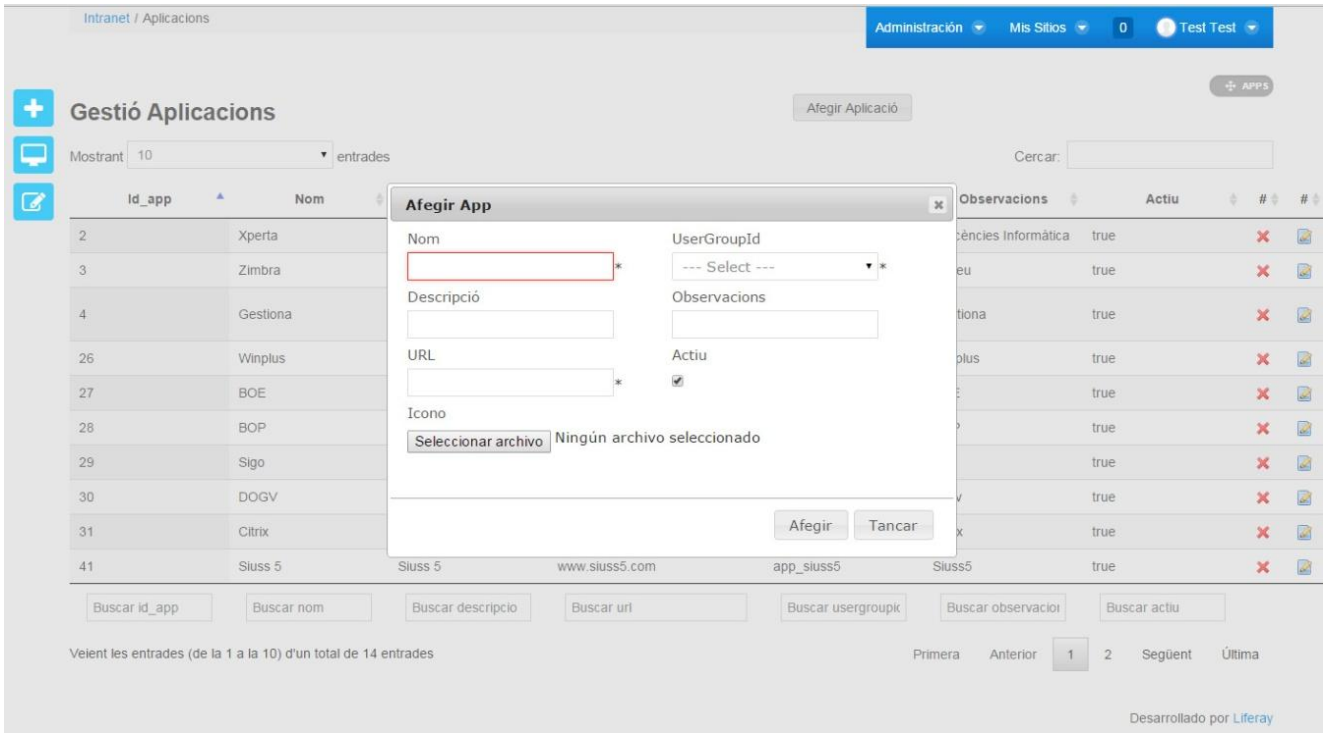
Serà l'administrador del portal Liferay i també l'administrador dels portlets Aplicacions i Wiki. Ens centrarem sols en l'administració dels diferents portlets i no amb l'administració del portal.

