UOCBOOK

Xarxa social fent ús de J2EE

TFC – J2EE	JavaServer Faces, ICEFaces, Hibernate, MySQL
Autor	Ivan Folgueira Bande
Estudis	ETIS
Professor	Salvador Campo Mazarico
Data	16 Gener 2012

Índex

1. Introducció

El present document detalla el procés de disseny i elaboració d'una xarxa social realitzada al voltant de la tecnologia J2EE. L'organització dels capítols d'aquesta memòria és la següent:

1. El present capítol, que serveix d'**introducció** al sistema.
2. **Descripció del producte** Descriu el producte d'es d'un punt de vista funcional. És a dir, només explica el que fa el producte i no entra en detalls tècnics.
3. **Estat de l'art i tecnologia emprada** Es realitza un repàs de la tecnologia usada des d'un punt de vista teòric.
4. **Implementació** Descriu l'aplicació des d'un nivell una mica més baix i tracta de donar un punt de vista més tècnic del producte.
5. **Valoració econòmica**
6. **Conclusions**
7. **Glossari de termes**
8. **Bibliografia**
9. **Annex**

1.2. Justificació del TFC i context en el que s'aplica

Donada la present embranzida existent entorn les xarxes socials, tals com twitter o facebook, resultava molt interessant el tractar de realitzar una xarxa social amb la tecnologia J2EE. Òbviament, des d'un punt de vista didàctic i sense ànims de fer ombra a aquestes grans empreses.

A més, partint d'un coneixement inexistent pel que fa a les xarxes socials i sense coneixements previs en J2EE. La realització d'aquest projecte suposava un doble repte personal: per una banda s'havia d'aconseguir interrelacionar diversos usuaris entre sí i fer que puguin intercanviar informació d'una manera més o menys ordenada; i per altra banda, es tenia la restricció implícita consistent en que la tecnologia havia de ser J2EE.

Per tant, aquest projecte resultava molt interessant per aquesta doble vessant: xarxa social i tecnologia J2EE.

1.3. Objectius del TFC

El principal objectiu d'aquest projecte ha estat el construir un canal de comunicació entre diversos usuaris i que aquest conjunt d'usuaris pugui formar una xarxa social. A més, es pretén que el cost sigui relativament econòmic.

1.4. Enfocament i mètode seguit

En la primera etapa del projecte, s'havia de respondre a dues preguntes bàsiques: *Què fer?* i *Cóm fer-ho?* La resposta a la primera pregunta ha estat molt senzilla i la seva justificació ja s'ha donat anteriorment. En canvi, per respondre a la segona pregunta, s'ha hagut de passar per un camí relativament tortuós en el qual es buscava trobar una solució econòmica, ben documentada pel que fa a la tecnologia i que permetés realitzar una aplicació el més atractiva i senzilla de fer servir possible per l'usuari final.

La segona etapa, i la més important, ha estat la de la preparació de l'entorn de treball, el disseny i la construcció de l'aplicació. Després de preparar tot l'entorn (IDE, sistema de BBDD, etc), i, un cop ja s'havien decidit les eines i frameworks que es farien servir, faltava posar una mica d'ordre i dividir l'objectiu final en diverses fites de menor tamany. D'aquesta manera, s'ha pogut realitzar una millor estimació del temps que es trigaria en fer cadascuna d'aquestes parts i s'ha pogut veure fins a on es podia arribar. I, un cop que ja s'havien decidit aquests casos d'ús, el següent ha estat el tractar d'implementar tots els casos que s'havia previst com a realitzables i deixar l'aplicació el millor preparada possible per tal de poder afegir els casos d'ús que han quedat pendents per futures versions del sistema.

Per últim, l'etapa final del projecte consisteix en tractar de plasmar tota la feina realitzada en la memòria final i la correspondent presentació.

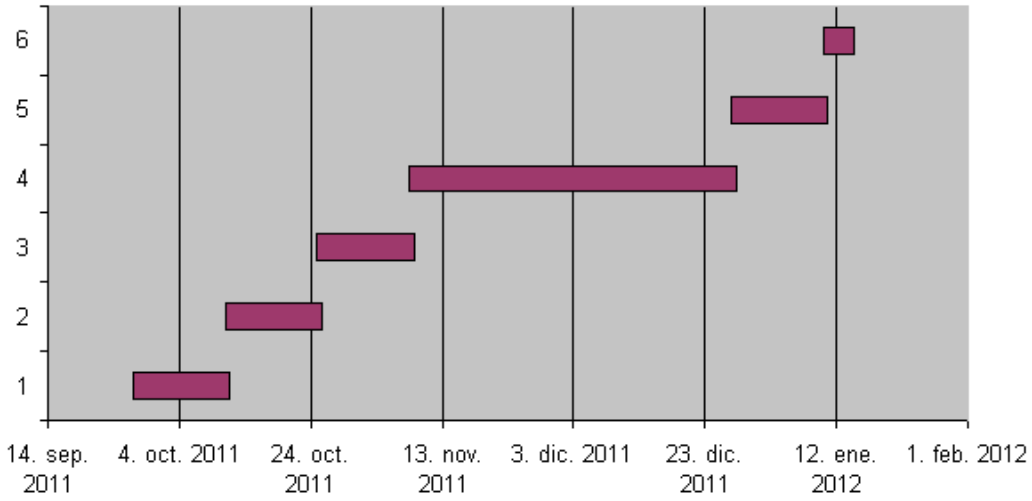
1.5. Planificació del projecte

A continuació es mostra una taula que mostra una divisió temporal amb l'estimació inicial pel que fa al temps de dedicació en l'elaboració de les principals parts del projecte.

Tasca	Temps per realitzar-la
1. Estudi dels principals conceptes de J2EE i del que es pot fer amb aquesta tecnologia.	15 dies
2. Definició de les principals funcionalitats del sistema així com la realització d'un estudi sobre la viabilitat tècnica i operativa per tal de dir si és factible realitzar-les mitjançant J2EE.	15 dies
3. Aquesta part és la corresponent a l'enginyeria del software i ha de començar a traduir totes les idees i conceptes definits a l'anterior part en l'arquitectura i estructura que finalment hauria de tenir el sistema.	15 dies
4. Implementació i validació del sistema seguint tots els esquemes i patrons definits en l'anterior etapa del projecte.	50 dies
5. Redacció de la memoria	15 dies
6. Realització de la presentació	5 dies

I el corresponent diagrama de Grantt és el següent:

Activitats	Data d'inici	Duració (dies)	Data final
1	27. sep. 2011	15	11. oct. 2011
2	11. oct. 2011	15	25. oct. 2011
3	25. oct. 2011	15	8. nov. 2011
4	8. nov. 2011	50	27. dic. 2011
5	27. dic. 2011	15	10. ene. 2012
6	10. ene. 2012	5	14. ene. 2012



1.6. Producte obtingut

El producte obtingut és una xarxa social virtual que permet als usuaris establir vincles amb altres usuaris i intercanviar informació. Es tracta d'una aplicació fàcil de fer servir i intuïtiva per a l'usuari final i que li permet realitzar les següents accions:

- Determinar la manera com li veuen els altres usuaris, és a dir, pot editar el seu perfil i imatge personal.
- Escriure missatges que puguin ser vistos per la resta de la comunitat.
- Buscar amics a dintre de la xarxa.
- Realitzar i rebre peticions d'amistat.
- Escriure i rebre missatges privats amb els amics de la xarxa.

2. Descripció del producte

En aquest capítol es dóna una descripció a nivell funcional del sistema sense entrar en detalls tècnics.

2.1. Requeriments funcionals

Els principals requeriments funcionals que suporta el sistema són els següents:

- **Interconnexió** El sistema ofereix la possibilitat d'interconnectar diversos usuaris entre sí tot formant una xarxa de vincles.
- **Trobar persones** L'aplicació ofereix la funcionalitat de trobar amics que s'hagin enregistrat al sistema però que encara no s'hagi establert cap vincle dintre de la xarxa.
- **Intercanvi d'informació** La xarxa permet als usuaris intercanviar missatges de manera pública (a tota la xarxa) i també de manera individual (un usuari pot enviar un missatge a un únic destinatari).
- **Conèixer gent nova** La xarxa permet fer nous amics. Els usuaris, a l'intervanviar comentaris a través de l'espai comú, poden conèixer-se millor entre sí i així arribar a trobar persones amb similar manera de pensar.

2.2. Casos d'ús

Com es pot extreure de l'apartat anterior, una xarxa social ofereix la capacitat d'interconnectar diversos usuaris dintre d'un mateix espai virtual. Per tant, els principals casos d'ús estaran estretament lligats a operacions del tipus *enregistrar usuari*, *escriure missatge, editar perfil,* etc. Amb això, els casos d'ús en que es divideix el projecte són els que es descriuen a continuació.

I. Enregistrament d'un usuari a la xarxa

Actor: usuari que vol enregistrar-se a la xarxa	
Precondició: L'usuari no està enregistrat	
Cas normal	**Cas alternatiu**
1) L'usuari introdueix les seves dades en els camps de text.	
2a) El sistema dóna per vàlides les dades de	2b) El sistema considera que alguna de les

l'usuari.	dades de l'usuari no són vàlides (per exemple, la direcció de correu electrònic és errònia o la contrasenya és massa curta).
3a) El sistema dirigeix a l'usuari a la pàgina de benvinguda o “home”. Així mateix, el sistema emmagatzema les dades de l'usuari a la BBDD i aquest queda enregistrat per futurs accessos a la xarxa.	3b) El sistema informa a l'usuari de quin camp ha introduit malament per tal de que el tracti de corregir. La notificació apareix a la pròpia pàgina de “login/registre”.
Post condició: l'usuari està enregistrat	**Post condició:** l'usuari **no** està enregistrat

II. Desenregistrament d'un usuari de la xarxa

Actor: usuari que vol desenregistrar-se de la xarxa	
Precondició: L'usuari està enregistrat, ha accedit a la xarxa i es troba a la pàgina “home”.	
Cas normal	**Cas alternatiu**
1) Des de la pàgina de “home”, l'usuari selecciona la opció “desenregistrament”.	
2) El sistema mostra a l'usuari un quadre de diàleg preguntan-li si realment vol realitzar l'acció de desenregistrament. Les dues possibles respostes són 'sí' o 'no'.	
3a) L'usuari confirma l'acció tot fent click al botó del “sí”.	3b) L'usuari respon fent click al botó del “no”.
4a) A l'usuari se li mostra la pàgina de “login/registre”.	4b) Desapareix el quadre de diàleg i l'usuari segueix veient la pàgina de “home”.
Post condició: l'usuari queda desenregistrat i s'esborren totes les seves dades. Aquest usuari ja no existeix a la xarxa.	**Post condició:** l'usuari roman enregistrat a la xarxa i la pot fer servir de manera normal.

III. Inici de sessió d'un usuari

Actor: usuari que vol iniciar sessió	
Precondició: L'usuari ja s'havia enregistrat previament en una sessió anterior, vol accedir a la xarxa i es troba a la pàgina de “login/registre”.	
Cas normal	**Cas alternatiu**
1) Des de la pàgina de “login/registre”, l'usuari introdueix dues dades que havia introduit en el moment d'enregistrar-se: la seva adreça de correu electrònic i la contrasenya.	

El sistema compara les dades introduïdes per l'usuari amb totes les dades de tots els usuaris emmagatzemades a la BBDD.	
2a) L'usuari introdueix les seves dades (correu i contrasenya) correctament.	2b) L'usuari s'equivoca a l'hora d'introduir alguna de les seves dades (correu o password).
3a) L'usuari pot iniciar sessió i és redirigit a la pàgina de “home”.	3b) L'usuari segueix observant la pàgina de “login/registre” i el missatge corresponent notificant-li que alguna de les seves dades introduïdes no és correcta.
Post condició: l'usuari ha iniciat sessió i es troba a la pàgina de “home”.	**Post condició:** l'usuari no ha iniciat sessió i segueix visualitzant la pàgina de “login/registre”.

IV. Final de sessió d'un usuari

Actor: usuari que vol finalitzar la sessió	
Precondició: L'usuari ja s'havia enregistrat previament en una sessió anterior, vol accedir a la xarxa i es troba a la pàgina de “login/registre”.	
Cas normal	**Cas alternatiu**
1) Des de la pàgina de “login/registre”, l'usuari introdueix dues dades que havia introduit en el moment d'enregistrar-se: la seva adreça de correu electrònic i la contrasenya.	
2) El sistema compara les dades introduïdes per l'usuari amb totes les dades de tots els usuaris emmagatzemades a la BBDD.	
3a) L'usuari introdueix les seves dades (correu i contrasenya) correctament.	3b) L'usuari s'equivoca a l'hora d'introduir alguna de les seves dades (correu o password).
4a) L'usuari pot iniciar sessió i és redirigit a la pàgina de “home”.	4b) L'usuari segueix observant la pàgina de “login/registre” i el missatge corresponent notificant-li que alguna de les seves dades introduïdes no és correcta.
Post condició: l'usuari ha iniciat sessió i es troba a la pàgina de “home”.	**Post condició:** l'usuari no ha iniciat sessió i segueix visualitzant la pàgina de “login/registre”.

