Desenvolupament d'un projecte J2EE integrat amb *Android.*

***FrigoDroid*.** Sistema pel control d'estocs a l'àmbit familiar.

Ian Ramírez Fernández

ETIS

ALBERT GRAU PERISE

18/06/2012

1. Dedicatòria i agraïments.

A la Isa, principi i fi de la meva vida. Junts som imparables.

Al Marcel, un món fascinant t'espera allà fora.

2. Resum.

Aquest treball final de carrera es basa en l'anàlisi, disseny i implementació d'una aplicació pel control d'estocs. *Frigodroid* és una aplicació orientada a un ús per un frigorífic. Tot i que aquesta es pot enfocar a controlar qualsevol tipus de col·lecció d'objectes el punt fort per aquest projecte és que es basa en la practicitat de la lectura de codis de barres com a forma d'entrada dels ítems inventariats, això ens ajudarà en la identificació de l'objecte i en descartar possibles duplicitats en llistat de cada usuari. A més aquesta aplicació no necessita d'un codi de barres obligatòriament, si l'usuari ho decideix pot afegir objectes nous al inventari tot indicant la descripció del mateix.

El tret característic d'aquest projecte és la combinació entre una aplicació per *Android* on es realitzarà la part de l'escaneig, per afegir i/o retirar objectes a més de consultar el llistat , i un servidor web on cada usuari es podrà autenticar i administrar la seva informació com a usuari, és a dir: el seu inventari, paraula de pas i adreça electrònica. Per altra banda disposem d'usuaris de tipus administrador on podrem gestionar inventaris i usuaris via web.

Per la part web he escollit desenvolupar-la en J2EE amb suport de l'eina *Struts2* seguint un patró Model – Vista – Controlador. Per a la definició de les vistes també he fet servir el *framework Tiles* per definir les plantilles de presentació i ho hem enriquit amb *AJAX*, *Javascript* i *CSS*. Per la gestió de la base de dades ens hem decidit per *Hibernate*.

Per la part *d'Android*, l'aplicació ha estat desenvolupada en Java amb l'*Android SDK*, on també es serveix d'un model *MVC*. El patró *MVC* permet una ràpida localització de les classes que es volen desenvolupar o mantenir donat que aïlla cada funcionalitat per capes de presentació, càlculs i dades permetent un bon grau d'escalabilitat.

Finalment per la comunicació entre l'aplicació *Android* i el servidor hem fet servir JSON i GSON.

3. Índex de continguts

4. Cos de la memòria.

4.1. Introducció.

4.1.1. Justificació del TFC i context en el qual es desenvolupa.

La justificació d'aquest treball final de carrera parteix de dues motivacions personals.

La primera és que J2EE és la plataforma en la que es desenvolupen la majoria de clients web en el sector de serveis financers per als que tinc la sort de poder treballar des del món de la consultoria. Així doncs com a enginyer de programari especialitzat en COBOL és tota una oportunitat per poder adquirir nous coneixements que em donin valor afegit i poder obrir nous horitzons.

La segona motivació personal és la de poder aprendre i donar les primeres passes en el conjunt de tecnologies relatives a la mobilitat i específicament en el sistema operatiu per *smartphones Android*. Considero, personalment, que tant *iOS* com *Android* són el present i futur a nivell informàtic. Tecnològicament poden evolucionar sustancialment i considero que encara desconeixem tot el seu potencial que ha d'estar al servei de l'usuari convencional.

També vull tractar de demostrar l'assoliment dels coneixements adquirits durant la meva etapa universitària en la UOC com a estudiant d'enginyeria tècnica informàtica. Per això he plantejat un objectiu complex per tal de poder materialitzar-lo en un projecte real i tangible. Construïnt una solució escalable, útil i versàtil.

El punt de partida són els coneixements adquirits durant la carrera en diferents assignatures relacionades com poden ser, programació orientada a l'objecte, enginyeria del programari i bases de dades.

Primerament l'aportació d'aquest TFC és trencar amb la proposta de TFC on l'aplicació és només un web. Desenvolupant així una aplicació per *Android* que es comunica completament amb el servidor *Tomcat* on tenim les bases de dades i tota la lògica de l'aplicació. Actualment la forma d'interacció de l'usuari s'ha desplaçat als suports com tabletes, mòbils i ara televisors intel·ligents. Així doncs hem de saber integrar les nostres aplicacions, ja siguin en client o servidor a aquesta nova realitat.

Tot seguit una segona, i no menys important aportació d'aquest TFC, és la combinació dels *frameworks Tiles i Struts*, que ens aporta el patró model – vista – controlador, *Hibernate* emprat com a sistema de gestió de bases de dades. També hem pogut combinar diferents llenguatges de programació com *AJAX*, *Java*, *Javascript*, *HQL*, i *SQL*. Aportant riquesa al conjunt i tractant d'aprofitar les millors característiques d'aquests en cada moment.

La funcionalitat del projecte no és gaire complicada però si que està ben definida i finalment compleix el propòsit pel que va ser dissenyat. Això és degut al fet que partia d'un temps limitat i que el plantejament inicial ens oferia moltes possibilitats i corria el risc de tractar d'establir objectius inabastables. Hom que tingui experiència en projectes informàtics sabrà que moltes vegades aquesta és una fita ben difícil.

4.1.2. Objectius del TFC.

Els objectius del TFC són dos. Principalment vull aprendre de la vessant de *J2EE* i entendre d'una manera sòlida el sistema model – vista – controlador aplicat a *J2EE*, aquest objectiu és principal perquè encara que he tingut la possibilitat de participar en petits desenvolupaments que no he pogut arribar a entendre els seus principis i funcionaments. També és força important poder aprendre *Hibernate* donat que és un sistema de gestió de bases de dades bastant demanat en el mercat laboral i només en tinc coneixements de *SQL*. Com a última referència per aquest punt vull remarcar que també és força interessant i m'ho marco com a objectiu, aprendre algunes nocions bàsiques d'*AJAX* donat que és un llenguatge força nou que permet acabats força bonics.

En quant a la part d'*Android* l'únic objectiu que em marco és aprendre a fer una aplicació bàsica, doncs no és l'objectiu principal del TFC tot i que definirà força la part d'ús de l'aplicació. La part de comunicació entre servidor i *Android* és força important i per això haurem de dedicar una bona part de l'aprenentatge.

La corba d'aprenentatge doncs, es pot apreciar com a força pronunciada, ja que parteixo amb nocions bàsiques dels petits desenvolupaments Java que he pogut fer i hi ha una varietat de llenguatges i tecnologies que en desconec totalment.

4.1.3. Enfocament i mètode seguit.

Degut a la gran quantitat de conceptes nous i l'arquitectura complexa del projecte, hem tingut dos punts essencials de cara a l'enfocament que els he hagut de solapar com vaig suggerir al pla de treball per a poder arribar a temps a l'entrega.

L'anàlisi i disseny de l'aplicació han marcat una fase ben diferenciada de la fase de construcció i proves.

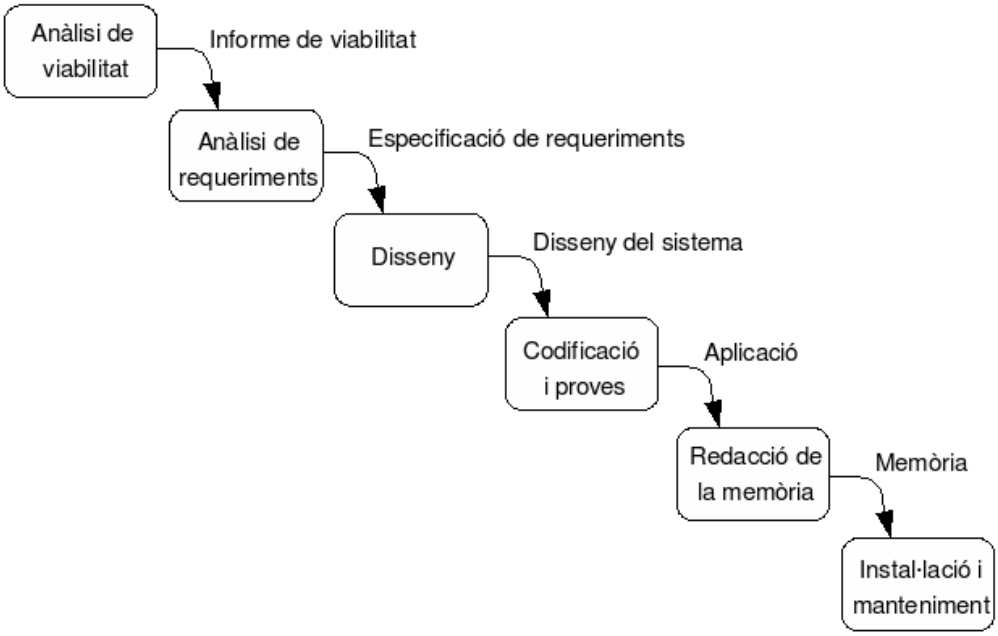
Per tal d'afrontar la corba d'aprenentatge hem tingut una fase d'investigació a cavall entre la fase de disseny i la fase d'implementació, on hem hagut de consultar forces manuals d'Internet on s'exposen casos pràctics de com poder implementar allò que en tot moment hem necessitat.

M'he hagut de centrar en allò que m'ha resultat més dificultós, fet que ha causat la majoria de retards sobre el pla de treball establert durant tot el procés que ha sigut *Hibernate* seguit *d'Struts.*

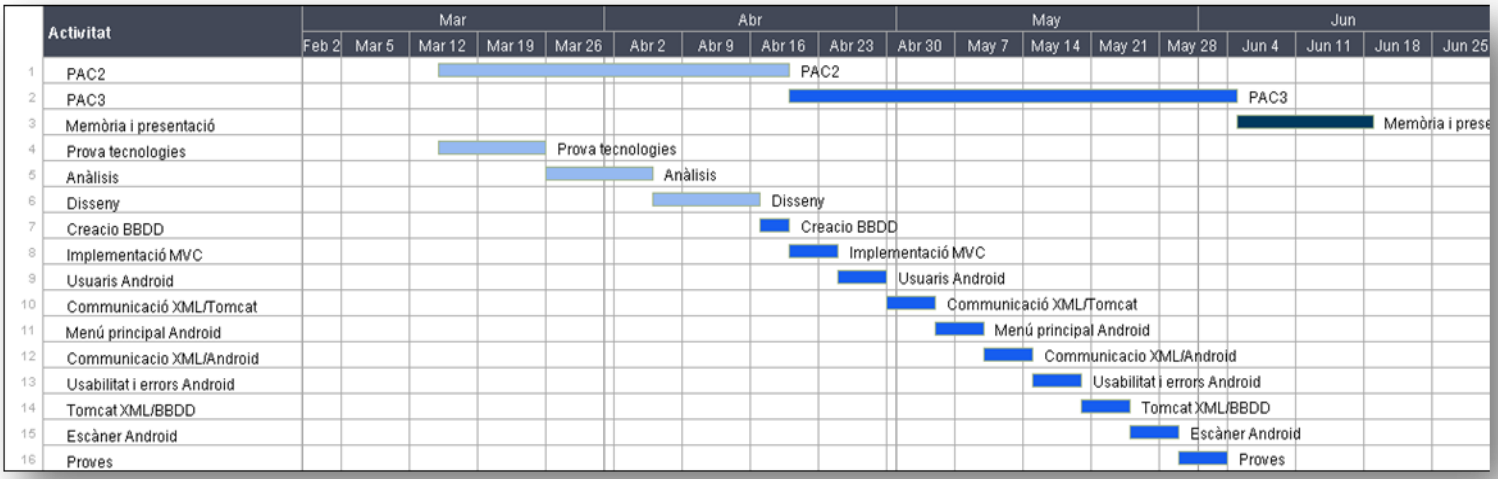
Finalment a l'etapa de construcció, amb una feina d'investigació prèvia he hagut d'encabir les diferents parts que tenia fetes i personalitzar-les per tal de complir amb les funcionalitats dissenyades i així és com hem pogut beneficiar-nos del temps invertit a la investigació a l'etapa de disseny i primeres setmanes de desenvolupament.

4.1.4. Planificació del projecte.

La planificació del projecte s'ha establert tenint en compte el cicle de vida d'un projecte informàtic seguint el model següent:



Tot i que han hagut alguns desajustaments, he intentat seguir la planificació establerta al pla de treball inicial del qual s'adjunta una imatge:



El treball s'ha repartit durant tota la setmana, una mitjana de tres hores, tot i que els caps de setmana hem hagut d'invertir bastants més hores de les programades.

Resten incomplertes les següents funcionalitats de l'entrega final:

- *Android*: Una millor gestió del sistema d'escaneig. La sensació que dona és que no està gaire controlat, donat que salta cada vegada que l'operació de modificar ha estat correcta. Potser, hauríem de deixar sota control de l'usuari si vol obrir l'escàner.

- *Web*: Gestió de sessions correcta. Ara mateix estem gestionant les sessions via interceptors. Queda per gestionar la destrucció de les sessions de forma correcta.

- *Web*: Gestió de paginació i refresc de les taules d'informació. Algunes taules no refresquen o no arriben a paginar correctament. Es tracta d'una qüestió de definició de les taules del *plugin* de *jquery* a les pàgines JSP.

- *Web* + *Android*: Validació dades en formularis. Ara per ara no estem validant el 100% de tots els camps raó per la qual, l'usuari pot estar introduïnt dades corruptes. S'ha establert a l'arquitectura els medis necessaris per corregir aquesta situació, només quedaria fer-ho extensible a tots els camps validables.

4.1.5. Productes obtinguts.

Els productes obtinguts són dos:

- L'aplicació ***Frigodroid*** per *Android*.
- L'aplicació web per a gestionar les dades obtingudes mitjançant l'aplicació ***Frigodroid***.

  Una aplicació web programada en l'especificació J2EE de Java amb capa de presentació en JSP.

- El fitxer per importar la base de dades en SQL amb dades inicials per poder fer ús del sistema.
- Manual d'usuari per ambdues aplicacions.
- El codi font d'ambdós projectes.
- Per a la part web el *WAR* per poder aixecar un servidor *Tomcat*. No entreguem la versió compilada d'android donat que necessitarem modificar la IP on tenim el servidor, està explicat al manual d'usuaris.
- *JavaDoc* de totes les classes per ambdós projectes.

4.1.6. Breu descripció dels altres capítols de la memòria.

A continuació tenim el cos de la memòria.