o Portlet Aplicacions

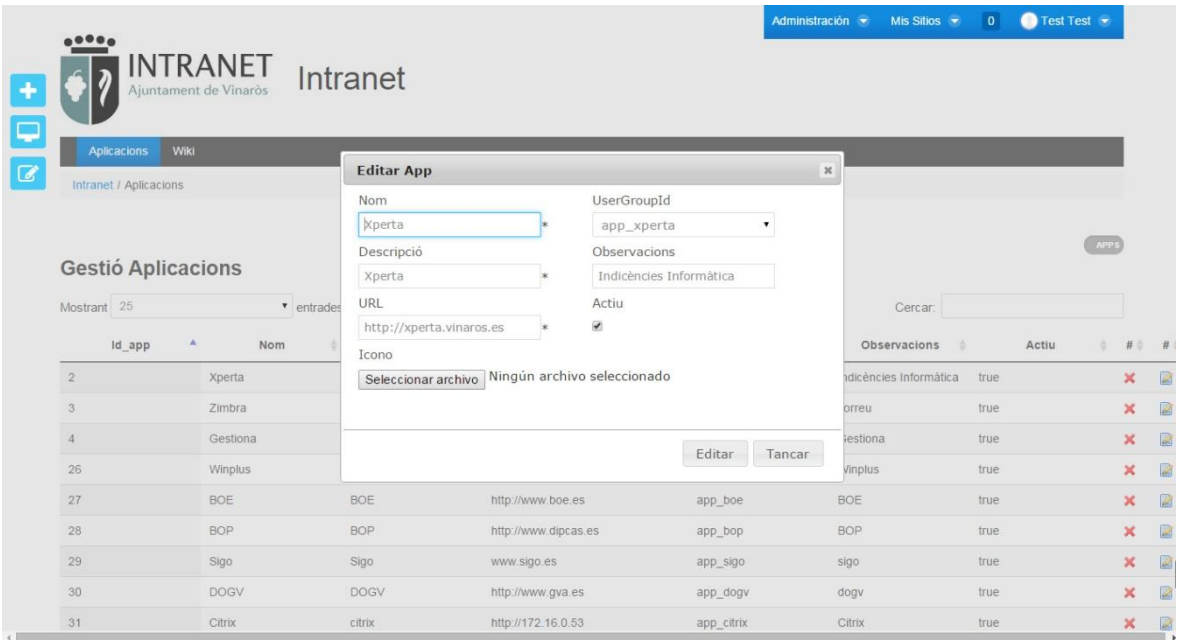
Es mostra el llistat de totes les aplicacions disponibles. On es poden fer cerques per camp i genèriques. Hi ha l'opció d'afegir una nova aplicació, de modificar-ne o d'eliminar-ne una d'existent . Totes aquestes funcionalitats de la taula ens la proporciona la *jqgrid*.



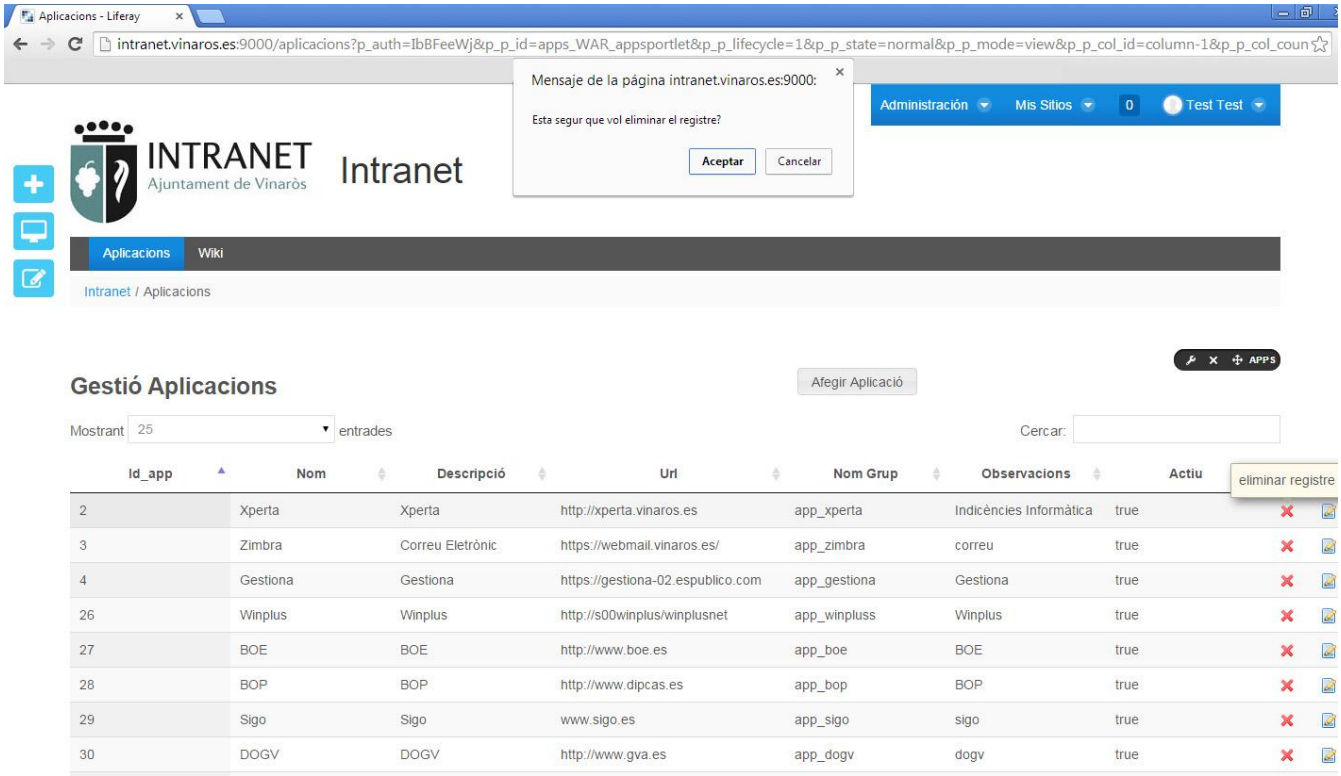
A l'hora d'afegir una nova aplicació s'ha escollit fer-ho en una finestra emergent per a que la pàgina principal no hagués de carregar una nova pàgina amb el formulari, d'aquesta manera millores la usabilitat de l'aplicació. També s'ha afegit validació a cada camp, per tant si no introdueixes les dades correctes als camps requerits, representats amb un "*"camp, apareix un missatge d'alerta i es posa el contorn en roig del camp que no s'ha introduït bé. També és carreguen el nom dels grups disponibles per tal de vincular-los ja en l'aplicació. Si no introdueixes cap imatge s'introdueix una definida per defecte. Un cop pitjat el botó d'afegir es refresca automàticament la taula i ja apareix la nova aplicació.