V. Accés d'un usuari al seu propi espai personal

Actor: usuari que vol accedir al seu propi espai personal
Precondició: L'usuari es troba a la pàgina de “home”.
Cas normal
1) Dintre de la pàgina de “home”, l'usuari selecciona l'opció “editar perfil” tot fent click en l'enllaç corresponent.

2) El sistema realitza una consulta a la BBDD per obtenir les dades relacionades amb l'usuari[1]. [1]De fet, en la primera versió de la xarxa l'usuari només té el seu nom i una imatge asociada. No obstant, es podrien afegir més dades de manera senzilla, en cas de que fos necessari.
3) El sistema redirigeix a l'usuari cap a la pàgina d'edició del seu perfil. En aquesta pàgina, inicialment l'usuari veu la imatge i el nom que ell mateix havia editat. Si no havia editat mai el seu perfil, l'usuari visualitzarà el nom que havia introduït en el moment del registre i observarà una imatge per defecte. En aquest moment, l'usuari pot editar el seu perfil o, si ho prefereix, pot deixar-ho com estava i tornar a la pàgina de “home”.
Post condició: l'usuari es troba a la pàgina d'edició del seu perfil.

VI. Edició del seu espai personal

Actor: usuari que vol editar el seu perfil		
Precondició: Continua pel punt 3 del cas d'ús anterior.		
Canvi de nom	**Canvi de la imatge personal**	**Cancel·lació**
1a) L'usuari edita el seu nou nom o “nick” (que es mostrarà a la resta d'usuaris) a través del quadre de text pertinent.	1b) L'usuari fa click sobre el botó d'edició d'imatge.	1c) L'usuari fa click sobre el botó de cancel·lació de la edició del perfil.
	2b) El sistema mostra a l'usuari un quadre a partir del qual pot fer click al botó “examinar”.	2c) El sistema torna a l'usuari a la pàgina “home” sense haver realitzat cap canvi.
	3b) L'usuari busca una imatge a través del sistema de fitxers del seu PC. La selecciona i fa click sobre el botó “pujar”.	
	4b) En aquest punt, l'usuari pot fer-se enrera i fer click al botó “cancel·lar” o pot corroborar la seva acció i fer click en el botó “validar”. En aquest últim cas, la imatge és pujada al sistema i s'emmagatzema una referència de la mateixa a la BBDD.	
Postcondició: l'usuari està a la pàgina “home” amb les dades modificades segons hagi volgut.		L'usuari torna a “home” sense cap canvi.

VII. Visualització dels amics d'un amic

A una xarxa social és interessant el poder visualitzar els contactes d'algun contacte teu ja que és probable que coneguis a aquest altre contacte però encara no hagueu establert una relació dintre de la propia xarxa. De fet, es pot considerar com una manera de buscar amics dintre de la xarxa.

Actor: usuari que vol visualitzar els contactes d'algun dels seus contactes. L'anomenarem *Usuari.*
Precondició: L'*Usuari* ha ingressat a la xarxa i es troba a la pàgina de "home". A més, l*'Usuari* ha de tenir almenys un contacte dintre de la seva llista de contactes.
1) L'*Usuari* fa click sobre el botó "Amics" que té un dels seus contactes. A aquest contacte, l'anomenarem *Amic.*
2) El sistema realitza una consulta SQL per tots els amics de l'*Amic* i els mostra en una llista de contactes. En aquesta llista, el sistema mostrarà tots els amics de *Amic* que encara no són amics de *Usuari.* A més, com és obvi, en aquesta llista tampoc apareixerà *Usuari.*
Postcondició: *Usuari* pot visualitzar els amics de *Amic.*

VIII.Petició d'amistat d'un usuari cap a un altre.

Aquest cas d'ús es divideix en dos casos d'ús clarament diferenciats. És per això que cadascun d'aquests casos els indico en taules diferents.

a) Petició d'amistat des de la pàgina de buscar amics

Actor: usuari que vol realitzar la petició d'amistat.
Precondició: L'*Usuari* ha ingressat a la xarxa i es troba a la pàgina de "home". A més, a la xarxa hi han d'haver més usuaris i no hi ha cap vincle entre l'usuari que realitza la petició d'amistat i l'usuari que la rep.
1) L'usuari fa click sobre el botó "buscar amics" de la pàgina "home".
2) El sistema redirigeix a l'usuari cap a la pàgina de "buscar amics". Es mostra una pàgina molt senzilla en la qual es pot veure un quadre de text, per introduir el nom que es vol buscar, i dos botons: un per realitzar la petició d'amistat i un altre per cancel·lar la operació i tornar a la pàgina de "home" sense haver realitzat la petició.
3) L'usuari comença a escriure el nom de la persona que vol buscar.
4) Per cada caracter que introdueix l'usuari, el sistema realitza una cerca entre tots els noms de tots els usuaris enregistrats a la xarxa. És a dir, si per exemple, l'usuari està buscant a una persona que es digui "Joan Manel", la seqüència de resultats que podria donar el sistema per cada nou caracter és la que es mostra a continuació:

Text escrit dintre del quadre de text	Noms que es mostren

J	Josep Maria Jaume Mendez Georgina Jimenez Juan Pedro Gregori Josep Lluis Joan Manel
Jo	Josep Maria Josep Lluis Joan Manel
Joa	Joan Manel

5) L'usuari selecciona a un usuari de la llista i fa click al botón “Acceptar”.

6) El sistema genera la petició d'amistat en forma d'entrada d'una taula concreta de la BBDD. Quan l'usuari que ha rebut la petició es connecti, veurà que té una sol·licitud d'amistat pendent.

Postcondició: a la BBDD consta que un usuari ha realitzat una sol·licitud d'amistat cap a un altre usuari.

b) Petició d'amistat visualitzant els amics d'algun contacte

Actor: usuari que vol realitzar una petició d'amistat.

Precondició: l'usuari parteix de la postcondició del cas d'ús **vii.**

1) De la llista d'amics que l'usuari observa, pot realitzar una sol·licitud d'amistat cap a un d'ells només fent click al botó de “Petició d'amistat” associat a l'amic seleccionat.

2) El sistema introdueix la sol·licitud d'amistat dintre de la BBDD de manera anàloga a com ho fa pel punt 6 del cas d'ús anterior.

Postcondició: a la BBDD consta que un usuari ha realitzat una sol·licitud d'amistat cap a un altre usuari.

IX. Acceptació/rebuig d'una proposició d'amistat

Actor: usuari que ha rebut la sol·licitud d'amistat

Precondició: l'usuari receptor i emissor de la sol·licitud no tenen cap vincle previ. A més, el receptor ha ingressat a la pàgina “home”.

1) L'usuari visualitza que a la pestanya “sol·licituds d'amistat” hi ha una nova sol·licitud entrant. I llavors fa click sobre aquesta pestanya per veure quí és el que li ha fet l'ha realitzat (en aquest

cas, l'usuari veu que té una però podria tenir vàries sol·licituds acumulades a la espera d'alguna resposta).	
Cas normal	**Cas alternatiu**
2a) El receptor accepta la sol·licitud per mitjà del botó corresponent.	2b) El receptor rebutja la sol·licitud per mitjà del botó corresponent.
3a) El sistema canvía l'estat de la sol·licitud i el passa a estat d'"amistat" tot actualitzant l'estat de l'entrada corresponent de la BBDD.	3b) El sistema suprimeix la sol·licitud d'amistat de la BBDD. En aquest cas, l'emissor de la sol·licitud mai sabrà si ha estat rebutjat (pensarà que està pendent).
Postcondició: en aquest cas la petició d'amistat es converteix en l'establiment d'un nou vincle d'amistat.	**Postcondició:** desapareix la petició d'amistat

X. Eliminació d'un usuari vinculat

En aquest cas, s'ha d'entendre que el vincle que hi havia era d'amistat.

Actor: usuari que desitja el·liminar el vincle d'amistat (usuari *actiu*)
Precondició: hi ha una relació previa d'amistat entre l'usuari *actiu* i el *passiu*. A més, l'usuari *actiu* es troba a la pàgina de "home".
1) L'usuari *actiu* cerca l'usuari que desitja el·liminar a dintre de la pestanya d'"amistats". En aquest moment, prem el botó "el·liminar" associat a l'usuari *passiu.*
2) El sistema elimina el vincle d'amistat entre els dos usuaris. Per tornar a reestablir el vincle s'hauria de procedir amb el "protocol" d'establiment d'amistat comentat en casos d'ús previs.
Postcondició: Ni usuari *actiu* ni *passiu* mantenen cap vincle. L'usuari *actiu* segueix estant a "home".

XI. Escritura de comentari a la zona Street

Com s'ha esmentat en el capítol anterior, la zona *Street* és la zona en la qual els usuaris poden compartir els seus comentaris amb la resta d'integrants de la xarxa.

Actor: usuari que desitja compartir un comentari amb la resta.
Precondició: l'usuari es troba a "home".
1) L'usuari es dirigeix cap al quadre de text de l'àrea *Street,* introdueix el seu comentari i prem el botó "Enviar".
2) El sistema afegeix una nova entrada a la taula de la BBDD relacionada amb *Street.* A més, mostra el comentari a l'àrea addient, donant informació sobre l'usuari que l'ha escrit (imatge

personal i nom) i dóna informació sobre la data en la que s'ha afegit el comentari.
Postcondició: tots els usuaris de la xarxa poden visualitzar el nou comentari.

XII. **Enviament de correu electrònic a un contacte**

Un usuari pot enviar missatges personals i privats a un altre usuari.

Actor: usuari (*Emissor*) que desitja enviar un mail a un altre usuari (*Receptor*).	
Precondició: l'usuari *Emissor* es troba a la pàgina "home" correctament identificat. A més, existeix una relació d'amistat entre l'*Emissor* i el *Receptor.*	
1) L'*emissor* realitza click sobre el botó d'enviament de mail que hi ha al cantó de l'usuari *receptor,* a dintre de la llista d'amics.	
2) El sistema li mostra un quadre en el qual pot afegir l'assumpte del mail i el missatge.	
Cas normal	**Cas alternatiu**
3a) l'*emissor* fa click sobre el botó "Enviar".	3b) L'*emissor* fa click sobre el botó "Cancel·lar"
4a) El sistema afegeix una nova entrada a la BBDD. Aquesta nova entrada representa el mail enviat. Conté l'assumpte, el missatge i la data d'enviament.	4b) El sistema oculta el panell d'enviament de mails.
Postcondició: l'*emissor* es troba a la pàgina "home" i hi ha una nova entrada a la BBDD en forma de mail que vincula a l'*emissor* i *receptor.*	**Postcondició:** mateixa situació que a la precondició.

XIII.Lectura de mails entrants

Quan un usuari accedeix per primer cop a la pàgina "home", el sistema realitza una consulta inicial sobre els mails que té i li indica el número de mails pendents de ser llegits.

Actor: usuari que vol consultar els mails entrants (tant els llegits com els pendents).	
Precondició: l'usuari *Emissor* es troba a la pàgina "home" correctament identificat. Suposarem que té algun mail a la safata d'entrada encara que també podriem suposar que no en té cap. En aquest cas, simplement veuria la llista de mails buida.	
1) L'usuari realitza click al botó de lectura de mails.	
2) El sistema redirecciona a l'usuari cap a la pàgina de mails entrants i li mostra el llistat de mails que ha rebut. Per cada mail, mostra la informació de l'*emissor*, l'assumpte del missatge, l'estat (llegit o no), dóna la opció d'esborrar-lo, i, òbviament, de llegir-lo.	
Cas normal	**Cas alternatiu**
3a) L'usuari fa click sobre el botó "Llegir" d'algun	3b) L'usuari fa clik sobre el botó "Esborrar".

dels mails.	
4a) Si el mail estava en estat “pendent”, passa a l'estat “llegit”.	4b) El sistema esborra la entrada corresponent en la BBDD i esborra el mail de la llista que estava visualitzant l'usuari.
5a) El sistema mostra un panell a on l'usuari pot visualitzar el contingut del missatge. A més, en aquest panell, a l'usuari se li dóna la opció de respondre el mail (els usuaris poden intercanviar mails indefinidament).	
Postcondició: el mail queda marcat com a llegit tot i que l'usuari el podrà consultar tantes vegades com vulgui.	**Postcondició:** el mail queda esborrat del sistema i l'usuari ja no el podrà tornar a llegir mai.

3. Estat de l'art i tecnologia emprada

En aquest capítol es donarà una visió de la tecnologia usada i es tractarà d'explicar cada part que composa el projecte d'una manera acurada sense arribar a explicar la totalitat de cada component ja que, de cada element se'n podria parlar molt.