Està dividida en diferents parts, primerament tenim l'especificació i l'anàlisi dels requeriments, on fem una descripció en profunditat de com haurà d'actuar la funcionalitat i què s'espera d'aquesta. Val com a primera presa de requeriments. Aquesta està dividida en una petita introducció, la descripció pròpiament i el disseny de les interfícies.

A continuació tenim el disseny tècnic de l'aplicació on expliquem l'arquitectura emprada tant en el servidor web com en l'aplicació *Android*. També s'inclouen diferents diagrames que van ser construïts durant la fase de disseny i presentats en la PAC2.

Finalment trobem la valoració econòmica, conclusions, glossari i Manuals.

4.2. Especificació i anàlisi dels requeriments.

4.2.1. Introducció.

Aquest capítol reflexa els requisits del sistema a implementar definint detalladament les necessitats del programari per tal de poder resoldre els problemes proposats. Això inclou diferents diagrames d'estat, casos d'ús, activitats i seqüències i es complementa amb un diagrama UML per a facilitar una posterior implementació.

4.2.2. Requisits funcionals.

Aquest projecte es tracta d'una aplicació que una vegada acabada tota la fase de proves podrà ser penjada a *Google Play* (és el nou *Android Market)* per part d'una empresa nínxol basada en un una estratègia *B2B2C.* Es dedica a la creació de programari pel consumidor directe i per negocis.

En aquest cas, es tracta del primer de dos projectes que tracta de fer-se un lloc en usuaris de telèfons intel·ligents que tinguin la necessitat de dur un control sobre l'estoc de les existències a la seva llar. Per aquest projecte amb un model basat en publicitat per poder capitalitzar aquest projecte.

Aplicació *Android*:

Una vegada l'usuari es baixa l'aplicació, i se la instal·la, s'ha d'enregistrar tot i que la pantalla inicial és per s'autentiqui. El sistema d'enregistrament d'usuaris per part de l'aplicació *FrigoDroid* és bastant simple, no demana més que un nom d'usuari, una adreça de correu electrònic per mantenir el contacte amb l'usuari, la paraula de pas per poder accedir al sistema i la periodicitat amb la que es voldrà rebre un recordatori via adreça electrònica de les existències. L'empresa ha decidit no demanar més que les dades bàsiques donat que el que prima és un ràpid accés al sistema de l'usuari, ja que ens interessa un gran nombre d'usuaris enregistrats i actius en el sistema doncs això serà la força de venda de cara al segon projecte.

Tot i que les aplicacions entregades només són en català, el llenguatge definit per part de l'aplicació *FrigoDroid* dependrà de la configuració local del telèfon mòbil donat que *Android* només requereix de les traduccions per poder presentar-les en l'idioma del telèfon.

Una vegada l'usuari introdueix les dades de registre, si el registre ha estat satisfactori l'usuari ha de poder autenticar-s'hi a la pantalla inicial amb el codi d'usuari i la contrasenya escollides. Aquesta de primera vegada vindrà en blancs per motius de seguretat però una vegada el procés d'autenticació sigui correcte l'aplicació els recordarà per reduïr el temps que triga l'usuari en accedir perquè només haurà de prémer el botó per poder-se autenticar. Tot i que l'usuari pot escollir diferents codis d'usuari per poder autenticar-se *FrigoDroid* només recordarà l'últim executat amb èxit.

Una vegada autenticat l'usuari accedeix a la pantalla principal on disposa de tres opcions, però primer haurà d'escollir el mètode per entrar els objectes. Aquest pot ser manual o escàner, per defecte serà manual.

Afegir i esborrar: Manual: L'usuari podrà introduir la descripció de l'objecte i la quantitat d'objectes. Acte seguit podrà prémer el botó d'afegir o esborrar.

Escàner: Una vegada l'usuari canvia el valor del desplegable a escàner apareixerà la pantalla per escanejar ítems. Si el procés finalitza satisfactòriament l'usuari comptarà amb una descripció proposta pel sistema tot i que podrà modificar-la.

Llistar: Si l'usuari selecciona llistar l'aplicació retornarà una llista d'objectes que pertanyen al l'usuari. Amb una icona verda vol dir que encara tenim en estoc amb una icona verda guixada

en vermell vol dir que l'estoc es més petit que zero. Si l'usuari selecciona qualsevol ítem del llistat l'aplicació informarà del nombre exacte d'existències.

Escàner:

Per poder implementar el lector de codis de barres hem escollit les llibreries de *ZXING*. Aquest és un projecte de codi obert que ha estat programat en *Java* però que serveix per a diferents llenguatges de programació.

El fet de fer servir aquestes llibreries ens obliguen a publicar una menció al projecte ZXING, que està penjat a

http://code.google.com/p/zxing

i els termes de la llicència en la que està publicada, que es pot consultar aquí: http://www.apache.org/licenses/LICENSE-2.0.html

Tot i que aquestes llibreries permeten la configuració de la crida de l'escàner en mode massiu hem escollit l'escàner de forma selectiva per fer que l'usuari esculli si vol escanejar cada objecte. Aquestes llibreries també permeten llegir infinitat de codis de barres tot i que nosaltres ens centrarem en els codis de barres de productes (genèricament *EAN13*).

Una vegada escanegi el producte en qüestió aquest farà una consulta al servidor i retornarà una descripció si anteriorment ha estat escanejat aquest codi de barres, en canvi, retornarà el codi de barres a la caixa de text de la descripció si no troba cap coincidència en el servidor.

Web:

Tenim dos opcions segons el tipus d'usuari. Tot usuari accedirà al sistema de gestió web per una mateixa finestra per autenticar-se.

- Si l'usuari és administrador:

  Si l'usuari és administrador podrà fer la gestió sobre tots els usuaris. Sobre cada usuari podrà gestionar el seu inventari, modificant la quantitat de cada ítem relatiu a l'usuari en qüestió. L'administrador no podrà afegir ítems nous.

Obtindrà una taula amb tots els usuaris existents a la base de dades. Cada fila es correspondrà a un usuari on trobarem: la contrasenya, l'adreça electrònica de contacte, si és administrador, si és actiu.

Per poder filtrar en aquesta taula l'usuari disposa d'una caixa de text on podrà seleccionar el codi d'usuari desitjat. Finalment per completar el filtrat l'administrador pot prémer el botó per buscar i així obtindrà totes les coincidències segons el criteri amb el que decideixi buscar a la taula esmenada anteriorment.

Una vegada selecciona un usuari fent clic sobre el codi d'usuari la informació relativa a l'usuari seleccionat es carrega deixant com a modificable els camps referents a la quantitat dels ítems, si l'usuari és administrador, si l'usuari és actiu i si la periodicitat del recordatori.

Una vegada l'usuari modifica qualsevol cosa podrà prémer el botó per actualitzar la informació en la base de dades.

- Si l'usuari és client:

Si l'usuari és client podrà gestionar les existències al seu inventari, així com de la seva informació personal que conté el sistema. Aquesta informació personal es refereix als components de l'adreça de contacte, la paraula de pas, la freqüència del recordatori i la quantitat dels ítems. En aquest cas l'usuari tampoc podrà afegir ítems nous, és a dir, ja hauran d'existir.

Una vegada actualitzada la informació, l'usuari podrà prémer el botó d'actualitzar, aquesta informació serà actualitzada a la base de dades.

L'usuari també podrà esborrar el seu compte. Aquest no desapareixerà de la base de dades, sinó que es marcarà com inactiu sent-hi un usuari administrador l'únic que pugi reactivar el compte.

Pel cas de l'aplicatiu web compta amb el sistema i18n semblant al que es fa servir a *Android* on només ens caldria definir diferents fitxers amb les traduccions. Tot i que en aquest lliurable només comptem en l'edició en català.

Comunicació *Android* / Servidor *Tomcat*:

Durant el procés d'ús per part de l'usuari de l'aplicació *Android* aquesta haurà de comunicar-se amb el servidor *Tomcat* per tal de poder validar, guardar, o presentar informació. Per aquest motiu establim els moments essencials, amb el tràfic mínim i imprescindible, per tal d'evitar la sobrecarrega del servidor o una errada en la gestió de la informació, que pot provocar registres corruptes i/o indesitjats. En tot moment serà competència de l'aplicació *android,* interpretar qualsevol error retornat a les respostes del servidor a les peticions.

Els moments en els que l'aplicació *Android* i el servidor *Tomcat* tenen comunicació són els següents:

- Autenticació usuari: L'aplicació envia un codi d'usuari i una paraula de pas en text pla. *Tomcat* ha de retornar un booleà a cert si el procés d'autenticació ha estat el correcte. I si és fals, en cas contrari.

- Registre usuari: L'aplicació, una vegada l'usuari fa clic al botó de registrar si no hi ha errors de validació al formulari de registre, enviarà tota la informació al servidor. *Tomcat* com en el cas d'autenticació rebrà un booleà a cert, si el procés d'enregistrament ha sigut correcte i fals, en el cas contrari.

- Llistar: Cada vegada que l'usuari prem en el botó de llistar de la pantalla principal, l'aplicació envia una petició de llistar informant el codi d'usuari per la que vol el llistat,. El servidor retornarà tot el inventari en aquell moment que es trobi associat a l'usuari per tal que l'aplicació la pugui llistar degudament.

- Afegir/Esborrar: Cada vegada que l'usuari faci clic en afegir esborrar s'enviarà una petició de modificació de les existències amb el codi d'usuari, el signe de l'operació, l'ítem i la quantitat a restar o sumar. *Tomcat* com en el cas d'autenticació rebrà un booleà a cert si el procés de modificació ha sigut efectuat correctament i fals en el cas contrari.

- Escaneig correcte: En cada escaneig l'aplicació enviarà els dígits resultants de l'escaneig i *Tomcat* retornarà la descripció si troba correspondència a la taula d'ítems, espais en cas contrari. Acte seguit l'aplicació deixa visible aquesta descripció a la caixa de text de la descripció de l'ítem com a descripció suggerida.

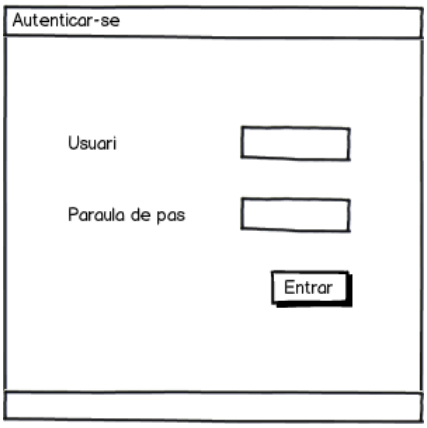
4.2.3. Interfícies.

- *Android*:

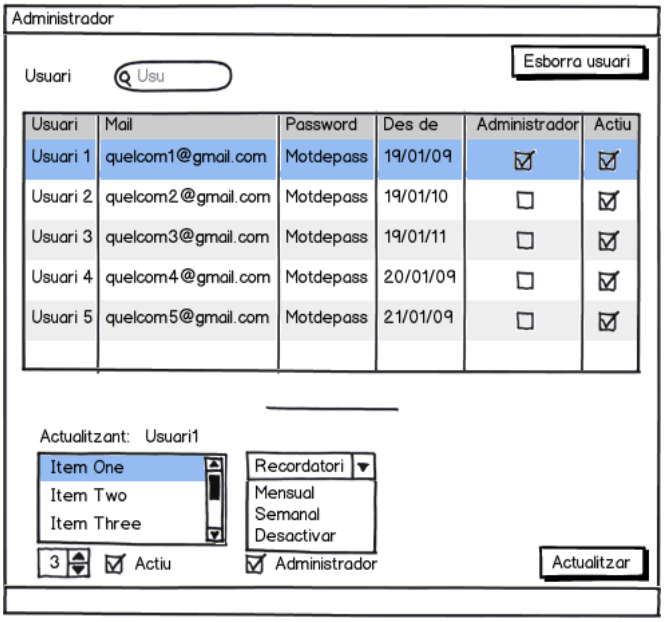
  - Registre.
  - Autenticació.
  - Pantalla principal per introducció/esborrar manual.
  - Pantalla principal si l'usuari demana un llistat.
  - Introducció esborrar mitjançant escàner de codi de barres.

- Web

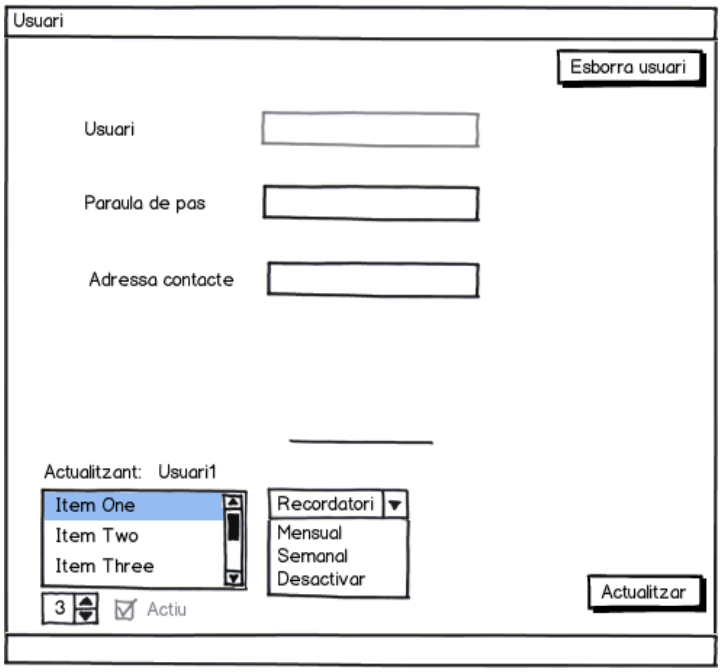
  - Finestra autenticació.



- Finestra administrador.