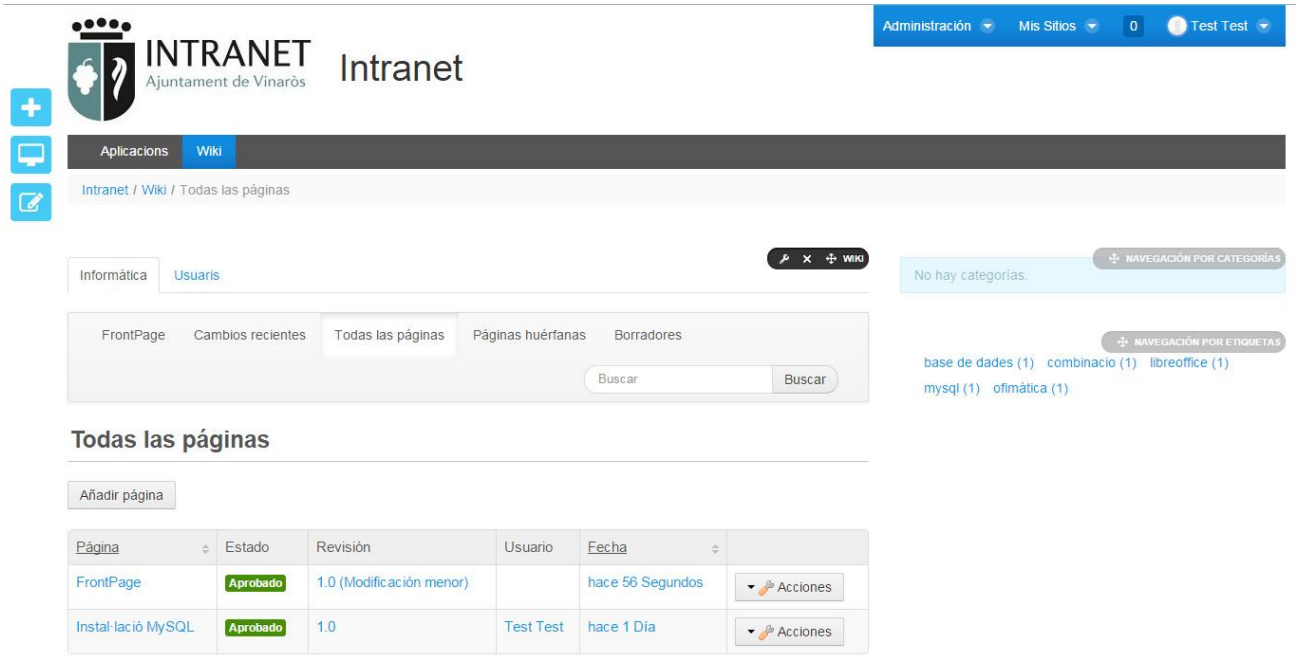
Quan es vol editar un registre només cal fer clic a la icona editar de la fila a editar i s'obrirà una finestra emergent amb tota la informació de l'aplicació. Només cal modificar els camps necessaris i tornar a guardar l'aplicació on també es refresca automàticament la llista d'aplicacions.