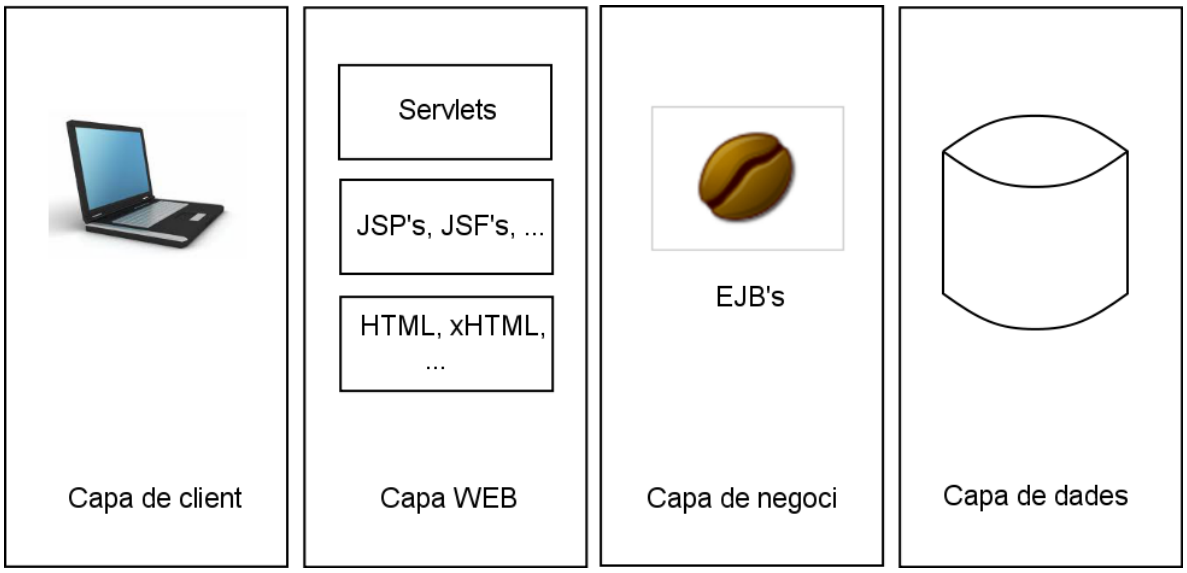
3.1. J2EE

Són les sigles de "*Java 2 Enterprise Edition"* i es tracta d'una plataforma de desenvolupament per la creació d'aplicacions escrites en Java, les quals estan destinades principalment a la prestació de serveis a través d'Internet.

Un dels principals avantatges d'aquesta tecnologia és que permet el desenvolupament d'aplicacions amb cost molt reduït. A més, aquest entorn de treball es pot importar fàcilment des de l'*Eclipse* (IDE gratuït emprat en aquest projecte).

Un altre avantatge que ofereix aquesta tecnologia és la portabilitat *("write once, run anywhere").* Al ser una plataforma basada en Java, no cal haver de compilar el codi per cada nova arquitectura ni sistema operatiu de cada servidor en el qual es vulgui instal·lar.

L'arquitectura bàsica que ofereix J2EE és la que es mostra a continuació:



En aquest model primer tindriem la capa del client, que és la que representa a l'usuari final que fa servir l'aplicació web. En aquesta capa tindríem el navegador web, que és l'encarregat d'interpretar el codi (HTML, ó xHTML, etc) que és enviat pel servidor i llavors representar a l'usuari tots els elements que li permetran realitzar accions i rebre informació del sistema (botons, quadres de text, imatges, etc).

Posteriorment tindríem la capa web, que principalment és l'encarregada de proporcionar i construïr una presentació del sistema cap a l'usuari i a més, és una passarel·la cap a capes més internes del sistema. És a dir, ofereix una interfície entre l'usuari i la lògica de negoci i dades que hi ha al servidor. Per tant, aquesta capa és l'encarregada de rebre les peticions dels usuaris i traslladar-les a capes més profundes, i alhora, és l'encarregada de servir les respostes i enviar el codi (HTML, xHTML, etc) i dades perquè siguin interpretades pel navegador web i llavors presentades a l'usuari de manera ordenada. En aquesta capa és on resideixen els *Servlets*, els quals són programes escrits 100% en Java i que poden rebre peticions d'un o més clients. Els *Servlets* comencen a executar-se quan s'inicia el servidor i segueixen executant-se fins que el servidor els hi ho indica (alguns servidors no aturen els servlets fins que no s'aturen). Un *Servlet* és una classe Java que ha d'heretar de la classe *HttpServlet* i ha de sobreescriute els següents mètodes:

- **init()** Aquest mètode és opcional el reimplementar-lo i és l'encarregat de realitzar totes les tasques d'inicialització pertinents, com per exemple, la inicialització de la connexió a la BBDD. A més, aquest mètode és executat quan es genera una instància del *Servlet*, és a dir, s'executa quan el *Servlet* és carregat per primer cop a la memòria del servidor d'aplicacions.
- **service()** S'executa després del mètode *init()* i és obligatori el reimplementar-lo. A més, conforma el nucli funcional del *Servlet* i explota els recursos reservats previament.
- **destroy()** Aquest mètode s'executa després del mètode *service()*, i, a l'igual que el *init(),* és opcional el reimplementar-lo. La seva funció principal és la d'alliberar els recursos adquirits al mètode *init().*

La tecnologia *JSP* (*JavaServer Pages*) també està continguda dintre la capa web. Es pot considerar com una manera alternativa i simplificada de construïr *Servlets*. Les *JSP* són fitxers amb extensió '.jsp' que, en essència, convinen codi HTML amb fragments de codi escrits en Java. En aquest cas, el servidor d'aplicacions s'encarrega d'interpretar el codi contingut en la pàgina *JSP* i de construïr el codi Java del *Servlet a generar.* I, alhora, aquest *Servlet* és l'encarregat de generar el fitxer que serà interpretat pel navegador del client.

La tecnologia *JSF* (*JavaServer Faces*) és la tecnologia de la capa web emprada en l'eboració d'aquest projecte. Es donaran més detalls en posteriors punts de la memòria però en essència, es tracta d'un framework per aplicacions web basades en Java, que permet generar components al servidor per la interfície d'usuari.

Seguint amb l'explicació del model de quatre capes, després tindriem la capa de negoci, la qual està conformada principalment per *Beans.* Els *Beans* es poden entendre com unes porcions de codi (classes escrites en Java) que encapsulen la lògica de negoci d'una aplicació. Un exemple de *Bean* que s'ha implementat en aquest projecte és l'anomentat *MailBean*, i conté, entre d'altres operacions, operacions com *"enviarMail()"*, o *"cancelarEnviament()"*, les quals, internament realitzen tasques relacionades amb la seva signatura.

Per últim, tindriem la capa de dades. Aquesta capa és la encarregada de donar persistència de la informació de l'aplicació i bàsicament està conformada pel sistema de gestió de BBDD. Per tal de connectar aquesta capa amb les esmentades anteriorment, es fa servir l'API *JDBC* (*Java Database Connection*), el qual permet accedir a la BBDD des de codi Java. A més, aquest API ofereix una cosa molt interessant, i és que permet el tractar consultes SQL directament com objectes Java. Per exemple, seguint amb l'exemple dels mails, al projecte es realitza una consulta que retorna un objecte de tipus *ArrayList<Mail>*, a on *Mail* representa un mail amb estructura *POJO* (*Plain Old Java Object,* classe molt bàsica amb només atributs, getters i setters). Per acabar amb aquesta capa, queda comentar que el seu emplaçament físic no té perquè ser el mateix servidor que conté l'aplicació. Pot ser que, per raons de seguretat o per rendiment, el servidor que conté la BBDD estigui a un altre servidor més potent i amb major protecció. A més, sempre que es pugui s'hauria d'intentar allotjar aquesta capa en un servidor relativament potent ja que les consultes a les BBDD són els principals colls d'ampolla de qualsevol aplicació pel que fa a la velocitat de resposta, sobretot si hi ha una gran quantitat de dades.

3.2. JSF (JavaServer Faces)

Es tracta d'una tecnologia desenvolupada per *JCP* (*Java Community Process*) sota el *JSR* (*Java Specification Request*) 314. *JSF* estableix un estàndard per l'elaboració d'interfícies d'usuari des del costat del servidor. A més, exposa un model de programació orientat a components i a events, és a dir, el fluxe dels programes és determinat pels events dels components (click a un botó, pulsació d'una tecla, etc).

Un dels principals avantatges que ofereix aquest framework és que permet realitzar aplicacions web amb una dificultat moderada. A més, els desenvolupadors no han de conèixer a la perfecció el cicle HTTP complert de petició-resposta ni tampoc han de descodificar i interpretar peticions.

La estructura de directoris i fitxers fonamental d'un projecte *JSF* és la següent:

- **src/** Aquest directori conté codi Java. Bàsicament conté la implementació dels *Beans* necessaris per l'aplicació i també la lògica per l'accés a la BBDD.
- **WebContent/** És el directori que conté, entre d'altres coses, la implementació dels fitxers '.jsp' o '.html', etc. de l'aplicació. No obstant, no és oblitarori que aquests fitxers estiguin en aquest directori.
- **WebContent/WEB-INF/lib/** És el directori a on s'han d'allotjar els fitxers '.jar' (un '.jar' és una llibreria Java comprimida en un '.zip') compartits.
- **WebContent/WEB-INF/faces-config.xml** És un fitxer necessari ja que en ell es defineixen els *ManagedBeans* (*Beans* que es fan servir a *JSF*) i també conté informació sobre el que es coneix com “ruta de navegació”. Donada la seva importància, el contingut d'aquest fitxer s'explicarà més endavant amb una mica més de detall.
- **WebContent/WEB-INF/web.xml** Aquest fitxer és un fitxer important. És conegut com a descriptor del desplegament i conté informació sobre els recursos requerits per l'aplicació. També es donarà una mica més de detall a continuació.

faces-config.xml Com s'acaba de comentar, aquest fitxer conté la declaració dels *ManagedBeans* i rutes de navegació.

Un *ManagedBean* és una classe Java que té atributs amb estructura *POJO* i que a més, pot contenir mètodes que realitzin operacions més complexes, com ara consultes a BBDD. L'aspecte que té la definició d'un *ManagedeBean* dintre del fitxer *faces-config.xml* és la que es mostra a continuació.

```
<managed-bean>

    <description>Descripció de les principals funcions del ManagedBean.
                    </description>

     <managed-bean-name>nomBean</managed-bean-name>

     <managed-bean-class>paquet.classe</managed-bean-class>

     <managed-bean-scope>tipus</managed-bean-scope>

</managed-bean>
```

En aquesta definició, tenim que 'nomBean' és el nom que servirà per fer referència a aquest *ManagedBean* des d'altres *ManagedBeans* o des del propi codi *JSP* o *xHTML,* etc. Així mateix, 'paquet.classe' no és més que el path complert (a nivel de paquets Java) cap a la classe que implementa totes les operacions del *ManagedBean.* Un exemple, seria *com.uocbook.MailBean*. Finalment, el 'tipus' indica l'abast del *ManagedBean* i pot ser de quatre tipus diferents:

- **none** Es fa servir en aquells *ManagedBeans* que pertanyen a d'altres.
- **request** Amb aquest valor, el *ManagedBean* té una vida curta. Només existeix des del moment en que s'envia la petició *HTTP* al servidor, fins justament després d'enviar la resposta cap al client.
- **session** L'abast del *ManagedBean* dura el temps des de que s'estableix la sessió fins que aquesta es finalitza. La sessió pot finalitzar tant si l'aplicació es tanca manualment per l'usuari com si es venç el timeout d'inactivitat. Per tant, els *ManagedBeans* d'aquest tipus poden perdurar al llarg de diverses pàgines de l'aplicació.
- **application** Els *ManagedBeans* d'aquest tipus perduren el temps de vida de l'aplicació.