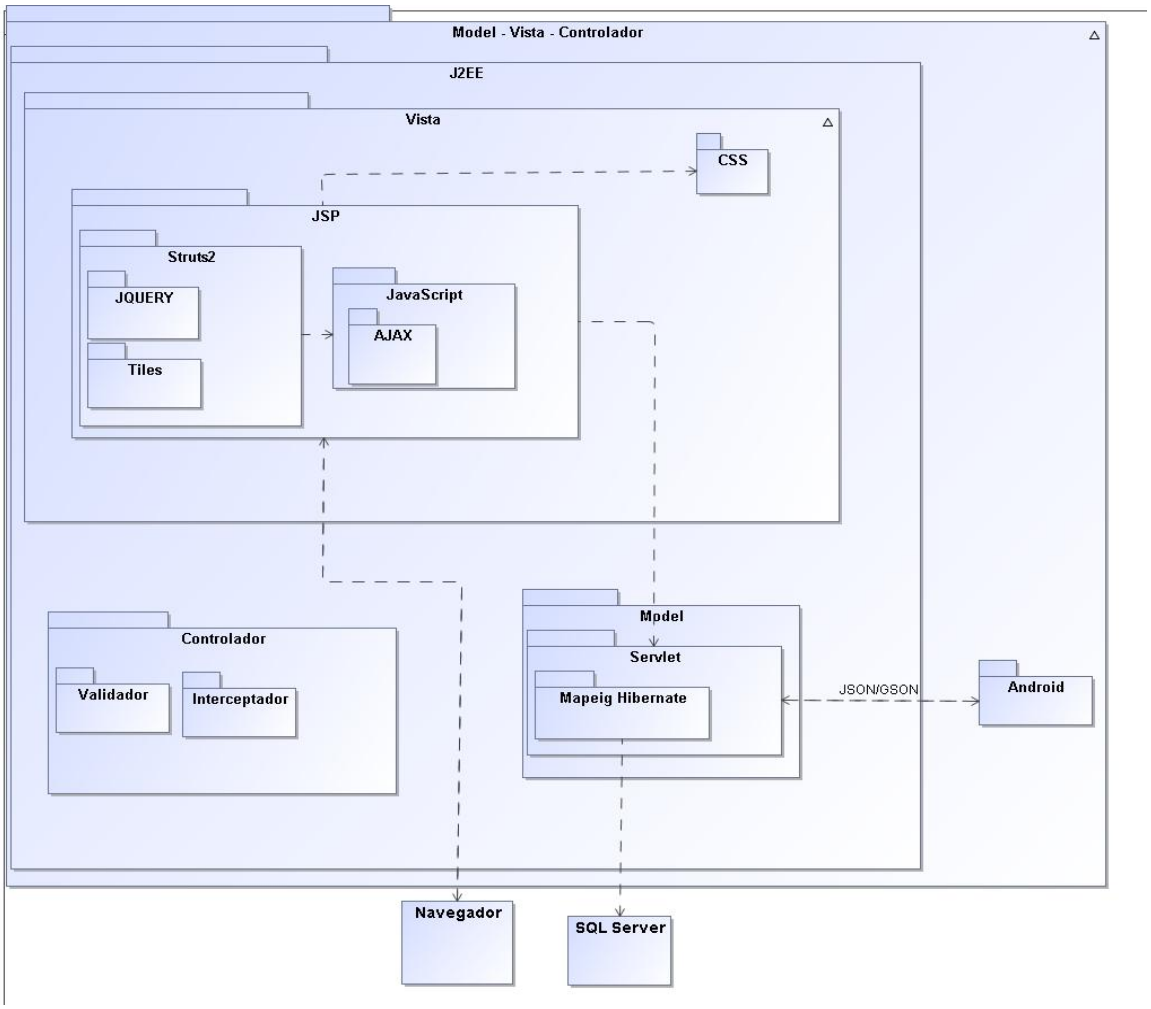
- Finestra usuari.



4.3. Disseny tècnic.

4.3.1. Arquitectura

El projecte està emmarcat dintre de la tecnologia J2EE, la solució *Java* modular amb arquitectura de *n* capes amb especificacions pròpies destinada a la programació d'aplicacions web, però J2EE és només la tecnologia mitjancera amb la que podem desenvolupar part de tot el projecte. Ara passem a explicar detingudament totes les parts que conformen l'arquitectura del projecte i com l'hem integrat en el nostre projecte.



El patró MVC és un patró d'arquitectura de programari que separa les dades d'una aplicació, la interfície de l'usuari i la lògica de negoci en tres components diferents. Com es pot apreciar aquest patró es idoni per desenvolupaments web i en tenir les diferents capes separades propicia un fàcil manteniment a priori.

4.3.1.1. Model.

La capa model de l'arquitectura MVC representa totes les classes relacionades amb la gestió de la base de dades i també de la lògica de negoci. Això és que en aquesta capa treballarem la els fluxos d'informació i de navegació definides per al propòsit del projecte facilitant la seva representació i donant forma al sistema que quan vam establir el disseny del projecte es va modelar.

4.3.1.2. Controlador.

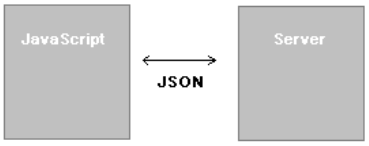
La capa controladora representa al sistema de navegació en el projecte responent a la interacció amb l'usuari i amb els esdeveniments externs al sistema que requereixin d'una resposta per part del nostre sistema. No tracten pròpiament la lògica de negoci però són essencials per al correcte funcionament de l'aplicació. En aquest paquet podem incloure la gestió de sessions, la validació de *JSP* i formularis mitjançant els interceptors.

4.3.1.3. Vista

Fa referència a tot allò que ens ajuda a interactuar amb l'usuari. *JSP, AJAX, JavaScript, CSS*. Aquesta capa representa el disseny i presentació de la informació per l'usuari.

4.3.1.3.1. AJAX.

AJAX, acrònim de Asíncron *Javascript* and XML. En aquest cas força part de *l'AJAX* emprat en el nostre projecte fa servir *JSON* per tant no està formatat en XML. *AJAX* proveeix de dinamisme les pàgines web encara que pot arribar a sobrecarregar el servidor si s'abusa.



La forma de funcionar *d'AJAX* és conceptualment molt senzilla. La interacció de l'usuari genera que segons la funcionalitat AJAX que haguem definit es cridi una acció en el servidor, aquest la processi i ens retorni els valors desitjats, després la funcionalitat AJAX la presenta en forma de caixa de text que s'autocompleta, de taula que s'actualitza,..

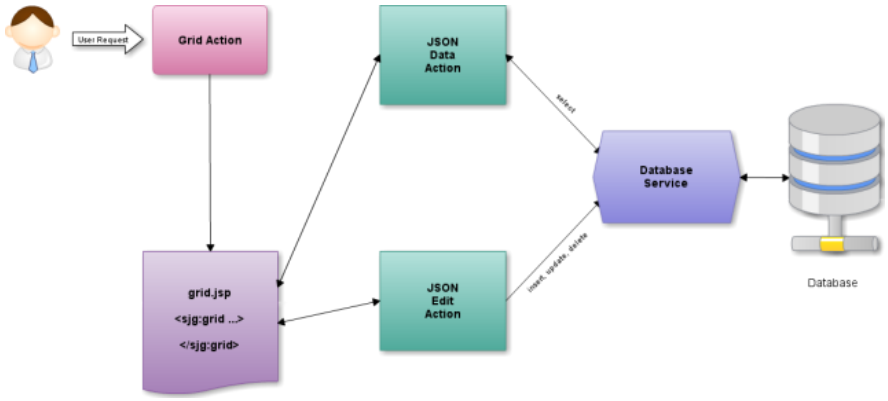
JavaScript és el llenguatge en el que s'efectuen les funcions de crida d'*AJAX* mentre que l'accés a les dades fa servir *XMLHttpRequest*.

Com exemple *d'AJAX* al nostre projecte tenim tots els *widgets* emprats donat que *JQuery* ve a ser una llibreria de funcionalitats *AJAX*.

4.3.1.3.2. JQuery

Inicialment, en l'entrega de la PAC3 no comptàvem amb JQuery donat a que teníem moltes deficiències en la seva implementació. Una vegada resoltes hem substituït els *tags* de *Struts* pels *tags* de *JQuery, versió 2.3.1,* per la presentació en els *JSP* de les taules d'informació pel producte final. Hem fet servir un *plugin* de *JQuery* per a *Struts2* aquest proveeix funcionalitat *Ajax* i ens dona *widgets* per enriquir la interfície d'usuari. En aquest cas hem fet ús de dos *widgets:*

DataGrid, per representar taules d'informació i dels Acordions per tal de poder agrupar panells segons la seva funcionalitat. Aquesta representa la informació que rep d'objectes *JSON* i els transforma en els elements que definim a la taula.



La taula de *JQuery* ens dona la facilitat que permet establir un altre *action* per poder modificar la taula. Per exemple en la part web, *FrigoDroidServe*r, tenim una *JSP* per poder administrar els usuaris. Aquests usuaris ens venen un llistat via *JSON* en la primera *action* definida just quan la *JSP* es carrega per primera vegada. Una vegada carregada i presentada a l'usuari aquest pot modificar el que desitgi i s'acciona l'*action* definida en el fitxer *Struts*. Aquesta acció s'encarrega d'arribar a la classe on efectuarem l'acció dissenyada en aquell punt, en aquest cas cridar els DAO d'Hibernate per poder actualitzar l'objecte usuari modificat.

Exemple d'implementació en el fitxer *struts* de les dues accions:

```
<action name="updateUsuari class="frigodroid.server.AdminView2" method="updateUsuari">
      <result name="success" type="json"/>
</action>
```

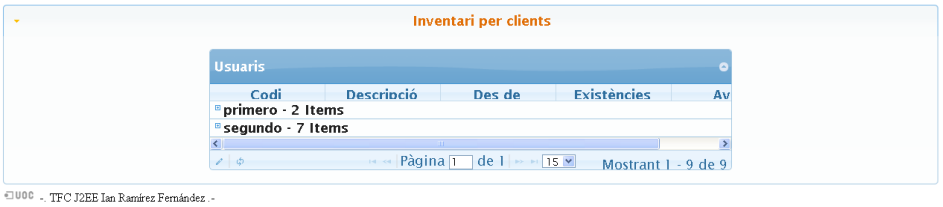
```
<action name="usuarisTable" class="frigodroid.server.AdminView2" method="getJSON">
      <result name="success" type="json"/>
</action>
```

En la següent imatge es pot veure el resultat de la taula prèviament descrita per a l'administració d'usuaris:

Un altre widget que hem emprat ha sigut l'acordió. Aquest ens permet agrupar informació segons la seva funcionalitat i ens ajuda a endreçar la vista que obté l'usuari. Aquest cas són completament estàtiques però el *plugin* de *JQuery* ens permet fer-les completament dinàmiques. Aquests acordions permeten bons resultats amb poques línies de codi, realment són interessants si volem millorar notablement la experiència per l'usuari.

En aquesta imatge tenim un acordió sense desplegar:

I ara desplegat:



4.3.1.3.3. JavaScript.

És un llenguatge de programació, orientat a objectes, basat en prototips (les classes a les que fem referència no són presents) i que s'executa en la part del client.

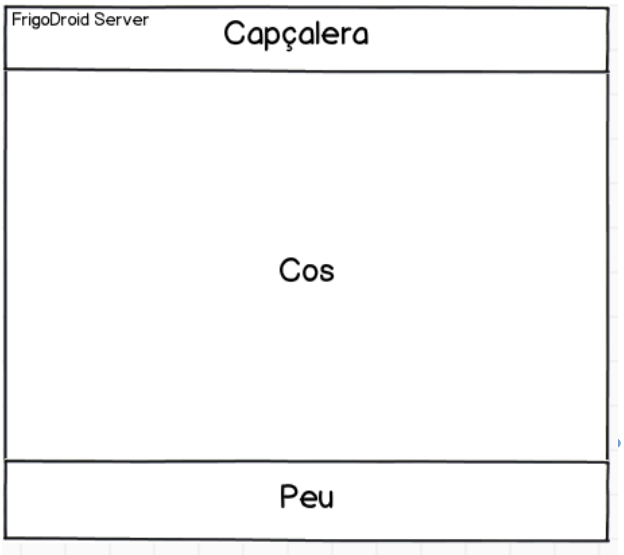
És la base d'*AJAX* i de *Struts* donat que les seves definicions durant el desenvolupament després en la part del navegador es despleguen majoritàriament en forma de codi *javascript*.

Aquí podem veure l'exemple anterior des de la part del navegador. On podem veure que es genera *JavaScript:*

```
<script type='text/javascript'>
jQuery(document).ready(function () {
    jQuery.struts2_jquery.require("js/struts2/jquery.gri
    var options_gridedittable = {};
    options_gridedittable.frozen = false;
    var options_gridedittable_colmodels = new Array();
    var options_gridedittable_colnames = new Array();


options_gridedittable_colmodels_codiUsuari = {};
options_gridedittable_colmodels_codiUsuari.name = "codiU
options_gridedittable_colmodels_codiUsuari.jsonmap = "co
options_gridedittable_colmodels_codiUsuari.index = "id";
options_gridedittable_colmodels_codiUsuari.editable = fa
options_gridedittable_colmodels_codiUsuari.sortable = tr
options_gridedittable_colmodels_codiUsuari.resizable = t
options_gridedittable_colmodels_codiUsuari.search = true
options_gridedittable_colnames.push("Codi Usuari");
options_gridedittable_colmodels.push(options_gridedittab


options_gridedittable_colmodels_mail = {};
options_gridedittable_colmodels_mail.name = "mail";
options_gridedittable_colmodels_mail.jsonmap = "mail";
options_gridedittable_colmodels_mail.index = "mail";
options_gridedittable_colmodels_mail.editable = true;
options_gridedittable_colmodels_mail.sortable = false;
options_gridedittable_colmodels_mail.resizable = true;
options_gridedittable_colmodels_mail.search = false;
options_gridedittable_colnames.push("mail");
options_gridedittable_colmodels.push(options_gridedittab


options_gridedittable_colmodels_password = {};
options_gridedittable_colmodels_password.name = "passwor
options_gridedittable_colmodels_password.jsonmap = "pass
options_gridedittable_colmodels_password.index = "passwo
```

4.3.1.3.4. Tiles

Tiles, versió 2.0.6, ens serveix per definir la plantilla de la nostra aplicació. Ens ajuda a definir els fragments que tindrà la nostra aplicació. Aquests *tiles*, els fragments definits, es poden incloure dins d'altres peces per tal de reduïr la duplicitat de codi. Permet establir consistència en el desenvolupament de la capa de presentació.



Els *tiles* definits, partint d'una base on tenim tres *tiles* essencials i les seves proporcions, capçalera(20%), cos(80%) i peu(20%). No hem necessitat més subdivisions doncs hem tractat de simplificar el disseny i

eliminar botons, menús i *links* que poguessin resultar superflus.