Per últim quan volem eliminar un registre tenim que fer clic a la icona amb la creu roja. Per seguretat ens apareixerà un missatge de confirmació, i en el cas que vulguem eliminar l'aplicació caldrà donar-li que si. Igualment que passa amb l'afegir i editar, la taula es refrescarà automàticament i aquest registre ja no sortirà.



- **Portlet Wiki**



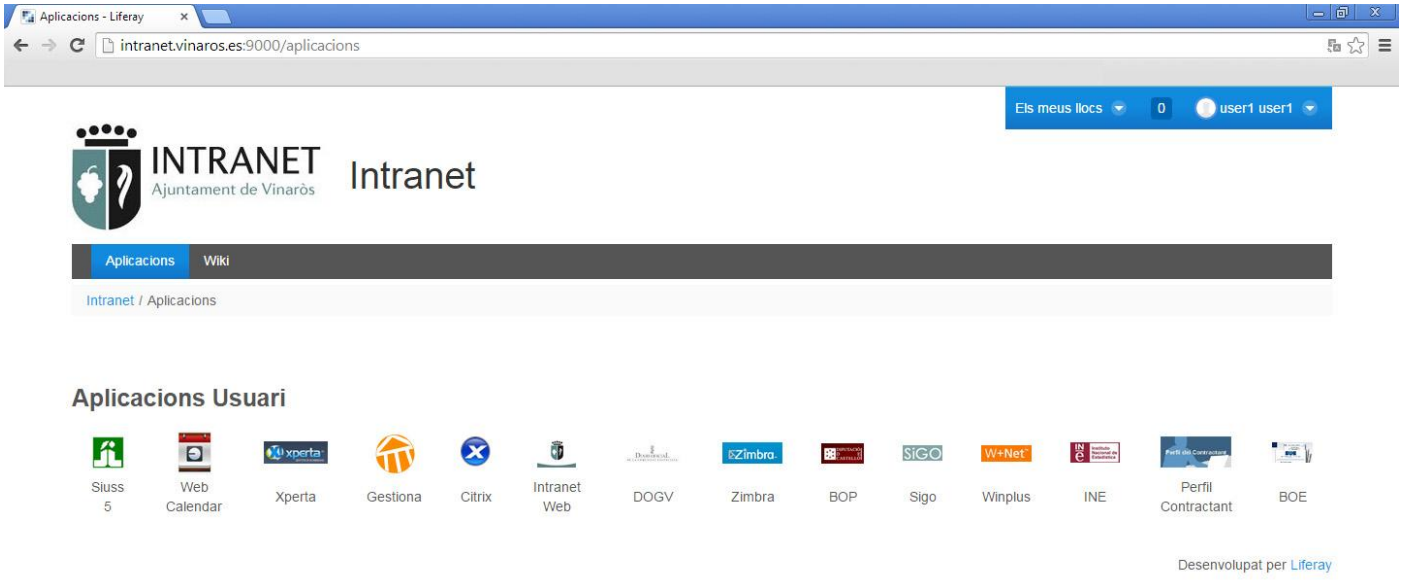
En el cas de l'administrador es pot comprovar que apareixen dos pestanyes on pots escollir a quina Wiki vols accedir, informàtica o usuaris. L'administrador podrà afegir els fils que vulgui a qualsevol Wiki, podent definir categories per fer més fàcil l'estructura de la Wiki. També és molt útil la utilització d'etiquetes per classificar els diferents temes, facilitant la cerca per paraules clau.

✓ **Usuaris**

Visualitzarà els portlets aplicacions i Wiki. Pel que fa al portlet aplicacions podrà veure les diferents aplicacions que té disponibles. En canvi, al portlet Wiki només tindrà accés a la Wiki d'usuaris i amb permís només de lectura.

- **Portlet Aplicacions**

En aquesta imatge podem veure les diferents aplicacions que té disponibles l'usuari *user1*. Fent clic a qualsevol aplicació sobri una nova pestanya amb la URL de l'aplicació. Hi ha webs que només són accessibles des de la LAN i altres que són públiques.



- **Portlet Wiki**

En el portlet Wiki es pot comprovar que l'usuari en aquest cas ja només veu la Wiki dels usuaris, i no la d'informàtica. Allí trobarà tots els fils que li puguen interessar, podrà fer cerques i també buscar per paraules claus.