Pel que fa a les rutes de navegació, aquestes també estan definides al fitxer *faces-config.xml.* A continuació es mostra una estrucura de navegació en el cas en que es puguin donar dos destinacions diferents. No és obligatori que tingui només dues opcions, de fet en pot tenir una o més.

```
<navigation-rule>

          <from-view-id>pàgina_origen</from-view-id>
           <navigation-case>
                 <from-outcome>etiqueta_1</from-outcome>
                 <to-view-id>pàgina_destí_1</to-view-id>
                 <redirect/>
          </navigation-case>
          <navigation-case>
```

```
            <from-action>managedBean.mètode</from-action>
            <from-outcome>etiqueta_2</from-outcome>
            <to-view-id>pàgina_destí_2</to-view-id>
            <redirect/>
        </navigation-case>

    </navigation-rule>
```

En aquest cas, en la primera opció de navegació podem veure el tag *from*-outcome. Aquest tag conté una cadena que és la que redirecciona cap a la pàgina destí. És a dir, aquesta cadena ha d'estar dintre del paràmetre *action* d'algun botó de la pàgina *pàgina_origen.* I, quan es premi aquest botó, el *FacesServlet* s'encarregarà de readreçar l'aplicació cap a la pàgina *pàgina_destí_1*. Un exemple de botó seria el següent:

```
<h:commandButton value="Nom del botó"
                 action="etiqueta_1" />
```

Pel que fa al segon cas de redireccionament, és similar al cas anterior, però en aquest cas es pot veure que tenim el tag *from-action.* Aquest tag indica el mètode, del *ManagedBean* corresponent, que provocarà el redireccionament. I, aquest mètode ha de constar necessàriament a l'atribut *action* d'algun botó de la pàgina d'origen. Això és útil en el cas de que es vulguin realitzar accions previes al redireccionament, com ara, una consulta a la BBDD. Un exemple de botó seria el següent:

```
<h:commandButton value="Nom del botó"
          action="#{managedBean.mètode}" />
```

I, el *ManagedBean* hauria de implementar aquest mètode, tal i com es mostra a continuació:

```
class Bean {
      ...
      public String metode() {

      .....  accions a realitzar abans del redireccionament ...

          return "etiqueta_2";
      }
}
```

web.xml Aquest fitxer pot contenir multitud de paràmetres que defineixen com es comportarà l'aplicació i quins elements tindra. A continuació s'enumeren els elements més representatius que podem trobar en aquest fitxer:

- **Definició dels *Servlets* de l'aplicació** En el cas de *JSF*, és necessari que hi sigui la declaració del *Servlet* `javax.faces.webapp.FacesServlet`.
- ***Servlet mapping*** Cada Servlet de l'aplicació ha de tenir el seu *Servlet mapping*. Aquest element indica un patró de URL associat al *Servlet* en qüestió. És a dir, es mapeja un URI amb el *Servlet* que l'ha de tractar. Per exemple, al present projecte s'ha inclòs el següent mapeig, amb el qual s'està indicant que totes les pàgines amb patró "/faces/*" han de ser tractades pel Servlet "Faces Servlet".

```
<servlet-mapping>

      <servlet-name>Faces Servlet</servlet-name>

      <url-pattern>/faces/*</url-pattern>

</servlet-mapping>
```

- **El número màxim de vistes d'una sessió** Aquest paràmetre indica el número màxim de pàgines que el servidor ha de guardar per poder realitzar l'acció del botó de "fer enrere".
- **Timeout de sessió** Indica el timeout màxim d'inactivitat a partir del qual la sessió es donarà per tancada**.**
- ***Saving state method*** Indica a on es guarda l'estat dels elements de cada vista dintre d'una sessió. Les possibles alternatves són al navegador del client, o al servidor. Si l'estat es guarda al client, l'estat de tota la vista (la pàgina) s'emmagatzema de manera oculta a un camp de la pàgina. En canvi, si l'estat es guarda al servidor, aquest es carrega directament a la seva memoria RAM.

Per acabar amb aquest punt, només resta parlar del fitxer *.WAR* (*Web application ARchive*). Aquest fitxer és el que conté tots els elements de la web de manera comprimida. Dintre en trobem les llibreries necessàries, en forma de *.jar's*; els fitxers HTML; les classes Java compilades (*.class*); XML's de configuració; i els recursos i imatges que necesssita l'aplicació. Aquest fitxer és el que es desplega al servidor d'aplicacions.

3.2. Servidor d'aplicacions

El servidor d'aplicacions *J2EE* triat per contenir l'aplicació ha estat JBoss en la versió 6.0.0. Es tracta d'un servidor d'aplicacions àmpliament usat en molts entorns empresarials, com és el cas de *El Comercio*, companyia asseguradora d'Argentina, o *Citi Street,* empresa d'Estats Units dedicada a oferir serveis financers.

JBoss és un servidor Open Source, i de lliure difusió, implementat en codi Java. Per aquesta raó, pot córrer en qualsevol sistema operatiu que contingui una màquina virtual Java. JBoss, a més es pot integrar molt bé amb Eclipse. I es pot dir que està mantingut i recolzat per una àmplia comunitat de desenvolupadors de tot el món. Un cop instal·lat, els directoris més importants que trobem al directori d'instal·lació són els que es mostren a continuació:

- **jboss-6.0.0/bin** En aquest directori es troben els binaris, *.jar*'s i scripts propis del servidor.

- **jboss-6.0.0/server/default/deploy** Directori a on es despleguen els fitxers (*.jar* o *.war*) que conformen les aplicacions.
- **jboss-6.0.0/server/default/conf** Directori que conté fitxers de configuració dels possibles serveis que es poden oferir, com per exemple, serveis de correu electrònic o serveis de seguretat (*JAAS*), etc.
- **jboss-6.0.0/cserver/default/log** Conté els logs que es van generant quan el servidor és arrencat per primer cop (*boot.log*) i els logs generats a mida que es van executant les aplicacions contingudes (*server.log*).
- **jboss-6.0.0/common/lib/** Llibreries d'ús comú per totes les aplicacions contenides. En el present projecte, en aquest directori s'ha inclòs la llibreria necessària pel driver *JDBC.*

3.3. Sistema de gestió de base de dades

El sistema de gestió de BBDD fet servir ha estat *MySQL* en la seva versió 5.5.12. És tracta d'una eina també *Open Source* i que es pot fer servir des d'un gran número de lleguatges de programació i tecnologies web. Es tracta d'un sistema molt estès, amb més de sis milions d'instal·lacions a tot el món gràcies a la seva estabilitat, i entre els seus usuaris principals, en podem trobar *Yahoo!* A més, es tracta d'un motor de BBDD molt obert i extensible, oferint multiples variacions, com per exemple *Berkeley DB* (sistema de BBDD embegudes amb API per treballar amb C, C++, Perl i altres llenguatges), o *InnoDB* (sistema que ofereix gran integritat referencial), entre d'altres.

Pel que fa al rendiment, optimitza molt els recursos de la màquina, pel que fa a CPU, espai en disc i memòria. I dóna una gran resposta en sistemes Windows i millor encara, en sistemes Linux o basats en Unix.

Per últim, un altre avantatge que ofereix és la seva relativa simplicitat i la facilitat de gestionar i administrar les BBDD.

3.4. Hibernate

Es tracta d'una part fonamental del sistema ja que permet la existència d'una interconnexió entre el servidor d'aplicacions i el sistema de gestió de BBDD. És a dir, gràcies a aquest framework s'ha pogut traslladar el món *MySQL* cap al món Java. Aquesta és, de fet, la connexió que s'ha establert en aquest projecte. No obstant, es poden donar altres convinacions, per exemple, *.NET* amb *MySQL* o Java amb *Informix*, entre d'altres. De fet, aquest és un dels grans potencials d'aquesta tecnologia, i és que, si en algun moment cal realitzar una migració en el sistema de gestió de BBDD cap a un altre, es pot fer amb relativa facilitat i, el que és més important, el codi font de l'aplicació no es veuria afectat.

Per aconseguir aquesta connexió, hibernate necessita tenir present el mapeig que existeix entre una classe Java i una entitat *MySQL*. És a dir, ha de conèixer com es relaciona cada atribut de la entitat en la taula de la BBDD amb cada atribut de la classe *POJO* corresponent. Aquest mapeig s'aconsegueix establir gràcies als fitxers amb extensió *.hbm.xml* (*hibernate mapping xml*). Seguint amb l'exemple de la taula que conté els mails en aquest projecte, el fitxer *Mail.hbm.xml* relaciona els atributs de la BBDD i de la classe *POJO* amb la següent sintaxi:

```
<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  ..............
        <property name="missatge" type="java.lang.String">

            <column name="missatge" />

        </property>
  ..............
</hibernate-mapping>
```

El tag *property* indica el nom de l'atribut de la classe *POJO* corresponent i, com es pot veure en aquest cas és de tipus *String.* D'altra banda, el tag *column* indica el nom de l'atribut dintre de la taula en qüestió que, en el cas del projecte és de tipus *VARCHAR*.

Hibernate també necessita informació que li indiqui cóm es connectarà l'aplicació a la BBDD. Aquesta informació li ve donada pel fitxer *hibernate.cfg.xml*, el qual és obligatori i, en el cas del present projecte té el següent aspecte.

```
<property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>

<property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/uocbook</property>

<property name="hibernate.connection.password">uocbookpswd</property>

<property name="hibernate.connection.username">uocbookuser</property>
```

En aquest fragment del fitxer de configuració podem veure els punts:

- **hibernate.connection.driver_class** Indica el tipus de driver que farà servir l'aplicació per connectar-se a la BBDD.
- **Hibernate.connection.url** Indica la URL completa de la BBDD. En aquest cas, es pot veure que la BBDD es troba a la mateixa màquina que el servidor d'aplicacions (localhost). També es pot veure el port, que està monitoritzant el servei *MySQL*, a través del qual s'enviaran les consultes SQL. Finalment, es pot veure el nom de la BBDD (*uocbook*).
- ***Password i username*** Són el nom i la contrasenya de l'usuari que té permís per accedir a la BBDD, realitzar consultes i afegir files a les taules addients.

A banda d'aquesta informació, altres elements que s'han de trobar necessàriament dintre d'aquest fitxer són aquells que indiquen quins són els fitxers *.hbm.xml* que s'han de considerar. Aquesta informació es dóna amb el següent tag (donant l'exemple de l'entitat *Mail* del projecte):

```
<mapping resource="database/entitats/Mail.hbm.xml" />
```

A part d'aquesta informació, també es poden trobar altres tags opcionals que indiquin altres característiques sobre com el sistema es connecta a la BBDD.

4. Implementació

En els apartats anteriors hem vist els casos d'ús que s'han implementat des d'un punt de vista funcional. Després, en el capítol anterior, s'han explicat les eines que s'han fet servir en la realització d'aquest projecte des d'un punt de vista teòric.

En el present capítol es donarà una visió referent a com està implementat el projecte, fent servir les eines explicades en el capítol 3, per arribar als requeriments funcionals donats en el capítol 2. El capítol es divideix en les tres parts que conformen el patró de disseny *MVC.* És a dir, es divideix en la part de la *Vista*, la part del *Model* i en la part del *Control*.

4.1. Vista de l'aplicació

En aquest apartat es donarà una explicació de l'aplicació pel que fa a la seva vista, és a dir, pel que fa a l'interfície d'usuari.

4.1.1. Pàgines

L'aplicació consta de sis pàgines, cinc amb format *.xHTML* i una amb format *.jsp*. La pàgina que està en format *.jsp* és la més senzilla de totes i es tracta de la pàgina de índex, és a dir, és la primera pàgina que es visita i el que fa és redireccionar a l'usuari directament cap a la pàgina de *login-registre.*

El contingut de *index.jsp* és el següent:

```
<html>
	<head> </head>
	<body>
		<% session.invalidate(); %>
		<jsp:forward page="faces/loginRegistre.xhtml" />
	</body>
</html>
```

Aquesta pàgina, primer neteja la sessió (*session.invalidate()*) per eliminar el login de l'usuari en cas de que l'hagués i després redirecciona cap a la pàgina *loginRegistre.xhtml.* Com es pot apreciar, la URL conté la cadena “faces/*”. Amb això s'està indicant que el *Servlet* encarregat de gestionar la presentació d'aquesta pàgina ha de ser el *JavaServlet Faces* tal i com s'indica al fitxer *web.xml*

```
<servlet>
	<servlet-name>Faces Servlet</servlet-name>
	<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
	<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
	<servlet-name>Faces Servlet</servlet-name>
	<url-pattern>/faces/*</url-pattern>
</servlet-mapping>
```

Les altres cinc pàgines són: *home.xhtml*; *buscarAmics.xhtml*; *editarPerfil.xhtml*; *loginRegistre.xhtml*; i *mail.xhtml*. De les quals es donarà una explicació a continuació.

I. loginRegistre.xhtml

Aquesta pàgina és la primer que veu l'usuari quan visita *uocbook.* Està dividida en dues parts diferenciades: la zona de registre; i la zona de login. El més interessant d'aquesta pàgina és el test d'integritat de camps que es realitza en el moment de registre o en el moment de login. Per exemple, en el moment del registre, pel que fa al camp *password*, es comprova que tingui una longitud superior o igual a quatre caracters. Així mateix, pel que fa al camp *mail,* es comprova que aquest almenys tingui un '.' i alhora, un caràcter '@'. Aquestes revisions d'integritat es realitzen dintre del mètode que és invocat en el moment de fer click sobre el botó *Ok* corresponent. Si els camps escrits per l'usuari acompleixen les condicions necessàries, llavors, es redireccionaria a l'usuari cap a *home.xhtml*. Si no s'acompleix alguna de les condicions, llavors, dintre del propi mètode s'escriu el missatge de resposta que llegirà l'usuari i a més, es retornaria un “” (cadena buida), la qual cosa provoca que no es redireccioni a l'usuari enlloc.