4.3.1.3.5. Struts

És una eina, versió 2.3.1, de suport pel desenvolupament, sota llicència *Apache*, de les aplicacions web sota el patró MVC a la plataforma *Java Enterprise Edition*. Permet reduir temps de desenvolupament, fem servir la versió dos de *Struts* amb algunes millores per tal de simplificar el desenvolupament com per exemple la integració *d'AJAX.*

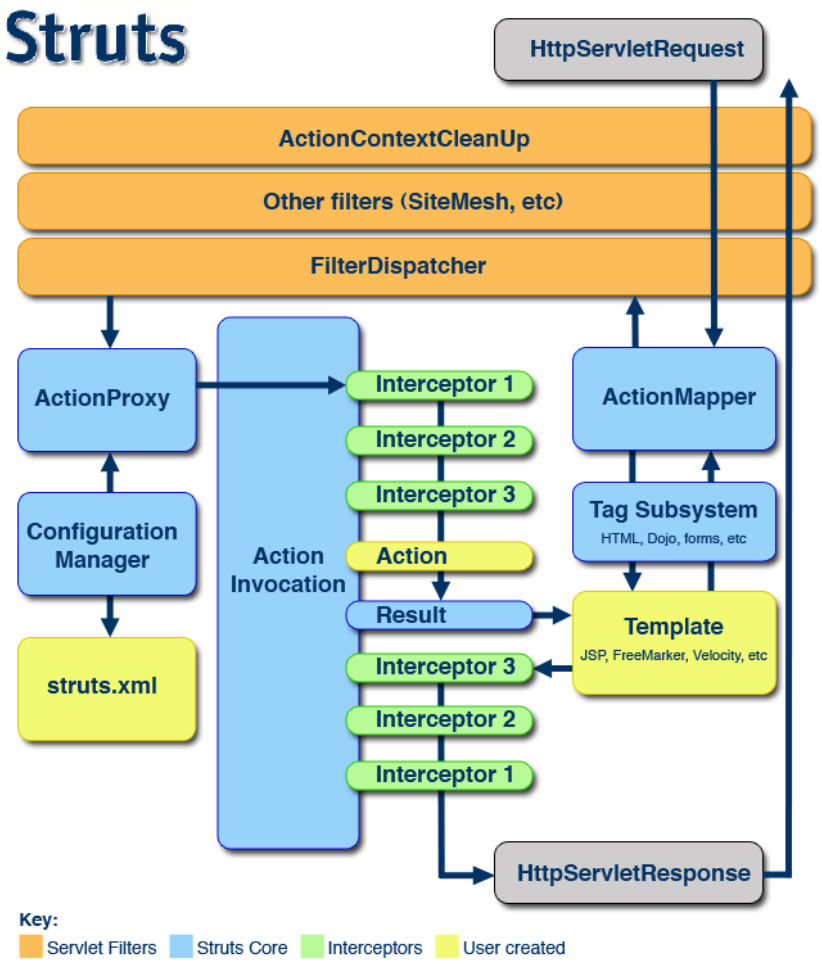
Struts proporciona tres components claus:

- Un gestor de sol·licituds proporcionat pel desenvolupament de l'aplicació mapejat a la URL estàndard.

- Un gestor de respostes que transfereix el control a altres recursos per a completar la resposta.

- Una llibreria de *Tags.* Aquesta ajuda als desenvolupadors a crear aplicacions basades en formularis amb pàgines de servidor (*JSP* en aquest cas).

En la següent figura podem apreciar el funcionament bàsic de *Struts* i a seva arquitectura:



Il·lustració - struts.apache.org

En el nostre cas, podem explicar la implementació de *Struts2* mitjançant el cas per la finestra d'autenticació de l'usuari:

En la pàgina *Login.jsp* definim un *tag* formulari de *Struts*.

```
<s:form action="login" method="post">
        <s:textfield name="username" key="label.username" size="20" />
        <s:password name="password" key="label.password" size="20" />
        <s:submit method="execute" key="label.login" align="center" />
</s:form>
```

Com podem veure estem fen referència a quina acció haurà d'anar a buscar la classe controladora per poder continuar el flux de navegació segons s'hagi establert en *Struts.xml*.

Definim una acció al fitxer de configuració de *struts:*

```
<action name="login" class="frigodroid.server.LoginAction">
      <interceptor-ref name="loggingStack"></interceptor-ref>
     <result name="admin" type="redirectAction">admin</result>
     <result name="client" type="redirectAction">client</result>
     <result name="error">Login.jsp</result>
</action>
```

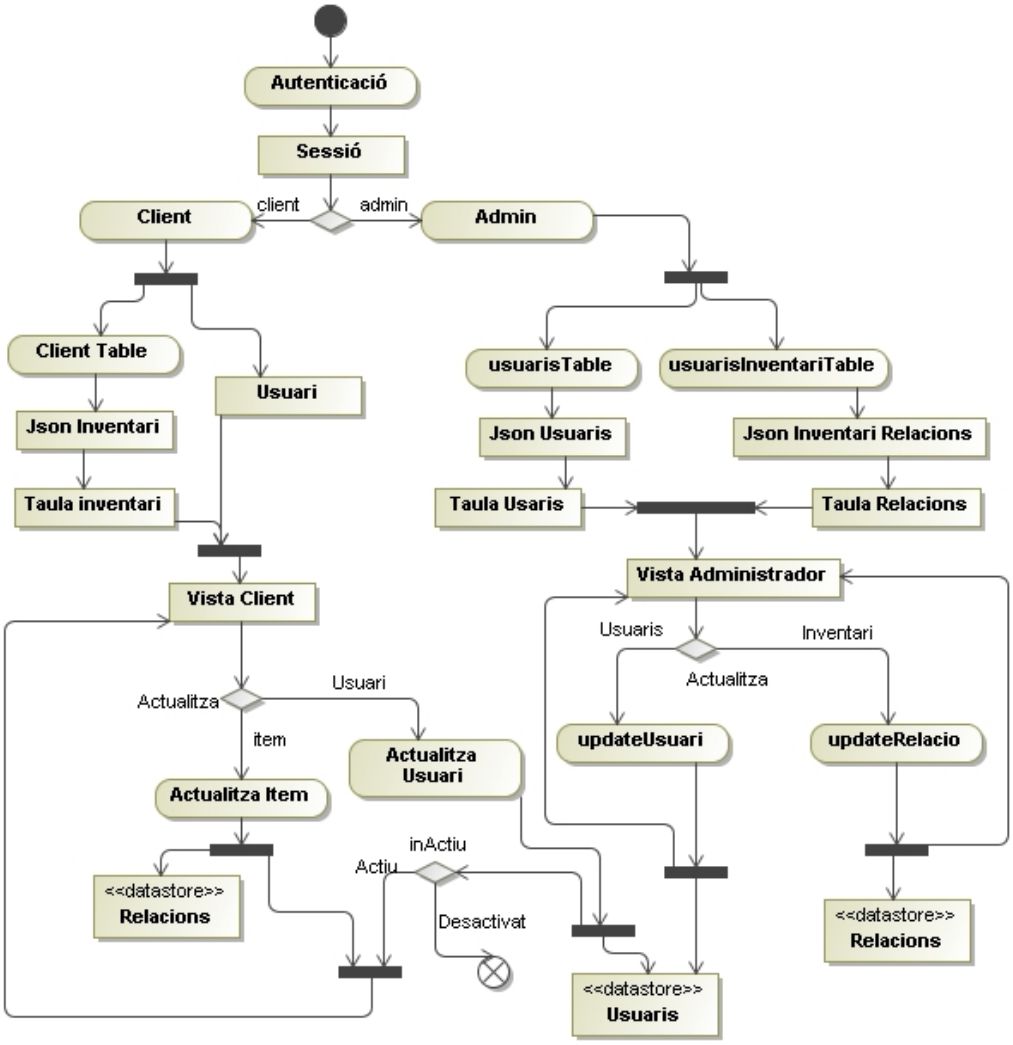
El procés definit diu que primerament s'executarà la classe controladora a la que fa referència *loggingStack* que hem definit prèviament com interceptor per defecte en el fitxer *Struts.xml* :

```
<interceptors>
       <interceptor name="mylogging"
       class="frigodroid.server.interceptor.MyLoggingInterceptor">
       </interceptor>
            <interceptor-stack name="loggingStack">
                <interceptor-ref name="mylogging" />
                <interceptor-ref name="defaultStack" />
            </interceptor-stack>
</interceptors>
```

Dins la classe *MyLoggingInterceptor* hem definit la lògica per al control de la sessió. En cas que tot vagi bé, deixarà continuar amb l'execució del flux de navegació. En cas contrari llençarà una excepció al sistema. Aquesta classe s'executa sempre que l'haguem definit en *Struts.xml*. Això actua com a sistema de seguretat donat que sinó hom que sabés la configuració de la URL correcta podria accedir a la informació de les bases de dades. Tot i que la implementació dels sistemes de seguretat en el projecte són molt bàsics i de cara a un projecte real l'arquitectura implementada permet reforçar aquest punt fàcilment.

Reprenent el flux de navegació establert en l'acció tenim dos casos. Que la classe acció pel la vista de *login* ens retorni les cadenes "*admin*" o "*client*" i en conseqüència dirigim a l'usuari segons el seu perfil.

En la següent il·lustració tenim definida el diagrama d'activitats pel flux de navegació que tenim definida a *Struts*. Com es podrà apreciar ha quedat força més detallada que en la fase de disseny, això és donat que durant el desenvolupament hem hagut de tenir en compte els requeriments tècnics de l'arquitectura.



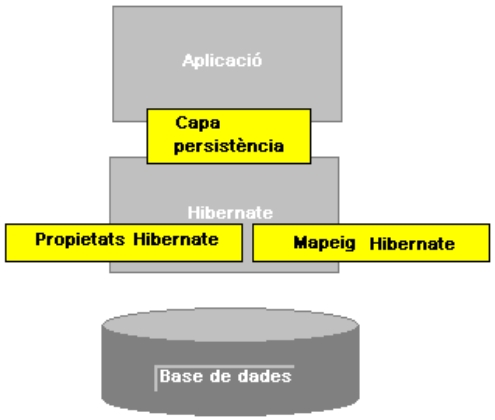
4.3.1.4. *Hibernate*.

Hibernate, versió 3,2 ,es una eina per l'assignació (*mapeig*) d'objectes propis de l'aplicació amb bases de dades relacionals. Això ho fa mitjançant arxius XML o anotacions. *Hibernate* és software lliure publicat sota llicencia GNU LGPL.

Hibernate busca solucionar el problema de la diferencia entre el model da dades orientat a objectes i el model de dades relacionals emprat en els sistemes gestors de bases de dades. Per fer això és permet al desenvolupador detallar com és el seu model de dades, quines relacions existeixen i quina forma tenen. Amb aquesta informació *Hibernate* li permet utilitzar a l'aplicació manipular les dades en la base de dades operant sobre objectes. *Hibernate* genera les sentencies SQL i entrega al desenvolupador la gestió manual de les dades que resulten de l'execució de dites sentències, mantenint la portabilitat entre tots els motors de bases de dades

amb un lleuger increment de tems d'execució. *Hibernate* està dissenyat per a ser flexible en quant a l'esquema de taules utilitzat per a poder adaptar-se al seu ús sobre una base de dades ja existent. De fet en aquest projecte primer vam definir el model de la base de dades, vam executar el *scripts* de creació de taules, vam crear els objectes necessaris en l'aplicació i finalment vam implementar *Hibernate* adaptant la configuració a l'arquitectura existent.

Arquitectura *Hibernate*:



Hibernate ofereix un llenguatge de consulta de dades anomenat HQL, del que també en fem ús en aquest projecte, al mateix temps que una *API* per construir consultes.

Un exemple d'ús d'anotacions, en una definit que serà per una taula anomenada ítems i en la segon definim la seva clau primària:

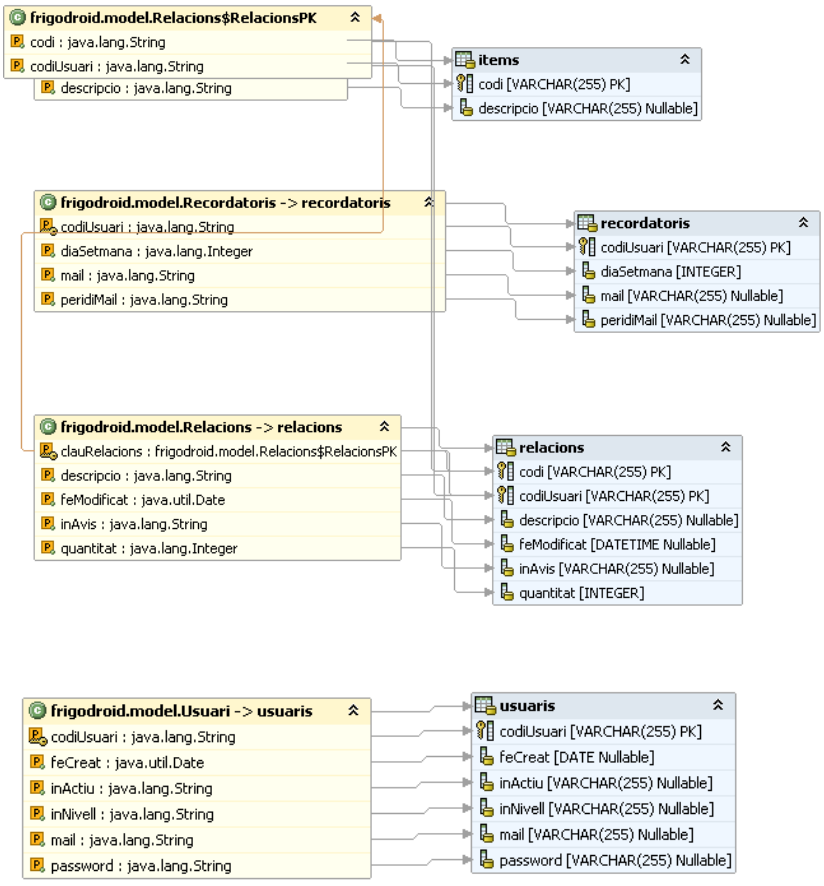
```
@Entity
@Table(name="items")
public class Items implements Serializable, Comparable<Items> {
    /**
     *


    @Id
    public String getCodi() {
        return codi;
    }

    /**
```

Hibernate consta de dos fitxers clau per poder fer-ho servir.

Hibernate.cfg.xml on per poder fer servir *Hibernate* hem hagut de configurar una connexió JDBC, a continuació els detalls de la mateixa:

```
<hibernate-configuration>

	<session-factory>

		<!-- Database connection settings -->
		<property name="connection.driver_class">
			com.mysql.jdbc.Driver
		</property>
		<property name="connection.url">
			jdbc:mysql://localhost:3306/frigodata
		</property>
		<property name="connection.username">root</property>
		<property name="connection.password">osciloscop</property>
```

A continuació mostrem el *mapeig* entre objectes i taules tal com ho relaciona *Hibernate*.