A continuació es mostra un petit fragment de codi a on es pot veure com es duu a terme aquestes comprovacions.

```
<h:form>

	<h:outputText value="Correu electrònic: " />

	<h:inputText value="#{registreBean.mail}"/>

	<p></p>

	<h:outputText value="Contrasenya: " />

	<h:inputSecret value="#{registreBean.password}"/>

	<p></p>


	<h:commandButton value="OK" action="#{registreBean.intentaEnregistrar}" />
```

```
        <p></p>
        <h:outputText style="font-size: 40; color: white; font-weight: bold;"
                      value="#{registreBean.result}" />
</h:form>
```

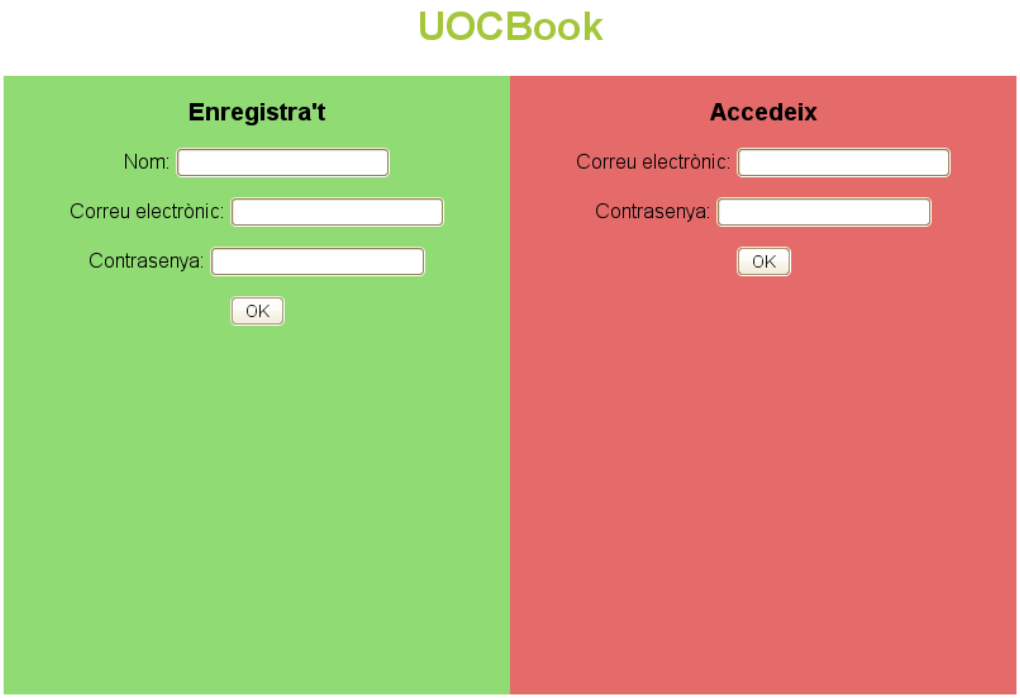
i el fragment del *RegistreBean* té el següent aspecte

```
         if(password.length() < 4){
             result = "El password ha de tenir almenys 4 caracters.";
             ret = false;
         }

         if (nom.compareTo("") == 0) {
             result = "El camp nom no pot estar buit";
             ret = false;
         }

         if (mail.indexOf("@") == -1 || mail.indexOf(".") == -1) {
             result = "Mail incorrecte";
             ret = false;
         }
```

A continuació es mostra l'aspecte d'aquesta pàgina



II. home.xhtml

Aquesta pàgina és la principal i està composada per diversos elements. La imatge que hi ha a continuació ha estat retallada una mica per no ocupar espai innecessari, però en realitat, la pàgina és més allargada verticalment. A continuació s'inclou un comentari per cadascun dels elements que es poden veure des d'aquesta pàgina

- **Imatge personal i nom de l'usuari** En la imatge es pot veure una icona amb forma de noi, que representa la imatge per defecte, i també es pot veure el nom que havia introduït l'usuari en el moment d'enregistrar-se.
- **Botó d'edició de perfil** Aquest botó és el que readreça a l'usuari cap a la pàgina *editarPerfil.xhtml.*
- **Botó amb forma de sobre** Aquest botó és el que readreça a l'usuari cap a la pàgina de correus entrants.
- **Número de correus entrants pendents de ser llegits** És el camp numèric, que apareix entre parèntesi, al costat del botó amb forma de sobre. En la imatge es pot veure que no hi ha cap correu entrant.
- **Amics** En aquest panell és a on es represent els amics de l'usuari, tot i que actualment l'usuari no ha establert una relació d'amistat amb cap usuari dintre de la xarxa. Per cada amic de la llista, l'usuari podrà realitzar tres accions: esborrar-lo de la seva llista d'amics; enviar-li un correu; o visualitzar la seva llista d'amics.
- **Peticions d'amistat** En aquest panell es mostren les peticions d'amistat, tot i que en la imatge es veu que no hi ha cap petició d'amistat pendent. L'usuari, per cada petició d'amistat, podrà realitzar dues accions: acceptar la petició o rebutjar-la.
- **Street** Aquest és l'espai compartit entre tots els usuaris de la xarxa. Quan un usuari afegeix un comentari i fa click sobre el botó *Enviar,* llavors el comentari s'afegeix just a sota del component *inputTextarea.*
- **Botó per buscar amics** Aquest botó readreça a l'usuari cap a la pàgina *buscarAmics.xhtml.*
- **Link per sortir** Aquest link el que fa és demanar a l'usuari, a través d'un panell de confirmació, que confirmi la operació de sortir de la xarxa, i, en cas de rebre una resposta positiva, es redirecciona a l'usuari cap a la pàgina de *loginRegistre.xhtml.*
- **Link per desenregistrar-se** Aquest link és anàleg a l'anterior amb la diferència de que, si l'usuari confirma la operació, aquest és redirigit a la pàgina de *logiRegistre.xhtml* i, a més, el sistema l'esborra de la BBDD (a ell i a tots els comentaris que hagi afegit a la zona comú).

Pel que fa a la implementació d'aquesta pàgina, el més destacable seria l'ús de *Managed Beans* per poder mostrar a l'usuari el nombre de mails pendents, i el nombre d'amics i de peticions d'amistat. A més, en aquesta pàgina s'introdueix un nou component, anomenat *panelSeries*, i que prové de *ICEFaces,* framework també *Open Source* i que afegeix nous components i efectes visuals als que ja té de per sí el framework *JavaServer Faces.*

A continuació es mostra el panell d'amistats-peticions mostrant que l'amic té dos amics i una petició d'amistat.

En aquest panel es pot veur que l'usuari *Ivan Folgueira* té dos amics i té una nova petició d'amistat de *Maria Garcia*. Si l'usuari rebutjés aquesta petició es decrementaria el número de peticions a zero i *Maria Garcia* desapareixeria de la llista de peticions.

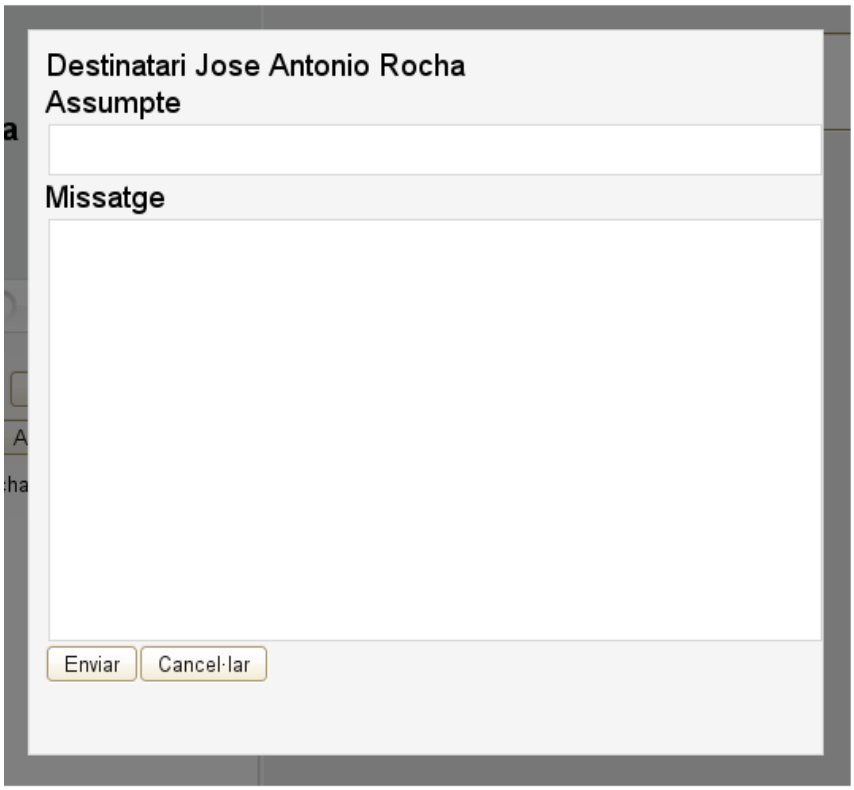
A continuació es mostra el panell d'amistat-peticions que mostra els tres amics de l'usuari d'exemple, després de que hagi acceptat la petició d'amistat de l'usuari *Maria Garcia.*

En aquest panell es pot veure que, per cada usuari, l'usuari *Ivan Folgueira* podrà eliminar-lo, enviar-li un mail o veure els seus amics, aquesta última opció és interessant per tal de trobar amics comuns. De fet, és una manera d'establir contactes dintre de la xarxa.

Si per exemple, l'usuari *Ivan Folgueira* vol veure els amics de l'usuari *Maria Garcia* per veure si coneix algun, quan faci click sobre el botó *Amics*, el sistema mostrarà la llista dels amics de *Maria Garcia* que encara no tenen una relació d'amistat amb l'usuari *Ivan Folgueira*. A continuació tenim el panell que mostra l'aplicació en aquest cas.

Per cada usuari de la llista, tal i com es pot veure, l'usuari *Ivan Folgueira* podrà fer-li una petició d'amistat. Si no desitja fer cap peticó llavors simplement ha de prémer el botó *Ok* i s'amagaria aquest panell sense fer res més.

Suposem ara que *Ivan Folgueira* vol enviar un correu al seu amic *Jose Antonio Rocha.* En aquest cas, l'usuari ha de prémer el botó amb forma de sobre i, el sistema li mostrarà el següent panell.



Ara suposem que s'envia el correu amb les següents dades

- **Assumpte** Feli**c**itaci**ó** Nadal
- **Missatge** Hola, què tal ? Bon Nadal !

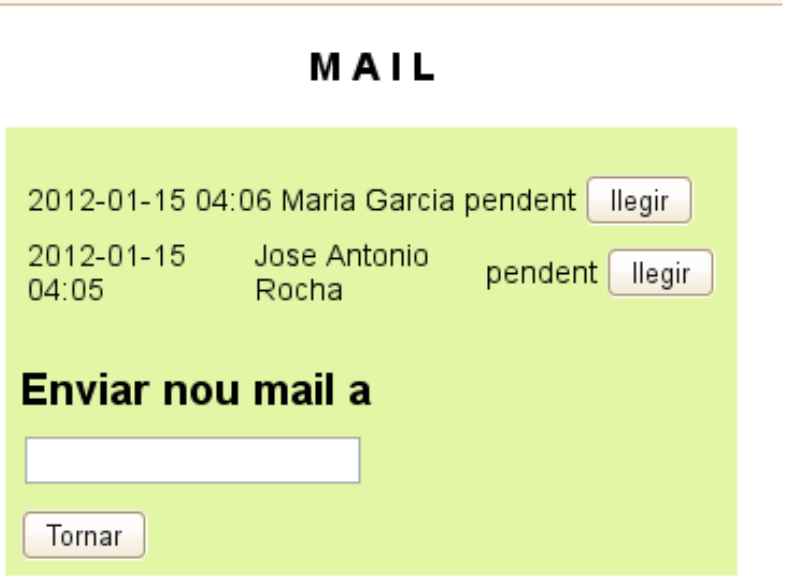
Suposem ara que l'usuari vol afegir un comentari a la zona comuna *Street.* El que veurien ell i la resta d'usuaris seria el següent:

En aquesta pantalla es pot veure que l'usuari té dos mails pendents de ser llegits. Considerem ara, que l'usuari fa click sobre el botó amb forma de sobre per tal d'anar a la pàgina *mail.xhtml.*

III. **mail.xhtml**

En aquesta pàgina es poden veure els mails rebuts per l'usuari enregistrat al sistema. A més, a partir d'aquesta pàgina també es poden respondre els mails rebuts i enviar nous mails a altres usuaris que tinguin una relació d'amistat amb l'usuari *Ivan Folgueira*.