I la classe on gestionem les sessions per *jdbc*:

```
private static SessionFactory buildSessionFactory() {
        try {
                // Create the SessionFactory from hibernate.cfg.xml
                return new AnnotationConfiguration().configure()
                                .buildSessionFactory();
        } catch (Throwable ex) {
                System.err.println("Initial SessionFactory creation failed." + ex);
                throw new ExceptionInInitializerError(ex);
        }
}
        /**
         * @return
         */
        public static SessionFactory getSessionFactory() {
                return sessionFactory;
        }
```

Després fer una *query* no requereix més línies que aquestes:

```
        Session session = HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        Items itemRecuperat =  new Items();
        try{
                itemRecuperat = (Items) session.get(Items.class,item.getCodi());
                session.getTransaction().commit();
```

4.3.1.5. Externs.

4.3.1.5.1. Apache Tomcat

Farem servir la versió 7.0 *d'Apache Tomcat*. És un contenidor de *servlets multi plataforma* que implementa les especificacions dels *servlets* i de les JSP de Sun Microsystems.

Tomcat no és un servidor d'aplicacions com *JBoss* o *JOnAS*. Inclou el compilador Jasper que compila JSP convertint-les en *servlets*. *Tomcat* pot funcionar com servidor web per si mateix to i que a les primeres fases d'aquest servidor es tenia la percepció que només era recomanable per entorns de desenvolupament com en el nostre cas.

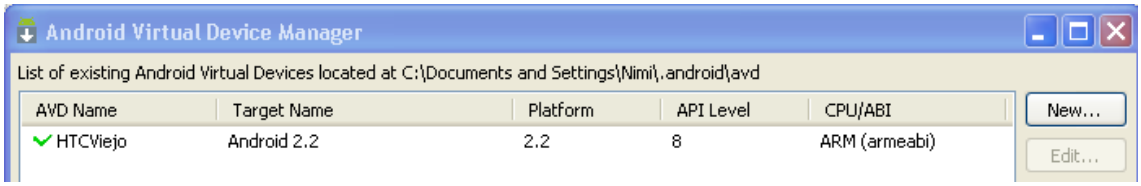
4.3.1.5.2. MySQL Server

MySQL server és un sistema per a la gestió de bases de dades relacional, multi fil i multi usuari. S'ofereix sota llicencia GNU GPL.

Per a l'entorn de desenvolupament hem fet servir la versió 5.5 d'aquest servidor, ha sigut escollit donat que és compatible amb l'ús *d'Hibernate* i ens facilita aprofitar la flexibilitat *d'Hibernate* a l'hora de poder construir sentències SQL que per desconeixement propi no seria capaç de fer en HQL.

4.3.1.5.3. Android.

La part del projecte *d'Android,* encara que estem treballant amb la última versió del SDK aquest projecte té com a *target Android* 8 això vol dir que està dissenyada per poder córrer a partir de la versió *Froyo* 2.2 d'*Android.*

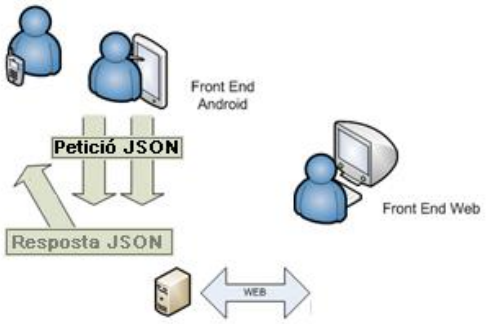


Per motius de recursos l'aplicació ha estat optimitzada per pantalles petites. En aquest cas les proves de l'aplicació s'han dut a terme amb un HTC Màgic de 320 x 480 *px* amb 180 *ppi*.

Per *Android,* que té una arquitectura semblant a la del servidor web donat que també fa servir el patró MVC, hem fet servir tres paquets de classes:

-Útils, trobem tot allò agrupat que es pot arribar a fer servir des de les classes *activity,* com les crides i peticions a servidor.

- Model tractem de representar els objectes necessaris per a la gestió del nostre model de negoci.



- I finalment el paquet amb totes les classes *activity*. De la mateixa forma que el servidor són les *action* aquí les *activity* són les primeres de la capa de model que reben la petició i comencen a gestionar la resposta per a la part controladora.

Podem trobar força analogies entre l'arquitectura del servidor web i la del client *Android.* Encara que trobem molts noms canviats.

Pel tractament de les respostes per part del servidor fem servir JSON. És un subconjunt de la notació literal d'objectes de JavaScript que requereix *XML*. S'ha establert com l'alternativa a l'ús d'*XML* també per *AJAX* també en pel projecte de l'aplicació web. A continuació mostrem un exemple de mètode que fem servir per recuperar descripcions d'ítems una vegada escanejats:

```
public static String wsDescripcio(String codi) {
    GsonBuilder gsonBuilder = new GsonBuilder();
    Gson gson = gsonBuilder.create();
    HttpPost httpPost = new HttpPost("http://"+IP+":8080/FrigoDroidServer/getJS
    String json = gson.toJson(codi, String.class);

    String jsonRebut = cridaWS(json,httpPost);
    Items item = gson.fromJson(jsonRebut, Items.class);
    if (item==null){
        return "buit";
    }else{
        return item.getDescripcio();
    }

}
```

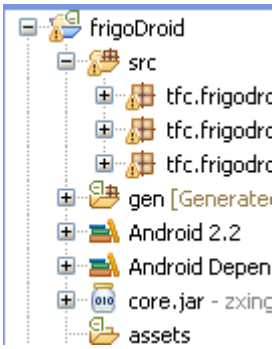
Podem veure que primer creem la petició per post, efectuem la crida i recollim el resultat on amb l'ajuda de *GSON* convertim l'objecte *JSON* en un objecte ítems. Val a dir, que aquest mètode només s'hauria de fer servir quan tractem amb una font de confiança donat que sinó mai podríem transformar el resultat en quelcom conegut.

La part de l'escàner s'ha fet utilitzant les llibreries *ZXing*, un projecte *open source* sota llicencia *Apache*, que ens permet treballar amb codis de barres multi format. No explicarem res mes donat que des de el punt de vista de desenvolupament aquestes llibreries són una caixa negra que ens acompanya durant tot el desenvolupament, de la que només ens hem d'encarregar de saber com fer les crides oportunes. Si que me de saber de totes formes que hi ha dos formes de desenvolupar el projecte fent servir aquestes llibreries:

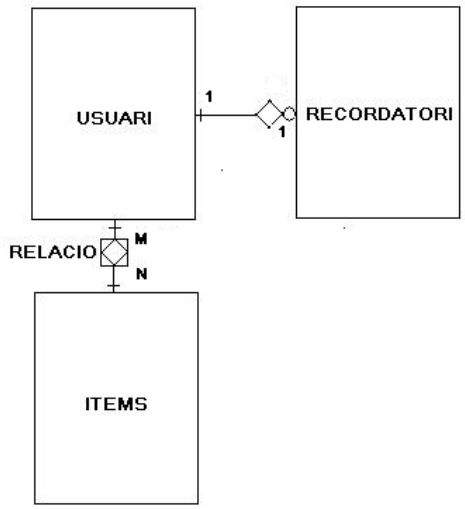
- La primera és fent servir l'aplicació pròpia de *ZXing* anomenada *BarCode Scanner*, la cridem com una *activity* normal i aquesta ens retornarà el codi escanejat. És la recomanada pels desenvolupadors de *ZXing* donat que així l'usuari pot gaudir de les

últimes millores i en cas que l'escàner falli per qualsevol motiu ens assegurem que l'usuari es pot adreçar les errades i consultes a l'equip de *ZXing*.

- La segona es compilant el projecte amb ANT i incloent el *core.jar* resultant en les nostres llibreries. D'aquesta forma l'usuari no necessita instal·lar cap altre aplicació per fer córrer el nostre projecte, així tenim una aplicació *stand-alone*. Aquesta és l'opció que hem triat, encara que les recomanacions de l'equip de *ZXing* les trobo totalment encertades i de cara a un altre projecte pot ser triaria no implementar el codi de barres propi a l'aplicació.



4.3.2. Diagrama d'entitats.



USUARI(codi usuari, *password*, mail, indicador nivell client, indicador_actiu)

ITEMS(codi, descripció)

RELACIO(Usuari, Ítem, descripció ítem, quantitat, data modificació, indicador avís)

On {Usuari} relaciona a USUARI

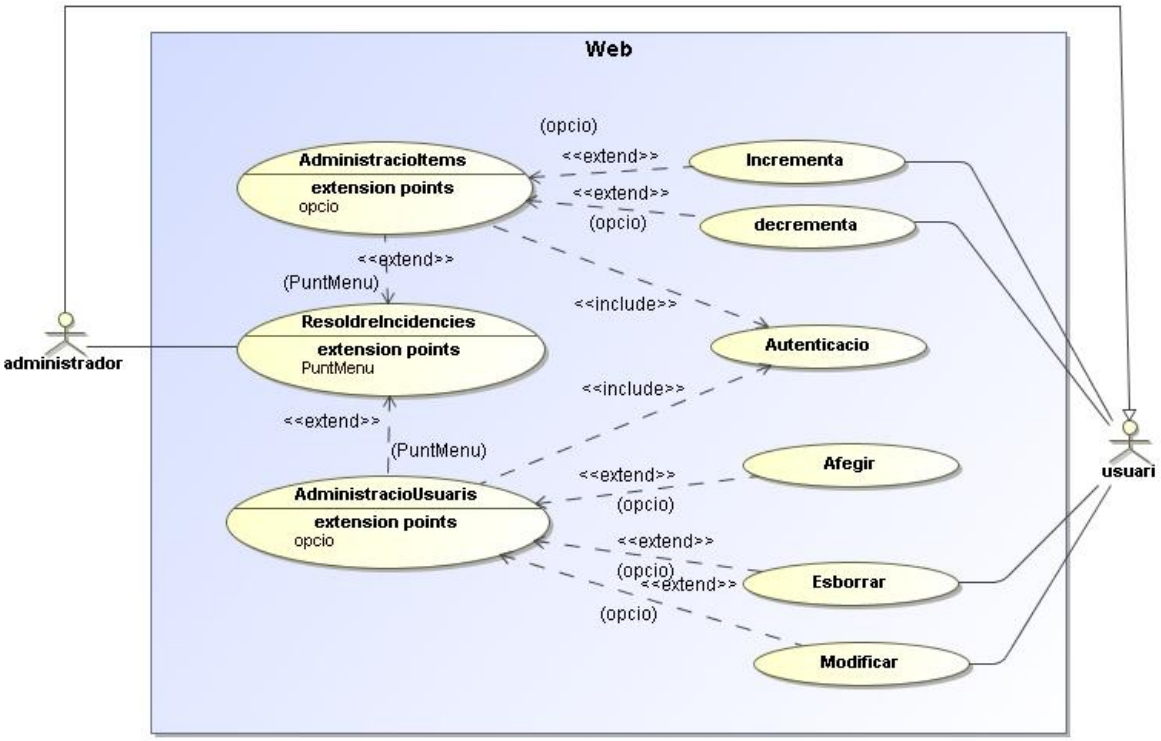
I Ítem a ITEMS.

RECORDATORI(Usuari, mail, periodicitat mail, dia setmana mail)

On {Usuari} relaciona a USUARI

4.3.3. Diagrames casos d'ús.

4.3.3.1. Casos d'ús per la web.



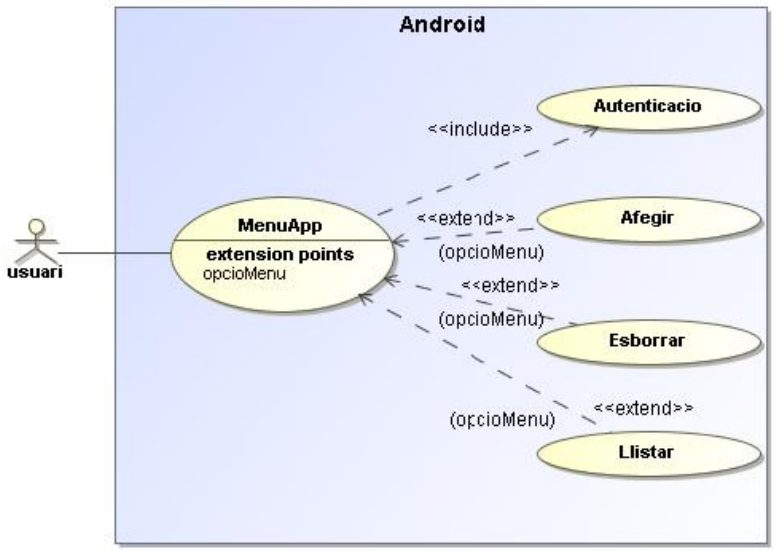
En aquest cas tenim dos actors, l'administrador i l'usuari. L'administrador no és més que un usuari amb drets especials. L'usuari podrà accedir mitjançant l'autenticació a poder incrementar i decrementar els seus ítems, a modificar les seves dades i a esborrar el seu usuari.

L'administrador per altre part mitjançant la pàgina de la web principal té accés a totes les funcionalitats existents. Podríem considerar que l'administrador en el moment de resoldre incidències pot accedir a dos àrees, la d'administració d'ítems assignats als diferents usuaris i la d'administració d'usuaris pròpiament a la qual l'usuari no té accés donat que podria gestionar comptes que no són seus.

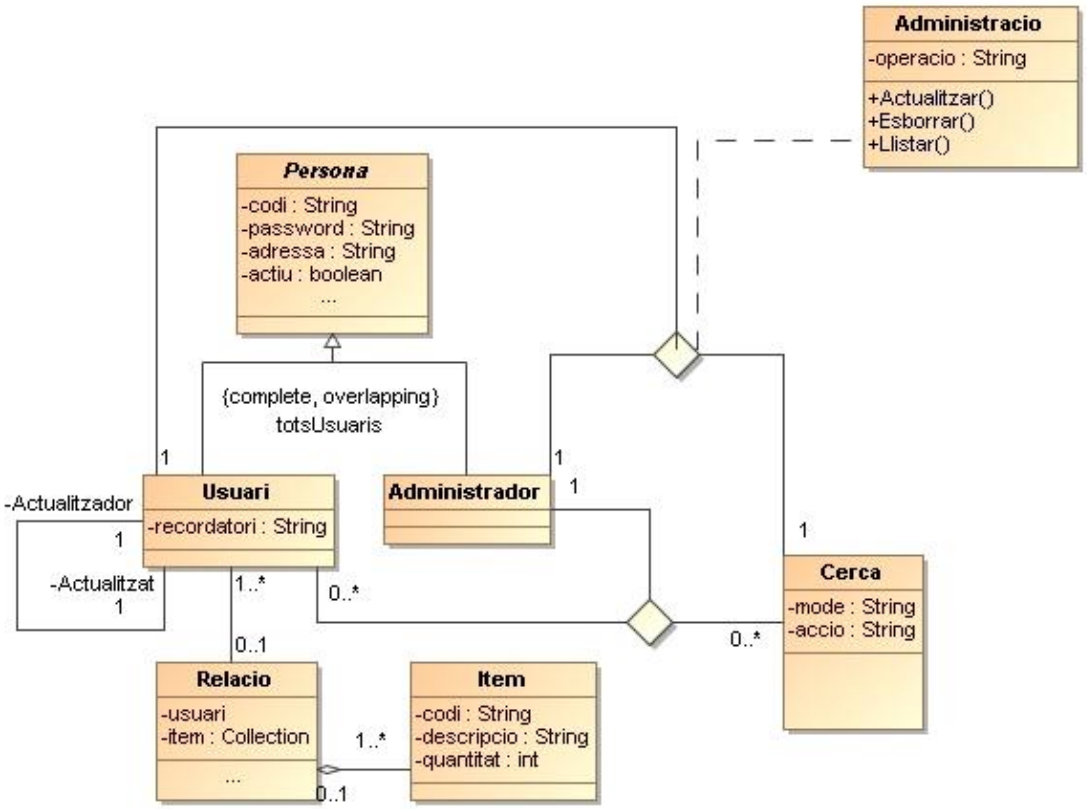
Hem de tenir en compte que per a totes les funcionalitats és requerit l'autenticació per part dels diferents actors.

4.3.3.2. Casos d'ús pel client *Android*.

En aquest cas, l'actor és l'usuari, mitjançant el menú de l'aplicació podrem afegir i esborrar ítems, demanar una relació d'ítems assignats al nostre usuari. L'autenticació és necessària.



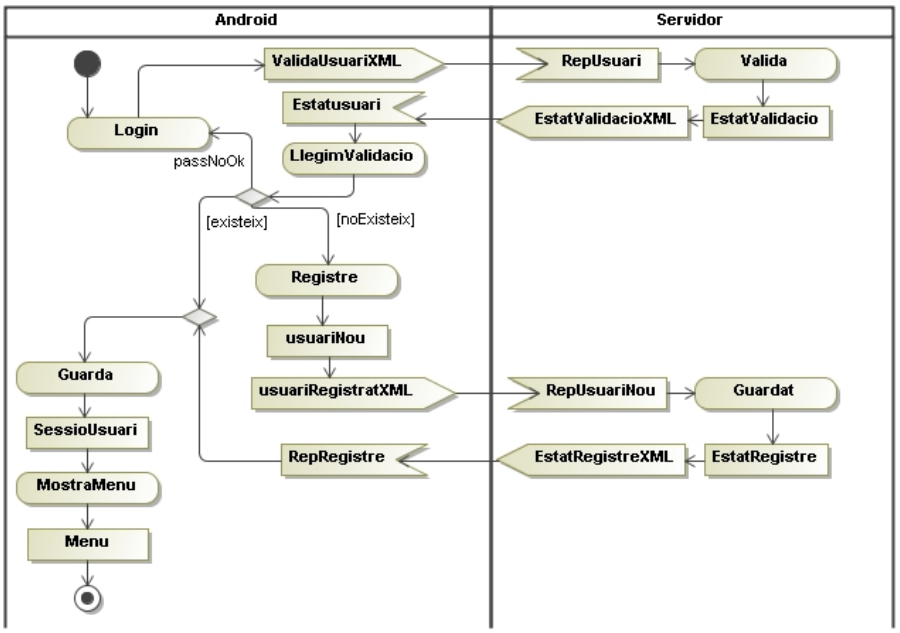
4.3.4. Diagrama de classes.



Tota persona ha de tenir unes propietats, encara que és una classe abstracta que no es pot instanciar perquè haurem d'instanciar com usuari o administrador. Cada administrador mitjançant una cerca pot accedir a 0 o molts usuaris llistats. Cada administrador una vegada hagi fet una cerca podrà administrar un usuari alhora. Cada usuari es podrà actualitzar a si mateix prenent rols d'usuari que s'actualitza. L'usuari es té una relació d'ítems, cada relació serà una agregació d'ítems, poden haver usuaris sense ítems.

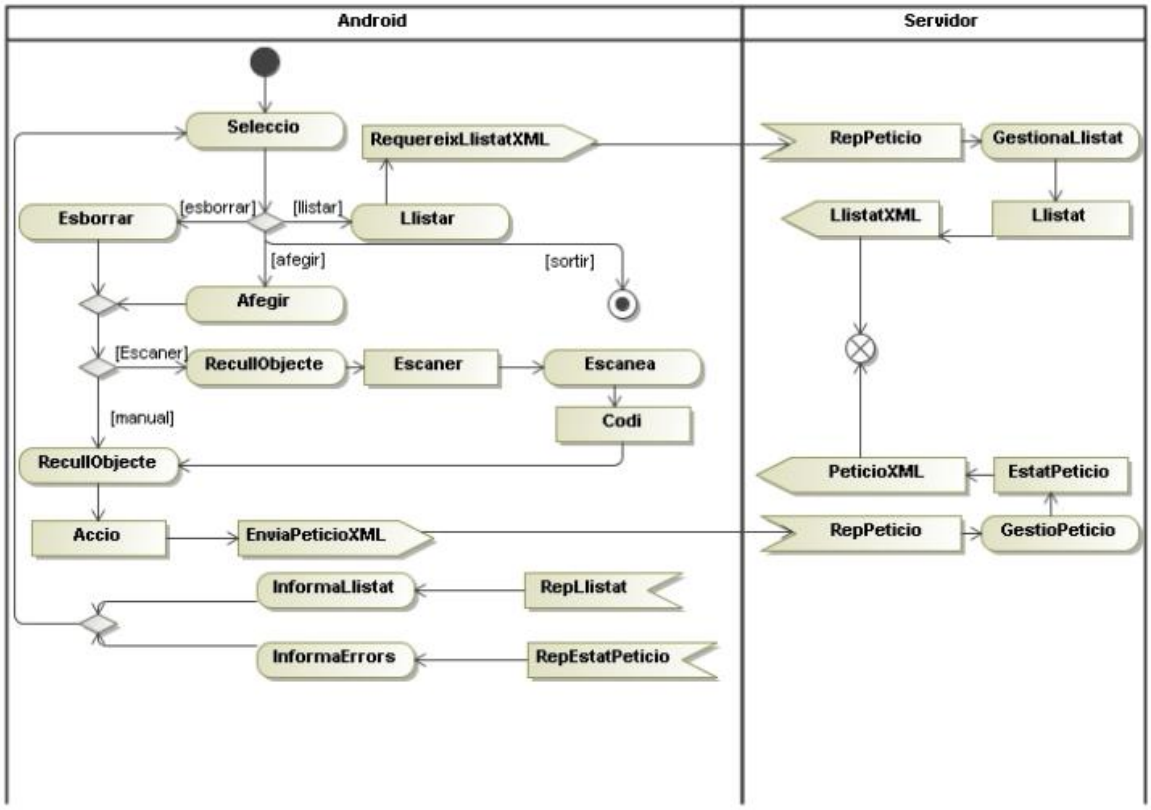
4.3.5. Diagrama d'activitats.

4.3.5.1. *Android* – Identificació usuari.



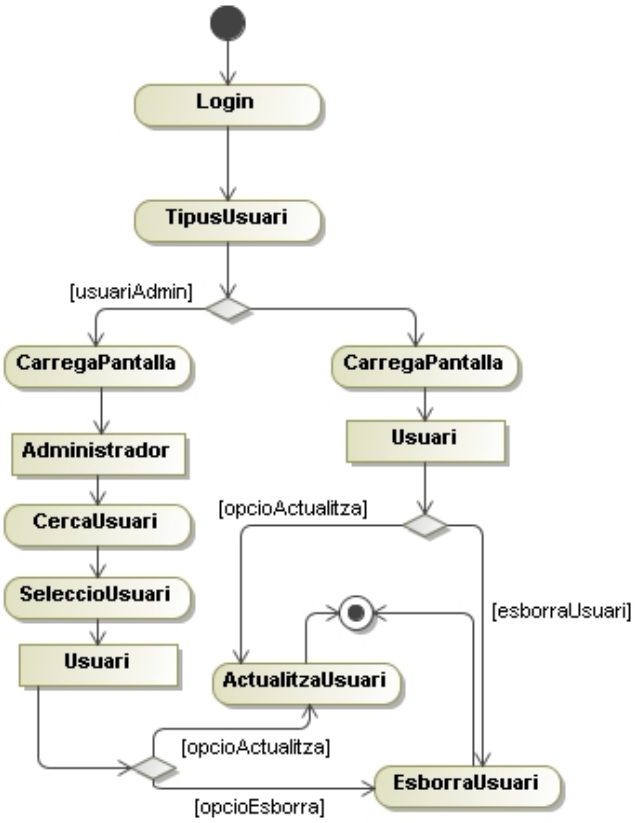
El client que farem per *Android* només considerarà l'usuari com a no administrador. Validarà l'usuari contra el servidor fent servir comunicació en fitxers XML. En cas que la validació si positiva guardarem la sessió i mostrarem el menú de la pantalla principal de l'aplicació. En cas que no existeixi forcarem a l'usuari a registrar-s'hi mostrant la pantalla de registre on una vegada s'omplen les dades principals podem enviar una petició en format XML al servidor perquè ens afegeixi aquest usuari a la base de dades. Si el procés ha anat correctament rebrem la resposta i continuarem el flux normal de l'aplicació creant sessió i mostrant la pantalla principal de l'aplicació.

4.3.5.2. *Android* – Operativa usuari.



Inicialment l'usuari pot llistar una opció per operar. En el cas que escolli llistar farem una petició *XML* al servidor que ens haurà de retornar la relació d'ítems per aquest usuari en format *XML* i l'aplicació haurà de presentar-ho per pantalla. En cas d'afegir o esborrar ítems, hem de mirar quin mètode ha escollit l'usuari (escàner o manual). En cas que hagi escollit escàner haurà de presentar el l'objecte per escanejar codis de barres i aquest quan tingui un codi escanejat ens ho haurà de retornar com si hagués sigut l'usuari qui ho hagués entrat per pantalla de forma manual, encara que en tot cas nosaltres hem de presentar la descripció de l'objecte. Una vegada tinguem la informació relativa a l'objecte amb el que l'usuari vol treballar podrem formatar la petició *XML* per poder enviar-la al servidor. Rebrem una *XML* de resposta amb l'estat de la nostra petició, acabada o pendent per error. Acte seguit comencem el flux de nou per tal que l'usuari pugi entrar més ítems si desitgés.

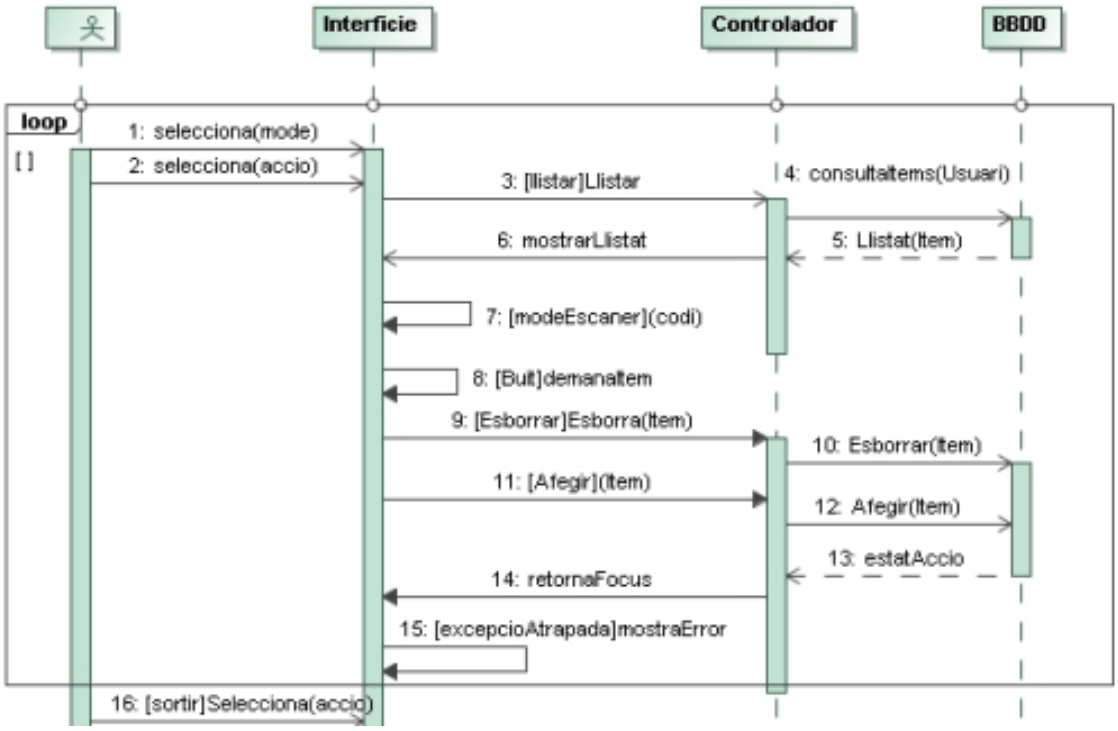
4.3.5.3. Web.



En aquest cas considerem que tant per l'administrador com per l'usuari normal. En aquest cas tant l'usuari com l'administrador comparteixen funcionalitats, actualitzar informació personal d'usuari o ítems ja que tot passa per la mateixa acció que és la d'actualitzar, esborrar usuari tot i que l'administrador ho pot fer seleccionant usuaris diferents.