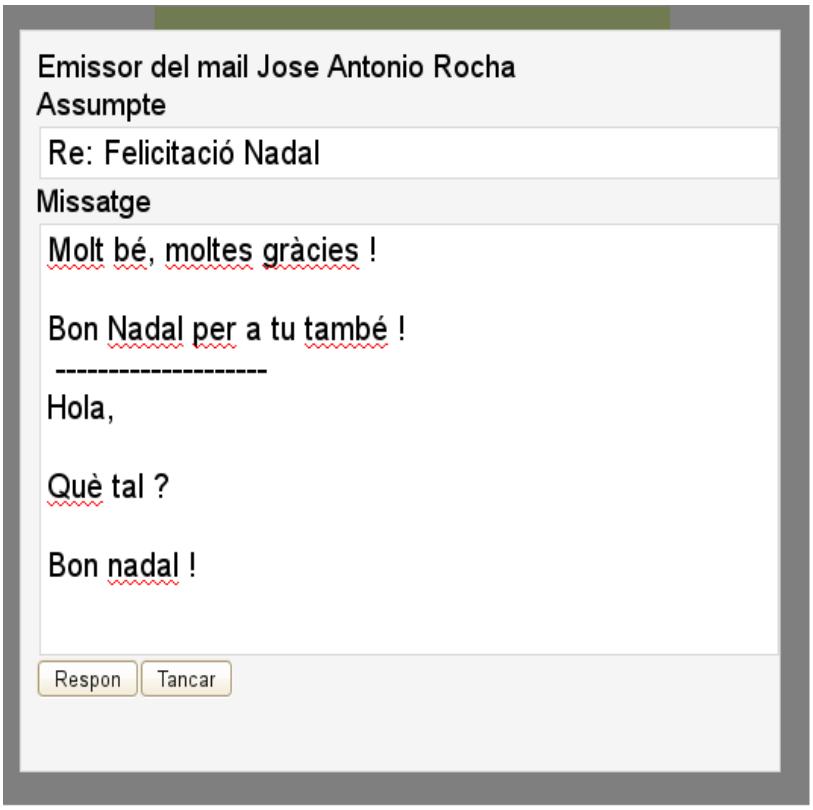
A continuació es mostra l'aspecte d'aquesta pàgina, seguint amb el fil del punt anterior, després de que l'usuari hagi premut el botó per mirar els mails rebuts en la pàgina *home.xhtml.*



En aquesta pàgina es pot veure els dos mails que ha rebut l'usuari. Un mail prové de l'usuari *Maria Garcia,* mentre que l'altre prové de l'usuari *Jose Antonio Rocha.* A més, es pot veure la data d'enviament de cada mail. També cal observar que l'estat dels dos mails és *pendent.*

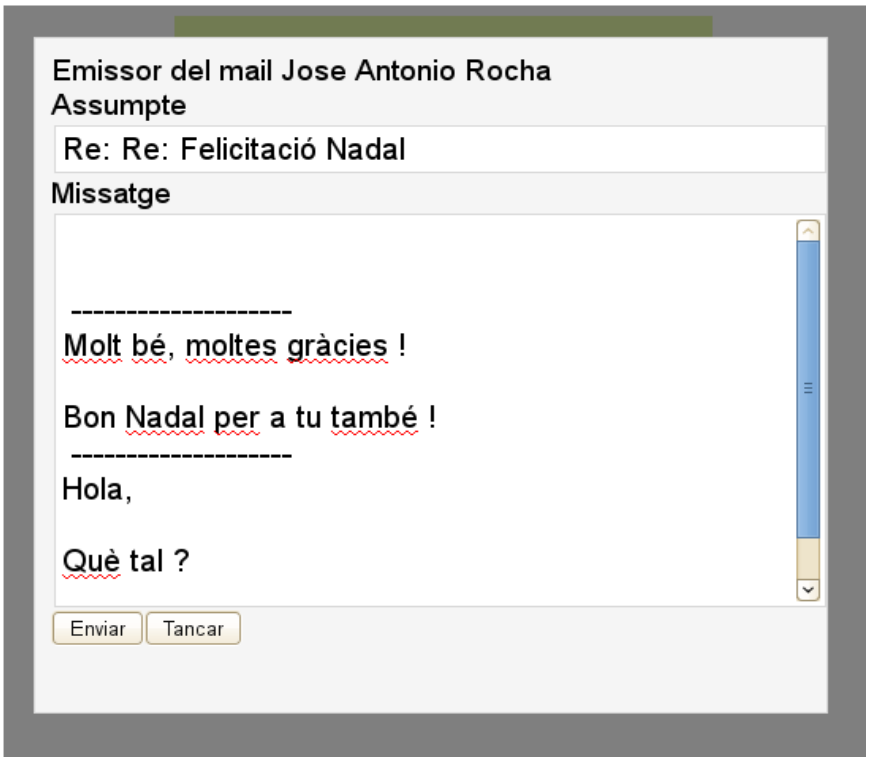
Per altra banda, a la part de més avall de la pàgina podem veure un quadre de text. En aquest quadre, l'usuari pot començar a buscar el nom d'un amic i trobar-lo ràpidament gràcies a aquest component. I, en el moment de seleccionar-lo, immediatament s'obrirà un panell per tal d'editar el nou mail. Aquest panell es pot veure a continuació, a on es veu que s'està editant un mail per enviar-li a *Juan Perez.*

Ara suposem, per exemple, que l'usuari desitja llegir el mail provinent del seu amic *Jose Antonio Rocha.* El que faria seria prémer el botó *Llegir* del mail en qüestió i el que veuria seria el següent:



En aquest cas, es pot veure que l'usuari *Jose Antonio Rocha* ha enviat un mail de resposta a l'usuari *Ivan Folgueira.* Com es pot apreciar, en el panell es pot veure qui ha estat l'emissor del missatge i també es pot veure que l'assumpte coincideix amb l'assumpte que havia escrit l'usuari *Ivan Folgueira* més la cadena "Re:" i, pel que fa al missatge, es pot veure la resposta de l'usuari *Jose Antonio Rocha* i el missatge que li havia enviat previament l'usuari *Ivan Folgueira.*

Ara suposem que l'usuari vol replicar de nou a aquest mail tot fent click sobre el botó *Respon.* Justament després de realitzar aquesta acció, el que veuria l'usuari *Ivan Folgueira* seria el següent:



Es pot veure que l'usuari podrà afegir un nou missatge i a més, el botó *Respon* "s'ha convertit" en botó *Enviar.* Bé, de fet, el que ha passat és que el botó *Respon* s'ha fet invisible mentre que el botó *Enviar* s'ha fet visible.

Amb tot això, tenim que dos usuaris poden intercanviar mails indefinidament.

Finalment, quan l'usuari torni a la pàgina *mail.xhtml* veurà que el mail procedent de *Jose Antonio Rocha* estaria en estat llegit mentre que el mail procedent de *Maria Garcia* encara estaria en estat *pendent.*

IV. editarPerfil.xhtml

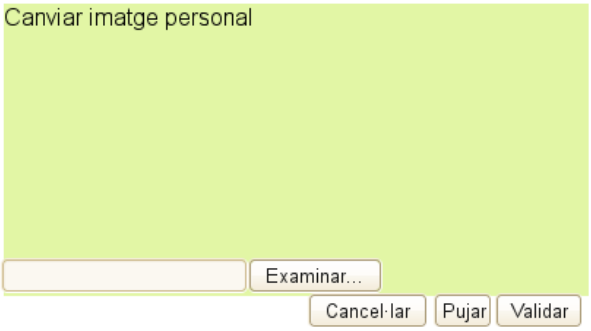
En aquesta pàgina, l'usuari pot canviar les seves dades personals. En concret, pot editar la seva imatge personal o el nom. Aquestes dades seran seran vistes per la resta d'usuaris tant a la zona *Street* de l'aplicació com als mails.

Aquesta pàgina, quan és visitada per l'usuari, el primer que aquest veu és la seva imatge i el seu nom. Si decideix canviar de nom, simplement ha d'introduir el nou nom al *inputText* corresponent.

En canvi, si desitja canviar la seva imatge, el que ha de fer és fer click sobre el botó *Canviar imatge.* En aquest moment, li apareixerà el quadre en color verd que li permetrà el pujar una nova imatge. Per pujar-la, primer ha de buscar-la pel seu PC, després d'haver premut el botó *Examinar.* Després, ha de fer click en el botó *Pujar,* el qual, el que farà serà pujar la imatge al servidor i mostrarà una vista prèvia a dintre del propi requadre verd. I si finalment l'usuari veu que queda bé la imatge, el que ha de fer és fer click sobre el botó *Validar* i, a partir d'aquest punt, la imatge de l'usuari quedaria canviada.

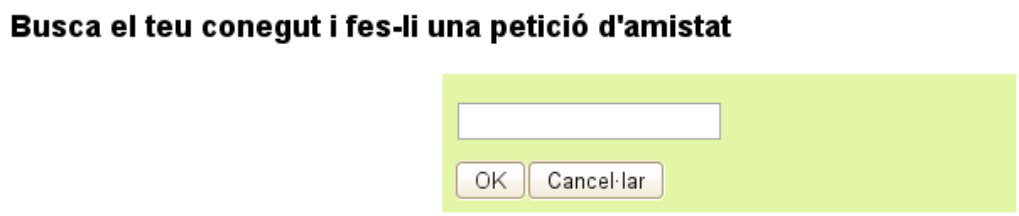
Si l'usuari fa click sobre el botó *Ok*, aquest el readreça un altre cop cap a la pàgina *home.xhtml* tant si ha canviat alguna de les seves dades personals com si no.

Pel que fa a la implementació d'aquesta pàgina, el més destacable és l'ús del component *fileEntry*, el qual pertany al framework *ICEFaces,* i és el que permet pujar un fitxer des del sistema de fitxers local fins al servidor. Aquest component té atributs amb els que es pot indicar el directori a on s'han de guardar les imatges. I també es pot indicar el tamany màxim del fitxer que es pot pujar que, en aquest cas, s'ha establert a 1048576 bytes (1MB).



V. buscarAmics.xhtml

Aquesta és una pàgina molt senzilla, des del punt de vista de l'usuari, ja que només té dos botons i un quadre per introduïr text.

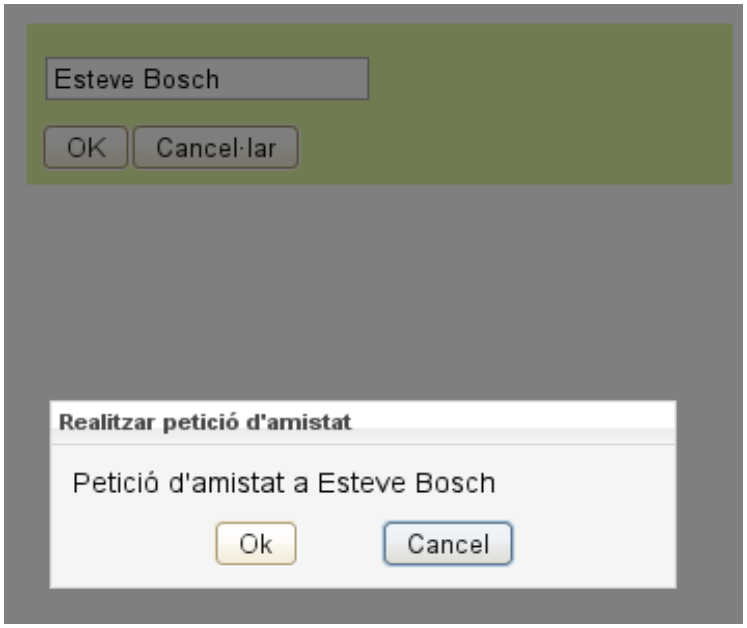


L'objectiu principal d'aquesta pàgina, com el seu propi nom indica, és el de buscar amics a tota la xarxa per tal de realitzar peticions d'amistat.

El més destacable d'aquesta pàgina és el component *selectInputText*, que també pertany al framework *ICEFaces.* Es tracta d'un component molt interessant ja que permet realitzar accions a mida que l'usuari va introduint caracters. De fet, aquest component és el mateix que s'ha fet servir a dintre de la pàgina *mail.xhtml* per tal de buscar a l'amic al qual es vol enviar un mail. L'única diferència és que en aquest cas es llisten tots els usuaris de la xarxa menys els que ja són amics o existeix una petició d'amistat pendent de confirmació en algun dels dos sentits.

El que es fa en aquesta pàgina és, per cada nou caràcter que l'usuari introdueix o esborra, es realitza una consulta per tots els usuaris de la xarxa i es llisten aquells que continguin la seqüència de caracters que tingui aquest component en aquell moment. Òbviament, quant més escrigui l'usuari, més restrictiva serà la búsqueda i, per tant, es llistarà un menor nombre de possibles amics.

Finalment, es selecciona l'usuari tot fent click sobre el seu nom dintre de la llista i, per a realitzar la petició d'amistat, l'usuari ha de prémer el botó *Ok.* Tot just després de fer això a l'usuari se li presenta un quadre de confirmació, preguntant-li si realment desitja realitzar la operació. En la següent imatge es pot veure un exemple de petició d'amistat cap a l'usuari *Esteve Bosch.*

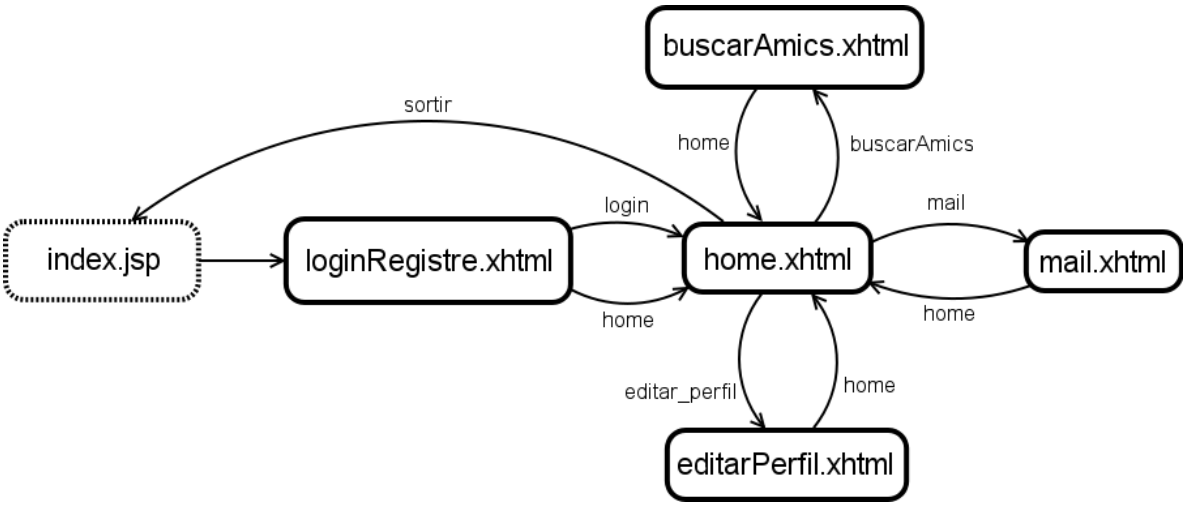


En aquest cas, si finalment l'usuari prem el botó *Ok*, la petició d'amistat es durà a terme. En canvi, si prem el botó *Cancel* no es realitzarà la petició.

4.1.2. Rutes de navegació

Un cop comentades les diferents pàgines que conformen l'aplicació, en aquest apartat es veuran les rutes de navegació d'una forma esquemàtica. Aquestes rutes de navegació, tal i com s'ha comentat en capítols anteriors, han d'estar definides dintre del fitxer *faces-config.xml*. Si no estan definides en aquest fitxer, les transicions es podran donar però l'aplicació no funcionarà correctament ja que les pàgines no hauran estat renderitzades pel *Servlet Faces*. Per exemple, el que passaria segur és que no es representaries correctament els components *JSF.*

A la següent imatge es poden veure les possibles transicions entre les diverses pàgines així com els events que les provoquen.



En aquesta imatge, les paraules que acompanyen les fletxes són els valors dels tags *from-outcome* que hi ha dintre del fitxer *faces-config.xml.* Aquests *Strings* poden ser retornats per mètodes, o poden estar directament escrites a dintre de l'atribut *action* del botó corresponent.

4.2. Model de l'aplicació

En aquest es dóna informació relacionada amb l'estructura de dades de l'aplicació, donant l'esquema E-R i justificant el disseny de la BBDD.

El model del sistema està conformat per quatre taules: USER; STREETPOST; VINCLE; i MAIL. A continuació es detalla l'estructura de cadascuna d'aquestes taules.

I. **USER** Conté informació relativa als usuaris enregistrats de la xarxa. Els seus atributs són els següents:

- **id** Es tracta d'un enter amb autoincrement i representa la clau primària de la taula.
- **nom** És un VARCHAR(20) i conté el nom de cada usuari
- **mail** És un VARCHAR(20) i conté la direcció de correu electrònic de cada usuari.
- **password** És un VARCHAR(20) i conté el password de l'usuari.
- **imatge** És un VARCHAR(50) i conté el path a on està la imatge al sistema de fitxers del servidor.

II. **STREETPOST** Conté informació sobre els posts que els usuaris de la xarxa envien a la zona *Street.* Els seus atributs són els següents:

- **id** Enter amb autoincrement. Clau primària.

- **cmt** És un atribut de tipus TEXT i conté el missatge que l'usuari ha introduit.
- **data** És de tipus DATETIME i permet emmagatzemar l'hora i la data en la qual s'ha escrit el comentari.
- **id_usuari** Clau forana que referencia la taula USER.

III. **VINCLE** Taula que emmagatzema relacions entre els usuaris. Les dues possibles formes de relació són *AMISTAT* o *PETICIO_DAMISTAT*. Els seus atributs són els següents:

- **id** Índex amb autoincrement. Clau primària.
- **id_usuari** Clau forana que referencia la taula USER. Aquest atribut és el que referencia a l'usuari que realitza la petició d'amistat.
- **id_usuari_vinculat** Clau forana que referencia la taula USER. Aquest atribut referencia a l'usuari que rep la petició d'amistat.
- **estat_vincle** Enter que indica l'estat en el que es troba el vincle. *PETICIO_DAMISTAT* (0) o *AMISTAT* (1). A dintre del *ManagedBean* corresponent es té en consideració entre estat del vincle i número 1 ó 0.

  Cal notar que, per adquirir els amics que té un usuari, s'ha de realitzar una consulta i obtenir les relacions en les que aparegui l'usuari amb *estat_vincle* igual a 1.

IV. **MAIL** Taula que permet emmagatzemar els missatges privats que intercanvien els usuaris. Els seus atributs són els següents:

- **id** Enter amb autoincrement. Clau primària.
- **id_usuari_origen** Clau forana que referencia la taula USER.
- **id_usuari_desti** Clau forana que referencia la taula USER.
- **assumpte** Atribut de tipus TEXT que representa l'assumpte indicat per l'usuari emissor del mail.
- **missatge** Atribut de tipus TEXT que conté el cos del correu enviat per l'usuari origen.
- **data** Atribut de tipus DATETIME que conté l'hora i el dia en el que s'ha enviat el missatge privat.
- **status** Enter que representa dos possibles estats: *PENDENT_LLEGIR* (0) ó *LLEGIT* (1). De la mateixa manera que succeeix en la taula VINCLE, la interpretació d'aquests valors es realitza a dintre del *Managed Bean* corresponent.

A continuació es mostra el diagrama entitat-relació de la base de dades. En aquest diagrama es pot veure com la taula VINCLE vincula als usuaris entre sí ja que cada entitat d'aquesta taula ha de contenir necessàriament dues entitats de la taula USER. D'altra banda, cada entitat de la taula USER pot estar relacionada amb un número indeterminat de entitats de la taula VINCLE ja que cada usuari pot tenir múltiples relacions d'amistat i a més, múltiples relacions en estat de petició d'amistat. Pel que fa a la relació entre USER i la taula MAIL es pot veure que cada entitat de la taula USER pot tenir un número indefinit de relacions amb la taula MAIL, ja que cada usuari pot excriure un número indeterminat de mails. En canvi, es pot veure que, per cada entitat de MAIL, aquesta ha d'estar necessàriament relacionada amb dues entitats de la taula USER, ja que cada

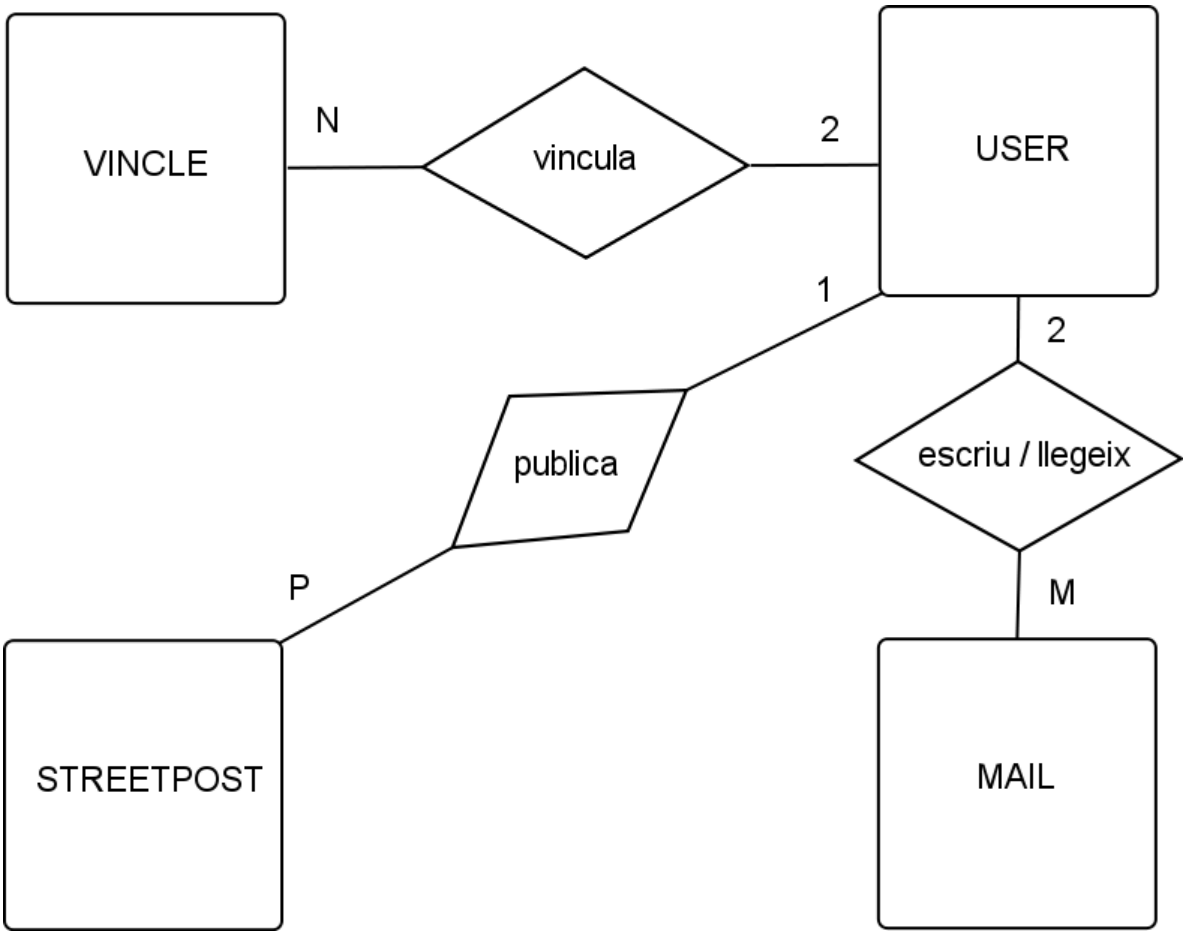
element de la taula MAIL conté a l'usuari emissor i a l'usuari receptor del mail. Finalment, la relació existent entre la taula USER i la taula STREETPOST és la més senzilla de totes. En aquest cas, es tracta d'una relació 1:P ja que cada usuari pot haver publicat diversos posts a la zona *Street*, i, a més cada post queda unívocament relacionat amb l'usuari que l'ha emès.

Per altra banda, cal comentar que la construcció de la BBDD s'ha realitzat de manera que s'aconsegueixi integritat referencial, és a dir, es garantitza que cada entitat de la BBDD sempre es relaciona amb altres entitats vàlides. Per aconseguir això, donat que la taula USER és el nucli de totes les taules (totes es relacionen amb ella), quan es construïa una nova taula que referenciava a la clau primària de USER (id) es feia servir la següent comanda *MySQL*:

....PRIMARY KEY (id), FOREIGN KEY (id_usuari) REFERENCES USER (id) ON DELETE CASCADE);

Aquesta és un fragment de la comanda per la creació de la taula STREETPOST. I amb això, el que s'aconsegueix és que, quan un usuari s'esborra de la taula USER, per què s'ha desenregistrat, immediatament s'esborren totes les entrades de la taula STREETPOST directament relacionades amb l'entrada esborrada de USER. D'aquesta manera s'aconsegueix que, quan un usuari es desenregistra, s'esborra tota la informació relacionada amb aquest usuari.

Això és així per la taula STREETPOST, ja que no té sentit que els usuaris de la xarxa puguin llegir els comentaris que ha afegit una persona que ja no pertany a la xarxa, ja que no li podran enviar ai una petició d'amistat. Aquesta integritat també es dóna a la taula VINCLE, ja que no té sentit que un usuari pugui mantenir algun tipus de vincle (d'amistat o de petició d'amistat) amb algun altre usuari que ja no existeix. En canvi, pel que fa a la taula MAIL, si un usuari es desenregistra, només s'esborraran aquelles entrades en les quals l'usuari desenregistrat consti com a receptor del mail i no com a emissor. En aquest cas, es considera que per un usuari és interessant el mantenir els mails d'un usuari que s'ha desenregistrat per tal de mantenir el “record” de l'amistat.



4.3. Control de l'aplicació

Continuant amb el patró de disseny aplicat en l'aplicació, en aquest apartat es descriurà el component referent al controlador de la mateixa, és a dir, es donarà una explicació dels elements de codi que contenen la lògica de negoci de l'aplicació. Aquests elements estan formats bàsicament per classes Java.

4.3.1. Classes

En aquest apartat es descriuen les principals classes Java que composen l'aplicació, així com les funcionalitats més importants de cadascuna d'elles. Les classes més rellevants d'aquest projecte

es poden agrupar en dos grups: un estaria conformat pels *Managed Beans*; i l'altre estaria conformat per totes aquelles classes necessàries per l'accés i control de la BBDD. Els *Managed Beans* usen les utilitats que els hi ofereixen les classes de control de BBDD.

M**anaged Beans**

I. **EditaPerfilBean**
Bean que permet la edició del perfil de l'usuari. Permet principalment pujar una nova imatge com a perfil i editar el nom o nick asocciat a l'usuari.

II. LogoutBean
Proporciona dos mètodes publics: *sortir*() i *sortirDefinitivament*(). Els dos mètodes acaben retornant l*'String "*sortir*"* per tal de readreçar a l'usuari cap a la pàgina *loginRegistre.xhtml.* No obstant, el mètode *sortirDefinitivament()* a més elimina a l'usuari de la taula USER de la BBDD.

III. MailBean
Proporciona els mètodes i atributs necessaris per tal d'anar a la pàgina *mail.xhtml*; mostrar el panell d'enviament de mails, tant a la pròpia pàgina *mail.xhtml*, com a la pàgina *home.xhtml*; i escriure o esborrar mails de la taula MAIL.