4.3.6. Diagrama de seqüències.

- Cas usuari fent servir *Android*:

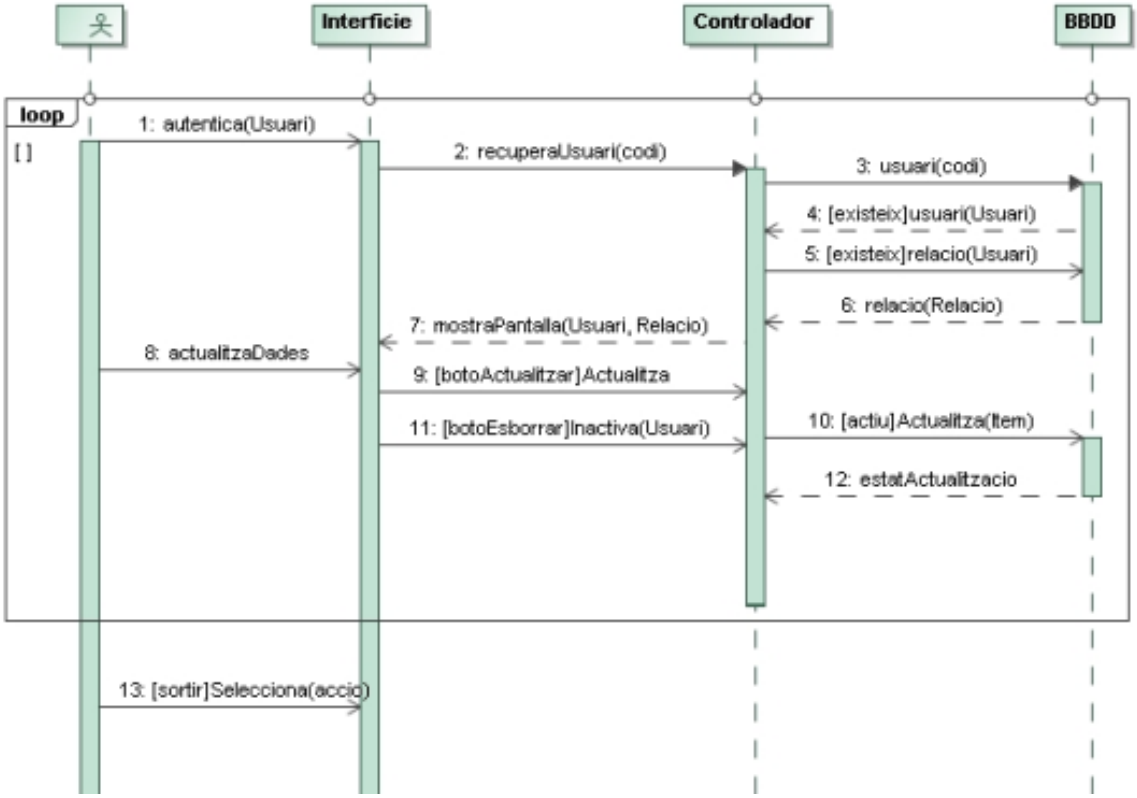


Assumim que l'usuari ja ha fet s'ha identificat. Podrà seleccionar el mode d'entrar les dades i si ha seleccionat manual podrà escollir posar la descripció de l'ítem i el nombre d'unitats o be esperar a que es carregui l'escàner per poder recollir el codi de barres que ens retornarà el seu codi. En cas que sigui buit podem tornar a demanar les dades. Una vegada tenim la informació necessària(descripció i quantitat) esperem que l'usuari premi o bé esborrar o bé afegir. No incloem un activitat entre l'actor usuari i d'interfície perquè no aporta gaire valor i ho controlem amb el punt 8 d'aquest diagrama.

Segons afegim o esborrem activem una guarda que ens dirigeix a l'acció entre el controlador per poder esborrar o activar. Una vegada completada una de les dues accions haurem d'obtenir un resultat sobre l'execució que haurem d'escalar cap a la interfície per si l'usuari hagués d'intervenir o fer cap consulta al administrador del sistema.

El bucle es trenca si l'usuari demana sortir de la aplicació.

- Cas usuari fent servir web:



En aquest cas si tenim en compte l'autenticació per tal que quedi descrit en algun moment. Una vegada l'usuari s'identifiqui recuperem l'usuari mitjançant un codi d'usuari, la classe controladora atacarà la base de dades i aquesta ens retornarà la relació d'ítems per l'usuari.

Acte seguit recuperarem les dades de l'usuari per poder mostrar-les i permetre l'usuari que pugi modificar-les. Això només podrà ser en cas que l'usuari sigui actiu. Si l'usuari decideix prémer el botó d'esborrar usuari aquest quedarà desactivat a la base de dades i per tant no podrà tornar a actualitzar res. Si s'hagués produït una inactivació per error hauria de comunicar-ho a l'administrador.

4.4. Valoració econòmica.

Tot i que el projecte *FrigoDroid* està orientat en una estratègia B2C basat en un model de publicitat i que s'estableix que hauria de ser gratuït per tal de poder assolir els objectius per

aquesta part del projecte, és a dir tenir el màxim nombre d'usuaris possibles. Haurem d'establir una valoració del projecte per tal de poder estimar el moment en el que el projecta hagi estat amortitzat.

Ens centrarem en el pla de treball establert en la PAC1 donat que no podem comptabilitzar totes les hores d'aprenentatge dedicades perquè no les podem imputar al projecte ja que ens donaria un valor força més irreal. Al pla de treball tenim els següent períodes establerts:

PAC2: Lliurament 19/04 (30 dies dedicats)

Proves conjunt tecnologies(10 dies).

Anàlisis (10 dies)

Disseny TFC.(10 dies)

PAC3: Lliurament 04/06 (45 dies dedicats a desenvolupament).

Memòria: 14 dies dedicats. Tot i que aquesta part no és pròpia de desenvolupament la entenem com necessària per enllestir documentació funcional i tècnica, a més de l'execució i reparació defectes de l'aplicació.

Així doncs tenim una suma total de 30+45+14= 89 dies de dedicació, és a dir, 12.7 setmanes. Estimem que durant cada setmana hem dedicat una mitja de 18 hores. Això fan 228 hores. Podem establir un preu un preu de 35 €/h, per una persona dedicada al projecte, que no és gens desorbitat si ens centrem en una consulta a la pàgina http://www.infolancer.net/freelancers/jefe_de_proyecto.

A més hem d'afegir els costos del manteniment del servidor *Tomcat*, encara que això dependrà de si tenim altres projectes, si és dedicat, del pla de preus, dels dominis contractats. És ben dificil establir una xifra concreta. Per tant arrodonim a un resultat de 8000 euros inicials.

Posar en marxa aquest projecte ens hagués costat 8000 euros aproximadament, així doncs tenint un sistema de publicitat per aplicacions mòbils com *adMob* amb *eCPM* aproximat de $0.09, *MobFox* amb *eCPM* aproximat de $4, *MobClix* amb *eCPM* aproximat de $1; podem establir un període d'amortització depenent del ràting del sistema de publicitat escollit i del nombre de visites i usuaris aconseguits, encara que els ràtings com els usuaris poden variar al llarg del temps. També podríem incloure un sistema com *Adsense* per exemple pel servidor tot i que estimem que no tindrà tantes visites.

4.5. Conclusions

Es va iniciar aquest TFC amb l'objectiu refermar els coneixements sobre la tecnologia J2EE donat que com a programador COBOL he pogut tenir participació en petits desenvolupaments J2EE però no tenia coneixement de l'arquitectura que hi havia darrere. De totes les paraules que estan relacionades només m'era familiar *Struts*, encara que no havia tingut oportunitat d'entendre què era.

Un altre objectiu era el de posar un peu en el món *Android* creant una petita aplicació, aquest món ho trobo fascinant i em dona la sensació que ara per ara són el futur.

Finalment, una vegada finalitzat el projecte, trobo que els objectius han estat complerts. Trobo que la corba d'aprenentatge ha sigut bestial. Que sincerament no sabia on m'endinsava, però que malgrat tot el patiment ha sigut tota una aventura fascinant i em deixa ganes de més. La corba d'aprenentatge ha sigut molt accentuada donat que si programes de forma estructurada la primera barrera que has de trencar es esquematitzar el teu cap en el paradigma de l'orientació a objectes, és a dir: pensar, dissenyar i construir una solució sencera pensant orientat a objectes.

Encara que els objectius han estat complerts i que estic content amb el resultat obtingut passo a analitzar el projecte en qüestions de qualitat:

Trobo que l'arquitectura de la que hem fet ús ha sigut clau pel desenvolupament. Permet refer parts de l'aplicació fàcilment. Això ho podem traduir en que aquesta arquitectura ens permet un manteniment fàcil.

Aquest projecte és escalable, vam començar amb un mòdul, li vam afegir la part de comunicació amb *Android,* posteriorment vam refer la capa de vista de la web, vam afegir *Hibernate* i amb els pas de les fases ens adonàvem que no havíem hagut de patir gaire en afegir cada fase i que no havíem de mantenir codi vell. Tot estava funcionant, endreçat i modulat.

Tot i que no hem tingut força temps trobo que, ara per ara, el projecte és bastant fiable. Hi ha parts que no hem pogut provar, per falta de temps, però fins on hem provat no hem trobat errades significants i si s'han trobat les hem pogut solucionar.

En termes d'usabilitat és en el punt que més fàcil ho hem tingut amb aquesta arquitectura i més content podem estar. Tant *Struts, AJAX com Android* estan dissenyats per fer lliurable amb un

mínim de qualitat notable. El fet que no s'hagi de reinventar la roda cada a cada pas, el fet que tinguem força eines de les que disposar fa que ens podem dedicar a programar gaire bé la lògica de negoci deixant la part d'arquitectura gaire bé funcionant des del primer moment.

Aquest últim punt fa que el "*time to market*" d'aquest tipus d'aplicacions sigui realment petit.

Trobo que he aprés molt, que ara mateix si tornés a fer un projecte igual el temps de desenvolupament es reduiria notablement. Tot i que la funcionalitat no es gaire complicada no podia posar en risc el projecte degut a la quantitat de coses que havia d'incloure en la meva corba d'aprenentatge. Puc dir que tot el que he fet ho entenc i que podria fer-me càrrec del manteniment del projecte o de futures ampliacions sense cap problema.

4.5.1. Funcionalitats futures.

Les diverses funcionalitats que podríem considerar per un futur són molt variades donat la versatilitat de la funcionalitat que estem dissenyant. A continuació passem a descriure algunes de les funcionalitats que en el seu moment es van establir en la fase de disseny o bé són fruit de tot el procés d'implementació i proba.

- Encriptació de la informació sensible com usuari i contrasenya tant en les peticions JSON com en les bases de dades. Per fer això podríem comptar amb alguna llibreria MD5.

- Escaneig continu: ara per ara cada vegada que escanegem un ítem tornarem a la pantalla principal de l'aplicació, seria bo que no tornéssim fins que l'usuari premés un botó del telèfon per tal de donar més velocitat al registre de quantitats mitjançant l'escàner. És a dir, una vegada fem clic a l'opció desitjada, afegir per exemple, podríem escanejar articles contínuament fins finalitzar i aquests podrien ser a la vegada enregistrats al servidor.

- Sistema multi usuari apuntant a un mateix frigorífic. personalitzar el consum de cada usuari per a situacions com un pis compartit on tothom aporta de forma independent a un mateix frigorífic o controlar de forma més precisa el consum individual de cada un dels membres de la família.

- Sistema de comunicació amb la botiga per tal de comprar els articles bàsics que l'usuari anés consumint amb una periodicitat preestablerta.

5. Glossari.

Action: Són les classes establertes en l'arquitectura MVC per recollir las interaccions amb l'usuari, treballar la informació de negocio i retornar-la en forma de resposta. Aquestes en el nostre cas estan definides al fitxer de Struts.

AJAX: Asíncron *javascript* i *XML*. Técnica de desenvolupament web per crear aplicacions interactives o *RIA*. S'executa en client i manté una comunicació de forma asíncrona amb el servidor que permet canvis sobre les pàgines sense necessitat de tornar a carregar—les.

Android Market o Google Play: Aplicació *d'Android* que permet als usuaris buscar i descarregar la majoria de les aplicacions disponibles per a *Android*.

Android: Sistema operatiu basat en Linux enfocat per a fer-se servir en aparells de telefonia mòbil, tauletes electròniques i altres dispositius. Desenvolupat per la Open Handset Alliance, liderada per Google.

COBOL: *Common Busness-Oriented Language.* Llenguatge de programació creat al 1959 destinat principalment per a la informàtica de gestió.

EAN13: Sistema de codi de barres constituït per 13 dígits.

eCPM: Cost estimat per mil impresions.

HQL: *Hibernate Query Language*. Sistema que ofereix *Hibernate* per poder fer consultes a les bases de dades.

Framework: Conjunt estandarditzat de conceptes, pràctiques i criteris agrupats en una estructura conceptual i tecnològica en la que ens podem basar per poder desenvolupar altres projectes.

GSON: Programari obert. Biblioteca que permet la serialització i deserialització entre objectes Java i la seva representació en objectes *JSON*.

Hibernate: Eina pel mapeig objecte - relacional per a la plataforma Java que facilita el mapeig d'atributs entre una base de dades i el model d'objectes d'una aplicació. Mitjançant la

declaració per fitxers *XML* o per anotacions en els *beans* de les entitats que permeten establir aquestes relacions.

IDE: Entorn per desenvolupar programari.

J2EE: Plataforma de programació per desenvolupar i executar programari en el llenguatge de programació *Java* i en arquitectura de *n capes* distribuïdes i que es recolza àmpliament en components de programari modulars sobre un servidor d'aplicacions.

JQuery: Programari lliure i de codi obert. Bilbioteca *Javascript* creada per simplificar la forma d'interactuar amb els documents *HTML*, manipular l'arbre *DOM*, fe servir events, desenvolupar animacions, i afegir interacció amb la tècnica *AJAX*.

JSON: Format lleuger pel intercanvi. Alternatiu i/o complementari a *XML*.

JSP*: Java Server Pages*. Tecnologia *Java* que permet generar contingut dinàmic per a la web.

Javascript: Llenguatge de programació interpretat, fet servir en la part del client, implementat com part d'un navegador web permetent millores en la interfície d'usuari i pàgines web dinàmiques.

MVC: Patró d'arquitectura de programari que separa les dades d'una aplicació, la interficie d'usuari i la lògica del negoci.

RIA: Aplicacions d'Internet enriquides. Aplicacions que tenen les característiques de les aplicacions comuns d'escriptori.

Servlet: Conjunt de classes que s'executen al servidor.

Smartphones o telèfon intel·ligent: Telèfon mòbil construït sobre una plataforma mòbil amb una gran capacitat de computació i connectivitat comparat amb un telèfon mòbil convencional. El terme intel·ligent fa referència a la capacitat de fer-se servir com una computador de butxaca.

Struts2: Eina de suport d'aplicacions web sota el model Model – Vista – Controlador sota la plataforma JEE.

Tablet o tauleta: Computadora portàtil de mida més gran que un telèfon intel·ligent integrat en una pantalla tàctil.