IV. PeticioAmistatBean
Ofereix atributs i mètodes que permeten a l'usuari anar a la pàgina *buscarAmics.xhtml*, realitzar la busqueda d'amics per tota la xarxa i finalment escriure la petició d'amistat a la taula VINCLE.

V. RegistreBean
Permet a l'usuari el poder enregistrar-se al sistema. Aquest *Bean* és l'encarregat de comprovar la integritat de les dades que l'usuari introdueix als *textInput'*s i, en cas de que acompleixin les condicions requerides, es redirigeix a l'usuari cap a la la pàgina *home.xhtml* i introdueix un nou usuari a la taula USER.

VI. StreetBean
Aquest *Bean* permet escriure els comentaris a la zona *Street* de la pàgina. Per tant, quan un usuari afegeix el seu comentari, aquest és introduït a la taula STREETPOST a través d'aquest *Managed Bean.*

VII. UsuariBean
Representa el *Bean* més important de tots i és referenciat en multitud de llocs de l'aplicació ja que conté la informació de l'usuari que ha ingressat al sistema. A més, ofereix les funcionalitats requerides per ingressar a la xarxa, tot comprovant la integritat de les dades introduïdes per l'usuari i comparant-les amb el que hi ha a la taula USER. I també permet acceptar o rebutjar amics.

Tots aquests *Beans* són de tipus *session* (paràmetre definit al fitxer *faces-config.xml*) ja que interessa que es guardi les seves dades mentres estigui activa la sessió de l'usuari. En canvi, *RegistreBean* i *LogoutBean* tenen àmbit de petició, és a dir, són de tipus *request* ja que només es necessita que existin a dintre de la petició *HTTP*.

Gestió i accés a la BBDD

I. **HibernateUtil**
Classe que aplica el patró *Singleton* per crear un objecte de tipus *org.hibernate.SessionFactory.*

II. **HibernateManager**
Classe abstracta que conté mètodes bàsics per la gestió de les taules de la BBDD. Per exemple, conté el mètode *afegirElement(Object o)*; el mètode *getElementById(Integer id)*; el mètode *rollback()*; entre d'altres.

III. **Classes per les taules**
MailsTable, StreetTable, UsersTable i *VinclesTable.* Per cada taula de la BBDD es té la seva classe associada per tal de realitzar operacions més específiques de cada taula. Totes aquestes classes hereten de la classe *HibernateManager.*

IV. **POJO's**
Classes que serveixen d'interfície, juntament amb els fitxers *.*hbm.xml*, per que l'aplicació es pugui connectar a la BBDD a través d'hibernate. Aquestes classes són: *Mail*; *Streetpost*; *User*; i *Vincle*.

5. Valoració econòmica

Un dels principals punts forts del projecte és el seu baix cost econòmic. Pel que fa a l'entorn d'execució del programa totes les eines que es fan servir són gratuïtes. Per tant, només s'ha de considerar el cost pel que fa al disseny i implementació de l'aplicació.

A continuació es detalla el cost associat a cada tasca, considerant la planificació inicial. A més, es considera que el projecte l'iniciaria un professional amb experiència prèvia en J2EE d'almenys tres anys.

Tasca	Temps per realitzar-la	Cost parcial (160 € / dia)
Definició de les principals funcionalitats del sistema així com la realització d'un estudi sobre la viabilitat tècnica i operativa per tal de dir si és factible realitzar-les mitjançant J2EE.	15 dies	2.400,00 €
Part corresponent a l'enginyeria del software i ha de començar a traduir totes les idees i conceptes definits a l'anterior part en l'arquitectura i estructura que finalment hauria de tenir el sistema.	15 dies	2.400,00 €
Implementació i validació del sistema seguint tots els esquemes i patrons definits en l'anterior etapa del projecte.	20 dies	3.200,00 €

Amb això, el cost total del projecte seria de: 8.000,00 €

6. Conclusions

La primera conclusió que es pot treure és que J2EE és un món molt ampli, en el que hi ha multitud de frameworks, eines i API's per a poder realitzar aplicacions web codificades en Java. També hi ha una àmplia oferta d'opcions a triar a l'hora de començar qualsevol projecte. Per exemple, es poden triar diversos tipus de servidors d'aplicacions; o diversos tipus de gestors de bases de dades; diversos tipus d'implementacions pel que fa a la vista de l'aplicació; etc.

Per tant, J2EE és una plataforma amb moltes possibilitats i a més, cadascuna d'elles està recolzada per una gran comunitat, de la qual, sempre en podràs trobar una solució a qualsevol requeriment funcional que es vulgui implementar.

Amb tot això, es pot dir que J2EE és un bon entorn pel qual apostar a l'hora de realitzar qualsevol projecte orientat al món web.

A nivell personal, el fet d'haver realitzat aquest projecte en aquest entorn, ha aportat un valor a nivell de coneixements que segur que podré aprofitar en el futur. A més, ha resultat molt satisfactori i gratificant el fet d'haver pogut realitzar una modesta xarxa social en J2EE.

7. Glossari

- **Xarxa social virtual** Interconnexió de nodes (usuaris) a través d'un espai virtual (web) i amb la possibilitat d'intercanviar informació entre ells.
- ***Street*** Àrea compartida per tots els usuaris de la xarxa. Tots els usuaris enregistrats poden escriure-hi perquè tots els altres llegeixin els seus comentaris.
- ***Servlet*** Aplicació Java que s'executa a dintre d'un servidor d'aplicacions i que es troba a l'espera de peticions.
- **Registre** Acció a través de la qual un usuari afegeix les seves dades a dintre de la BBDD del sistema. A partir d'aquest instant, l'usuari pot començа la seva vida social virtual.
- **Login** Acció a través de la qual, un usuari previament enregistrat, pot accedir al seu espai personal dintre de la xarxa i disfrutar de la vida social virtual.
- **Vincle** Connexió entre dos usuaris enregistrats a la xarxa.
- **Ruta de navegació** Possible transició, entre dues pàgines de l'aplicació, que es pot donar en el cas de que es produeixi un esdeveniment determinat.
- **Hibernate mapping** Traducció que realitza Hibernate, gràcies als fitxer XML *hibernate mapping* (*.hbm.xml), des del món de la BBDD cap al món Java i viceversa.

8. Bibliografia

A continuació es llista la bibliografia consultada per la realització del projecte:

- Tutorial: JSF (JavaServer Faces) 1.x, Apache MyFaces, & Facelets (*http://www.coreservlets.com/JSF-Tutorial/jsf1/*)
- Tutoriales :: Desarrollo de aplicaciones J2EE (JSF + Hibernate) *(http://www.tutelocreesnen.es/viewtopic.php?t=2184)*
- Chapter 3. JBoss Configuration (*http://docs.jboss.org/hibernate/core/3.3/reference/en/html/session-configuration.html*)
- JBossWithJSFCDDL | JBoss AS | JBoss Community (*http://community.jboss.org/wiki/JBossWithJSFCDDL*)
- Java EE 6 example application on JBoss 6 - Building the JSF 2.0 view (*http://www.mastertheboss.com/java-ee-16-articles/319-java-ee-6-example-project.html?start=5*)
- JSF Applications – JBoss Seam or Spring Web Flow? « Edem Morny's Tech Blog (*http://edemmorny.wordpress.com/2009/04/29/jsf-applications-%E2%80%93-jboss-seam-or-spring-web-flow/*)
- Ciclo de vida de la petición JSF *(http://www.proactiva-calidad.com/java/jsf/ciclo_vida.html)*
- JavaServer Faces Overview *(*http://www.oracle.com/technetwork/java/javaee/overview-140548.html*)*
- JSF (JavaServer Faces) Tutorial (*http://www.vogella.de/articles/JavaServerFaces/article.html#firstjsf_project*)
- Core JavaServer Faces (*http://horstmann.com/corejsf/*)
- IceFaces tutorial on JBoss (*http://www.mastertheboss.com/web-interfaces/201-icefaces-tutorial-on-jboss.html*)
- SetUpAMysqlDatasource | JBoss AS | JBoss Community (*http://community.jboss.org/wiki/SetUpAMysqlDatasource*)
- Hibernate *(http://jcsites.juniata.edu/faculty/thomas/Agile/hibernate.html)*
- Hibernate Tools y la generación de código (http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=hibernateTools)
- Hibernate 3.1, Colecciones, Fetch y Lazy *(http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=hibernateJoin)*
- How to Generate Hibernate Pojo Classes from DB Tables – wikiHow (*http://www.wikihow.com/Generate-Hibernate-Pojo-Classes-from-DB-Tables*)
- The Exadel Blog » Use Hibernate and JSF with Less Coding *(http://web.archive.org/web/20070406051317/http://blog.exadel.com/?p=8)*
- MySQL :: Connection pooling with MySQL Connector/J *(http://dev.mysql.com/tech-resources/articles/connection_pooling_with_connectorj.html)*
- Backing Bean Management *(http://download.oracle.com/javaee/1.4/tutorial/doc/JSFIntro8.html)*

- Servlets
  (http://javaweb.osmosislatina.com/curso/servlets.htm)
- Treball final de carrera (UOC)
  (http://materials.cv.uoc.edu/continguts/XW08_19018_00443/index.html?ajax=true)

9. Annex

En aquest apartat es veurà com instal·lar l'aplicació. L'aplicació ha estat desenvolupada i provada en un PC amb les següents característiques:

- 4 GB de RAM DDR2
- Intel Core 2 Duo T6400
- Open Suse 11.3

Instal·lació de *MySQL*

- Executar Yast2 com a superusuari.
- Buscar el paquet mysql i marcar el servidor per instal·lar.
- Fer click a *Acceptar.*

Construcció de la BBDD i creació de l'usuari uocbookuser

El servei *MySQL* ha d'estar executant-se. Per arrencar-lo, cal fer el següent com a superusuari: /etc/init.d/mysql start

Després, també com a superusuari, s'executa el client *mysql* des de la línia de comandes. I, una vegada allà, cal fer el següent*:*

- **Creació de la BBDD** create database uocbook;
- **Afegir l'únic usuari que accedirà a la BBDD** grant all privileges on uocbook.* to uocbookuser@localhost identified by 'uocbook'; (si el servidor de la BBDD estigués a una màquina diferent que la del servidor d'aplicacions, llavors, en comptes de *localhost* s'hauria de posar la IP de la màquina en qüestió).

- **Creació de la taula USER**
  create table USER (id INT NOT NULL AUTO_INCREMENT, nom VARCHAR(20), mail VARCHAR(20), password VARCHAR(20), imatge VARCHAR (50), PRIMARY KEY(id));

- **Creació de la taula STREETPOST**
  create table STREETPOST (id INT NOT NULL AUTO_INCREMENT, cmt TEXT NOT NULL, data DATETIME NOT NULL, id_usuari INT NOT NULL, PRIMARY KEY(id), FOREIGN KEY (id_usuari) REFERENCES USER(id) ON DELETE CASCADE);

- **Creació de la taula VINCLE**
  create table VINCLE (id INT NOT NULL AUTO_INCREMENT, id_usuari INT NOT NULL, id_usuari_vinculat INT NOT NULL, PRIMARY KEY(id), FOREIGN KEY (id_usuari) REFERENCES USER(id) ON DELETE CASCADE, FOREIGN KEY (id_usuari_vinculat) REFERENCES USER(id) ON DELETE CASCADE);

- **Creació de la taula MAIL**
  create table MAIL (id INT NOT NULL AUTO_INCREMENT, id_usuari_origen INT NOT NULL, id_usuari_desti INT NOT NULL, assumpte TEXT, missatge TEXT NOT NULL, status INT NOT NULL, PRIMARY KEY (id), FOREIGN KEY (id_usuari_origen) REFERENCES USER (id), FOREIGN KEY (id_usuari_desti) REFERENCES USER (id) ON DELETE CASCADE);

Instal·lació i preparació de JBoss

- Cal descarregar el paquest JBoss 6.0.0.Final des de la pàgina *http://www.jboss.org/jbossas/downloads/*
- Descomprimir el fitxer descarregat al directori */opt/*, per exemple.
- Configurar el data source de JBoss. Per fer-ho, s'ha d'editar el fixter mysql-ds.xml. Ha de tenir el següent aspecte:

```
<?xml version="1.0" encoding="UTF-8"?>

<datasources>
  <local-tx-datasource>
    <jndi-name>DefaultDS</jndi-name>
    <connection-url>jdbc:mysql://localhost:3306/uocbook</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>uocbookuser</user-name>
    <password>uocbookpswd</password>
    <exception-sorter-class-
name>org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorter</exception-sorter-class-
name>
    <metadata>
       <type-mapping>mySQL</type-mapping>
    </metadata>
  </local-tx-datasource>
</datasources>
```

- Al directori $HOME_JBOSS/common/lib s'ha de copiar el connector /j descarregat de la pàgina *http://dev.mysql.com/downloads/connector/j/*. Aquest fitxer *.jar* és el driver que li permetrà connectar-se a la BBDD.

Instal·lació de l'aplicació a JBoss

S'ha de desplegar el fitxer uocbook.war al directori *$HOME_JBOSS/server/default/deploy/*. Aquest fitxer *.war* ja conté les llibreries corresponents al framework *ICEFaces*.