Tiles: Marc de treball en plantilles construït per simplificar el desenvolupament de les interfícies d'usuari. Ajuda al desenvolupador a definir les pàgines per fragments.

Widget: Petita aplicació o programa que es solen fer servir freqüentment i estan dissenyats per a donar accés visual i ràpid a la informació sol·licitada.

XML: *eXtensible Markup Language*. Llenguatge de marques que permet definir una gramàtica en els llenguatges específics per estructurar documents grans. Dona suport a base de dades.

6. Bibliografia.

En forma general he consultat força al cercador *Google* i m'han fet molt d'ús *http://viralpatel.net*, *www.androideity.com,* www.mkyong.com, stackoverflow.com i www.vogella.com. Sense aquestes no hagués pogut tirar endavant el projecte.

En el període de *Hello Worlds* em vaig basar en:

J2EE General:

http://www.coderanch.com/forums

Struts2:

http://viralpatel.net/blogs/2009/12/struts2-validation-framework-tutorial-example.html

http://viralpatel.net/blogs/2010/01/tutorial-struts2-hibernate-example-eclipse.html

***Hibernate*:**

http://docs.jboss.org/hibernate/core/4.0/quickstart/en-US/html_single/

http://www.hibernate.org/docs

http://viralpatel.net/blogs/2010/01/tutorial-struts2-hibernate-example-eclipse.html

J2EE General:

http://www.coderanch.com/forums

Tomcat:

http://efunctions.wordpress.com/2011/12/19/configurando-apache-tomcat-7-en-eclipse-indigo/

http://mundogeek.net/archivos/2009/02/02/eclipse-y-tomcat/

Android:

http://developer.android.com/resources/tutorials/hello-world.html

http://www.weterede.com/2009/06/instalacion-sdk-de-android-en-eclipse/

http://www.ignacionario.com/2011/03/android-iv-diseno-de-layouts-o.html

http://developer.android.com/resources/tutorials/views/index.html

http://www.slideshare.net/SaamiirLpR/ejemplo-de-aplicacin-android-hola-mundo-botones-intents

http://stackoverflow.com/questions/2154438/how-to-implement-remember-me-function-in-login-of-android-activity

Escàner:

http://androideity.com/2011/11/23/trabajar-con-codigos-qr-en-tus-aplicaciones-android/

http://code.google.com/p/zxing

JSON/GSON:

http://www.mkyong.com/struts2/struts-2-and-json-example/

http://www.vogella.com/articles/AndroidJSON/article.html

Per completar aquesta memòria he fet servir també:

http://es.wikipedia.org

7. Annexos.

7.1. Manuals intal·lació i usuari.

En el cas *d'Android* no indiquem manual d'instal·lació de l'aplicació com executable en sí donat que per fer això s'hauria de pujar l'aplicació al *market* i considerem que encara està en una fase de desenvolupament i si la pengéssim al *market* seria en una fase d'entrada a producció de l'aplicació.

Pel cas del servidor tenim dues opcions podem aixecar el *Tomcat* amb el *.war* entregat o bé podem escollir arrancar el servidor des de *Eclipse*. Per aixecar *Tomcat* de forma autònoma només s'ha de desar el *war* en la carpeta webapps, assumim que l'usuari té correctament configurat el *Tomcat*. Sinó pot seguir les passes aquí indicades:

http://www.programacion.com/articulo/tomcat_-_introduccion_134

7.1.1. Manual usuari *Frigodroid*.

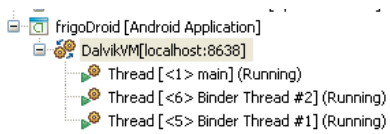
Manual d'instal·lació i ús de l'aplicació:

Android:

Ens haurem d'assegurar que *Android* apunta al servidor correctament. En la classe *JSonUtils.java* del projecte tenim la IP que haurem de modificar en funció de la nova IP:

private final static String IP = "192.168.1.34";

Podem córrer l'aplicació en l'emulador que *Eclipse* disposa.



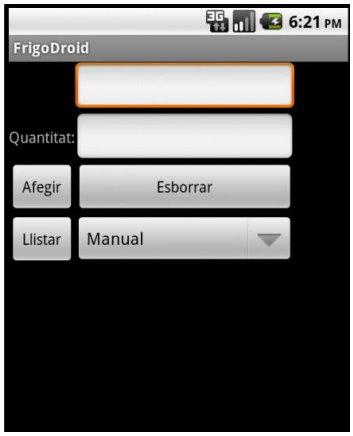
DalvikVM és la màquina virtual de la que disposem per poder fer córrer aplicacions *Android* dins d'Eclipse.

Una vegada engeguem el projecte si l'aplicació ha estat instal·lada (fitxer *.apk*) correctament haurem de veure la pantalla d'autenticació en l'emulador.

Tenim dues opcions, si no disposem d'usuari ens hem d'enregistrar. Si disposem l'haurem d'introduir per tal d'accedir a l'aplicació. Les posteriors vegades l'aplicació recordarà l'usuari i contrasenya en el formulari d'accés per estalviar temps a l'usuari. Si aquest vol canviar d'usuari haurà d'esborrar les dades existents en les caixes de text i posar la informació nova abans de prémer el botó per autentificar-s'hi.

Aquí veiem com en la següent vegada que accedim el formulari no està buit:

Una vegada accedim a l'aplicació aquesta és la pantalla, en aquest moment l'usuari pot afegir un ítem per la seva descripció i fent clic en afegir i en esborrar podrà modificar el seu inventari. No és necessari fer ús de l'escàner.



Si prenem llistar obtenim una pantalla com la següent:

Podem apreciar que la icona de *l'Android* tatxada vol dir que no tenim existències. L'aplicació tolera quantitats negatives donat que és pot donar en un qualsevol inventari i pot ser una alarma pels gestor del inventari.

Finalment resta mostrar la funcionalitat d'escaneig. Aquesta funcionalitat s'activa quan seleccionem l'opció d'escàner a la pantalla principal.

Se'ns obrirà la pantalla per escanejar:

Aquesta imatge no correspon al projecte, la real és en català.

Acte seguit si l'escaneig ha sigut satisfactori ens retornarà el codi de barres i ens ho suggereix com descripció de l'objecte que tractem però podem procedir a modificar la informació desitjada:

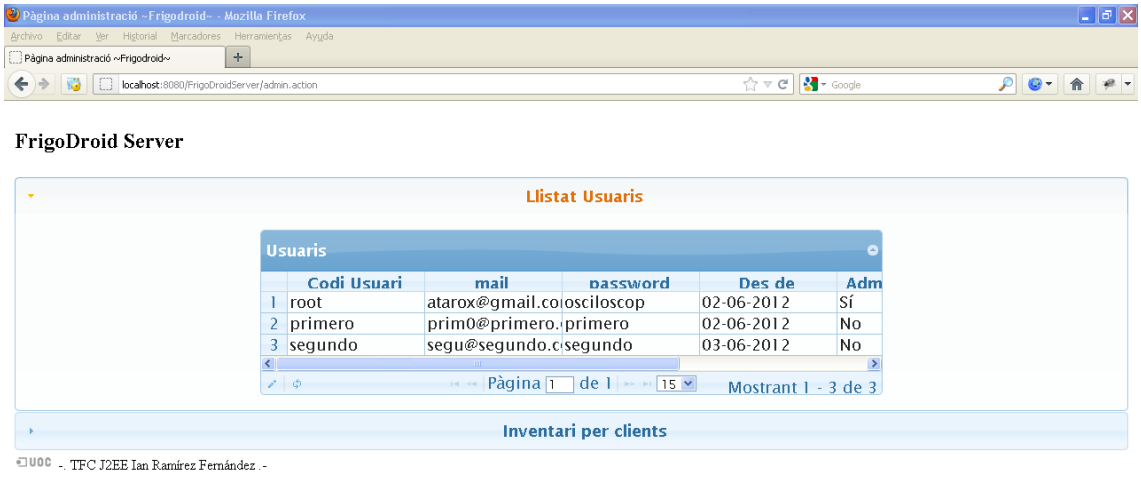
Així l'aplicació recorda que pel codi de barres 00898... tindrem una descripció "producte inventat" i una quantitat de 9. Si no modifiquem la descripció suggerida descripció i codi seran iguals.

7.2. Manual usuari *Frigodroid Server Administració*.

En aquest cas podem accedir a la URL de la web:

Podrem fer servir el nostre usuari, i depenent del seu perfil podrem accedir a una finestra o un altre.

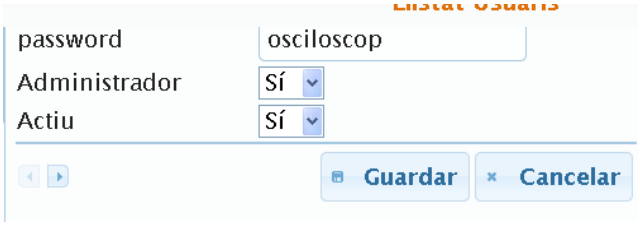
Si accedim amb un usuari administrador accedim al finestra principal on tenim desplegat el llistat d'usuaris:



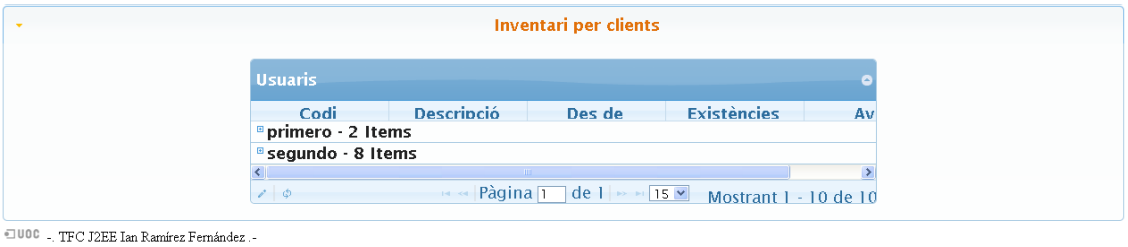
Trobarem tota la informació referent a l'usuari: codi, adreça de contacte, paraula de pas, data de creació si és administrador i si és actiu.

Si fem clic al llapis que tenim a sota de la finestra podrem accedir al panell de modificació:

Una vegada modifiquem les dades desitjades hem de fer clic en guardar:



Si accedim a l'acordió d'inventaris per clients tenim la mateixa operativa amb diferent presentació:



Podrem desplegar cada usuari:

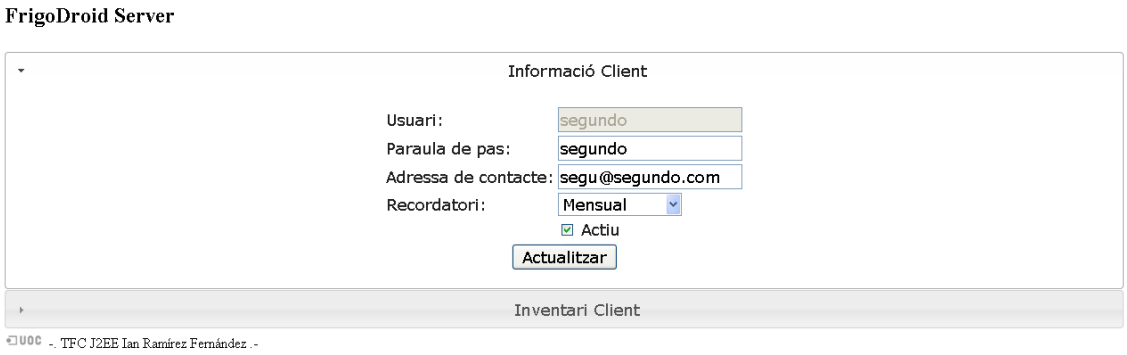
Si volem modificar haurem de seleccionar alguna filera sinó obtenim un missatge informatiu:

El menú de modificació del inventari és el següent:

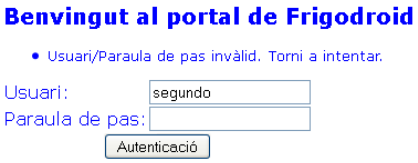
A través de les fletxes que tenim podem navegar pels diferents ítems presentats en la taula sense haver de seleccionar una nova cada vegada.

7.3. Manual usuari *Frigodroid Server Client*.

Una vegada ens autentiquem com usuaris amb perfil client tenim la següent pantalla principal:

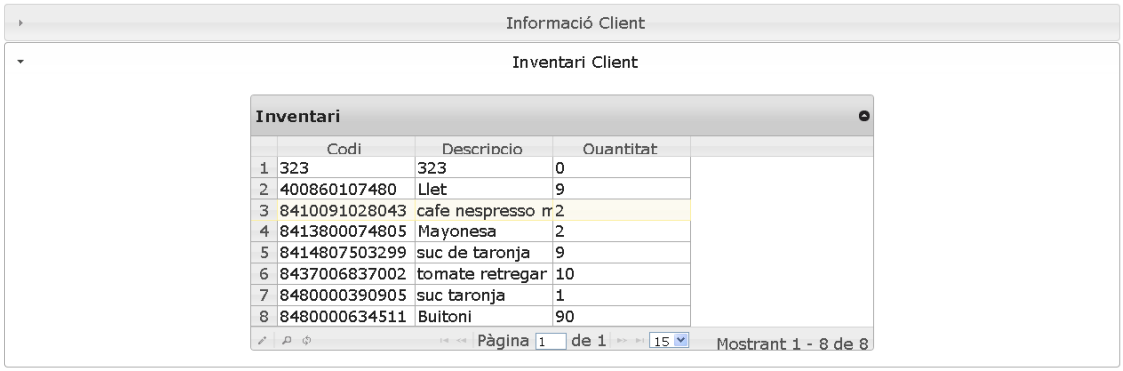


En aquest cas podem veure que l'usuari és capaç de modificar tota la seva informació. Una vegada fem clic en actualitzar aquesta anirà a parar a la base de dades. En cas que desactivem l'usuari el sistema ens redireccionarà a la pantalla de per identificar-se una vegada allà no ens podrem identificar:



Només un administrador ens podrà reactivar l'usuari. O ens haurem de crear un de nou. Tot i així ara per ara l'aplicació *Android* no te en compte que l'usuari hagi sigut desactivat per evitar impactar a l'usuari d'un ús incorrecte dels comptes d'administrador.

Si fem clic en l'acordió del inventari obtenim tota la informació relativa a les seves existències.



Tornem a obtenir les mateixes funcionalitats disponibles. És a dir, en aquest cas tenint el perfil de client només podrà modificar la quantitat tal i com es va establir en el disseny de l'aplicació: