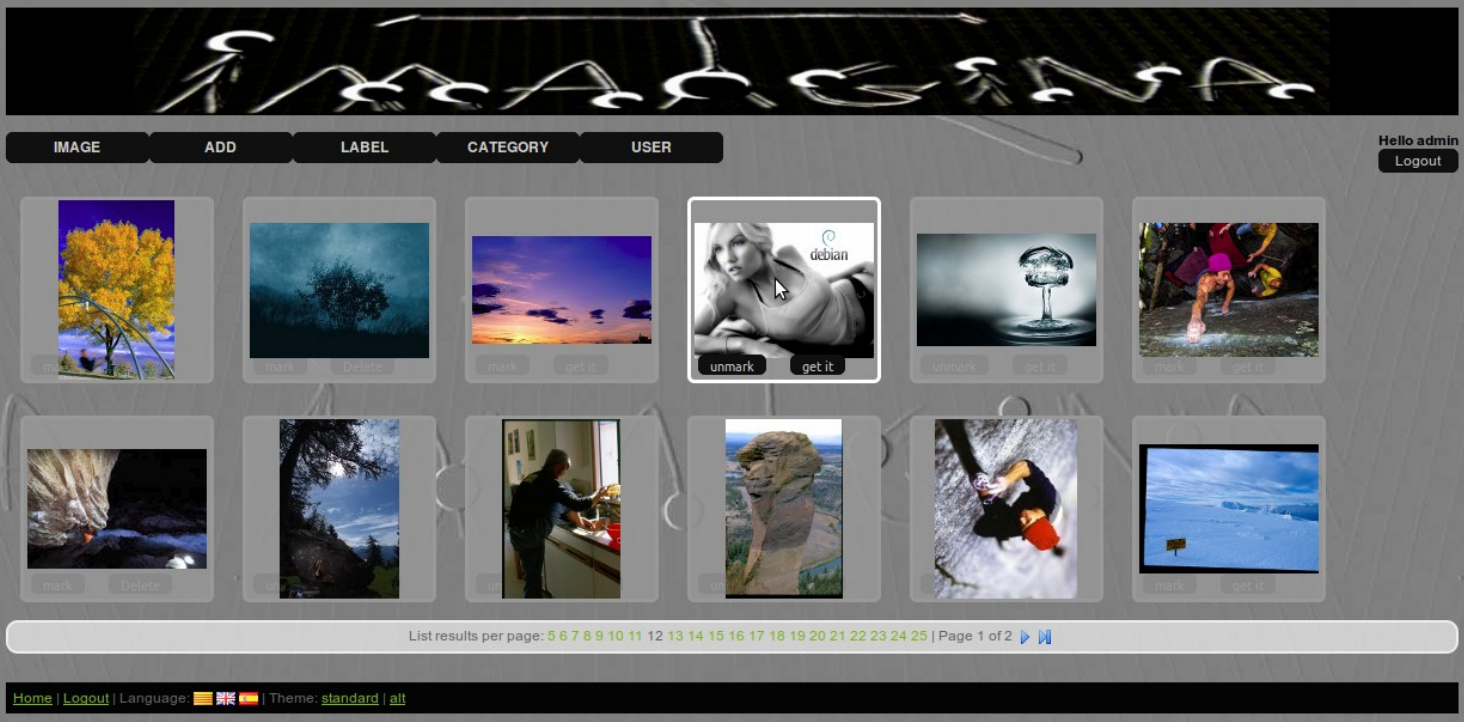
IMATGINA

Un món d'imatges a la teva disposició.



www.imatgina.cat

Assignatura:	**TFC – J2EE. UOC.**
Treball:	**Memòria.**
Alumne:	**Raúl Saúco.**
Consultor:	**Vicenç Font Sagrista**

1 Taula de continguts

2 Índex d'il·lustracions.

3 Agraïments.

Émilie,

pour la multitude d'heures perdues a la recherche d'une connexion a Internet, la centaine de sessions d'acroyoga qu'ont jamais arrive a être, toutes ces soirées tout seule tout i que j'étais a cotée toi, puis mes perdues de patience quand ça marchais pas avec les études, mon amour, je suis dessolée, je peux pas te retourner tout ça, mais au moins je peux te remercier.

Pour tout ça et plus, je t'aime cocote.

Mama,

puede ser no lo sabes pero cada vez que me proponía llamarte me pasaba, como mínimo, una semana, entre PACs i prácticas hasta que lo conseguia, lo siento mucho! Intentaré llamarte más en el futuro, y vernos un poco más a menudo también. Grácias por la inspiración, creo que ya lo sabes pero fuiste tu, volviendo a los estudios, quien me mostró que se podía hacer y eso me animó a intentar.

Ánimo, lo que cuenta no és la llegada sinó el camino!

A la comunitat UOC,

per uns anys d'aprenentatge molt enriquidors, el suport i la sensació de formar part d'una comunitat viva i dinàmica, moltes gràcies a tots. La nostra relació no ha fet més que començar!

Als meus consultors,

per sempre estar allà, per explicar amb paciència tots aquests temes que no acabem de comprendre, en general, per compartir els vostres coneixements i ajudar-nos a aprendre, moltes gràcies a tots!

4 Introducció.

Aquest document detalla el treball portat a terme en el desenvolupament de totes les etapes de l'aplicació web *IMATGINA*.

La dita aplicació té com objectiu servir de punt de contacte entre les entitats que vulguin comercialitzar contingut visual estàtic, com, per exemple, fotografies o il·lustracions, i altres entitats que vulguin adquirir drets d'utilització sobre aquells continguts, prioritzant el comerç entre entitats catalanes tot i intentar donar un bon suport per a usuaris no catalanoparlants.

En el context actual de crisi, les organitzacions es troben cada dia més conscienciades de la necessitat de donar suport a l'economia local, ja que és molt probable que aquesta economia, en major o menor mesura, les hi torni els valors investits oferint a l'organització suport, ja sigui humà, econòmic o social, en el desenvolupament de les seves tasques.

La motivació, o idea principal, que va evolucionar fins a convertir-se en Imatgina va venir donada pel meu contacte amb persones que podrien considerar-se els potencials usuaris futurs d'aquesta, com il·lustradors i fotògrafs, davant la seva manifestació de descontent amb l'actual estat del mercat per a productors d'imatges del nostre país.

Una queixa comú és que les empreses nacionals comprin producte importat, anant-hi directament als proveïdors de referència internacional, com *Corbis, Getty images*, o *Shutterstock*, sense ni tans sols plantejar-se cercar productes locals que puguin satisfer les seves necessitats.

Davant aquestes queixes, les organitzacions responen que no existeix una alternativa equiparable, en nivell d'agilitat i comoditat a aquests proveïdors a nivell nacional i que per a ells representaria una despesa extra de recursos donar suport al comerç local.

Aquest és l'espai que Imatgina aspira a omplir, servint com a pont entre aquests dos col·lectius, oferint funcionalitats per a comercialitzar imatges i publicar anuncis relacionats amb el món de les imatges.

5 Definició i planificació del projecte.

La primera etapa del projecte ha consistit en la demarcació dels objectius a atenir i la definició d'una planificació rigorosa de les dades de compleció de les diverses fases en les quals consta el desenvolupament.

5.1 Objectius proposats.

Inicialment es va determinar que l'aplicació hauria de oferir un cert nombre de serveis tan a productors de contingut visual estàtic, com fotògrafs o il·lustradors, com a consumidors d'aquest, com poden ser editorials, organitzacions a la cerca d'un logotip o una imatge per a la seva propera campanya publicitària o, fins i tot, un professional que pugui portar a terme una determinada tasca requerida per l'organització.

En la primera etapa de treball es van definir **dos tipus d'objectius**, **bàsics,** els quals s'han d'atenir per a considerar que l'aplicació és funcional, i **extres,** com poden ser millores en la implementació interna de la funcionalitat que dóna suport als casos d'ús, o funcionalitats les quals en el seu desenvolupament m'haurien de permetre experimentar amb noves tecnologies, més enllà de l'àmbit considerat bàsic en el desenvolupament del treball final de carrera.

Aquests objectius van anar evolucionant progressivament durant tota la fase de desenvolupament en funció de factors tals com:

- Descobriment de deficiències en el disseny o una millor manera d'oferir la mateixa funcionalitat als usuaris finals.
- La seva innecessarietat.
- La fusió de diverses funcionalitat, les quals inicialment es consideraven independents, en un únic cas d'ús.

De la mateixa manera, també, al llarg del desenvolupament van anar apareixent nous requeriments que s'han hagut d'integrar en el producte, dotant al procés d'un caràcter iteratiu i incremental.

Respecte a aquest punt, no hem d'oblidar que un dels principals objectius del treball final de carrera és que l'estudiant guanye experiència amb un cert seguit de tecnologies que les resulten desconegudes. En el meu cas, salvant-hi algunes pàgines llegides anteriorment, aquest és el primer contacte amb les tecnologies J2EE que he emprat al llarg del semestre.

He de dir que si hagués de portar a terme la mateixa tasca una segona vegada, amb l'experiència guanyada, ho podria fer d'una manera molt més eficient, impactant d'una manera especialment significant la meva experiència les fases d'anàlisi i disseny.

5.1.1 Objectius bàsics.

Aquests que s'han de portar a terme per tal de considerar l'aplicació funcional.

- Registre de nous usuaris.
- Creació de nous continguts visuals, en la terminologia de l'aplicació *imatges*.
- Personalització dels continguts creats; els usuaris haurien de poder anomenar, descriure i fins i tot, possiblement etiquetar els seus continguts visuals amb unes certes etiquetes predefinides creades per administrador/s del lloc web.
- Visualització de continguts visuals creats per altres usuaris, així com facilitats de cerca d'aquests, per exemple per contingut, etiquetes o paraules clau.
- Els continguts (imatges) portaran associats uns comentaris que es podran visualitzar de manera opcional (per defecte quedaran ocults). Aquests comentaris es visualitzaran sempre en ordre cronològic.[1]
- L'administrador podrà posar-se en contacte tan amb els autors com amb els consumidors per a ferlis un comentari. També podrà eliminar continguts (per considerar-los inapropiats), aquests continguts no s'eliminaran directament de la BD sinó que quedaran ocults per tal de que l'autor pugui reclamar i es puguin restablir en cas d'error d'eliminació.
- L'administrador també podrà eliminar comentaris de la BD i en aquest cas l'eliminació serà definitiva.[2]

5.1.2 Objectius extres.

Aquests que es poden considerar com proves amb idees interessants o experiments per a aprendre noves tecnologies.

1 Aquesta funcionalitat, tot i que es va proposar inicialment, va quedar eliminada del projecte en la fase d'anàlisi per considerar que no donava servei a una necessitat real dels usuaris finals.

2 Aquesta funcionalitat es va descarregar, en la fase d'anàlisi, sobre el sistema, alliberant als administradors d'una tasca que pot ser perfectament automatitzada en funció de l'espai lliure i de certs paràmetres definits prèviament.

- Quan un client no trobi un producte que respongui satisfactòriament als seus criteris de cerca, se li oferirà l'opció de rebre un correu en si en un moment posterior s'afegeix contingut que li pugui interessar (que respongui als criteris de cerca).

  Aquesta funcionalitat, desafortunadament no es va poder portar a terme donada la seva complexitat i la meva manca de temps.

  Encara la considero interessant i es tracta, molt possiblement, d'una funcionalitat que incorporaré més endavant.

- Si un comprador està registrat, també podem oferir un servei de marcadors, per exemple si troba una imatge interessant sobre la qual vol tornar més tard.

  Aquesta funcionalitat, per contra, va quedar, des de la fase d'anàlisi, definida com un dels requeriments més importants de l'aplicació i s'ha desenvolupat completament.

- Als autors se les hi pot oferir la possibilitat de oferir una versió *personalitzada* de la seva *galeria*, i de administrar els seus continguts, haurien de ser capaços tan d'afegir, com d'eliminar com de definir prioritats.

  Aquesta funcionalitat queda, de moment, completament descartada. Al llarg de la fase de desenvolupament de l'aplicació he constatat un punt que ja intuïa anteriorment, la meva completa falta de talent per al disseny gràfic, amb el qual tota la part visual de la interfície d'usuari ha quedat definida de manera *molt* provisional.

  El següent pas seria contractar els serveis d'un professional del disseny gràfic i deixar aquesta responsabilitat en les seves mans. Una vegada obtingut un disseny acceptable per defecte, es podria procedir a determinar les parts de la interfície d'usuari que serien adaptables i crear un cert nombre de dissenys opcionals per a aquestes.

  La seva incorporació a l'aplicació es podria portar a terme de manera senzilla gràcies al suport per a temes que ja ha quedat desenvolupat completament. Tot i això, considero aquest punt com de molt baixa prioritat.

5.2 Planificació.

5.2.1 Planificació, una visió general.

Degut al meu total desconeixement de l'extensió real de la tasca a portar a terme, les tecnologies a emprar per a fer-lo, el cost d'aprendre a utilitzar aquestes tecnologies i, en general, la manera de portar a terme un projecte d'aquesta envergadura, la planificació inicial del projecte va ser una de les etapes que més em va costar.

Des d'una certa perspectiva, aquest és l'únic punt novell d'aquesta assignatura; en totes les altres assignatures de la carrera, la planificació temporal queda perfectament definida en el pla d'estudi i l'únic que l'estudiant ha de fer és seguir les recomanacions d'aquest.

Des de la perspectiva que m'aporta trobar-me a finals del semestre, puc afirmar que totes les altres tasques portades a terme durant el desenvolupament del TFC es poden relacionar, de manera més o menys directa, amb alguna de les matèries estudiades durant la carrera.

La recollida i anàlisi de requisits s'estudien en profunditat, i es practiquen, en Enginyeria del programari orientat a objectes, la programació es veu en nombroses matèries, començant en Fonaments de la programació, passant per Programació orientada a objectes i derivant en nombroses branques com Estructura de la informació, Ampliació de sistemes operatius..., l'estudi del disseny i implementació de

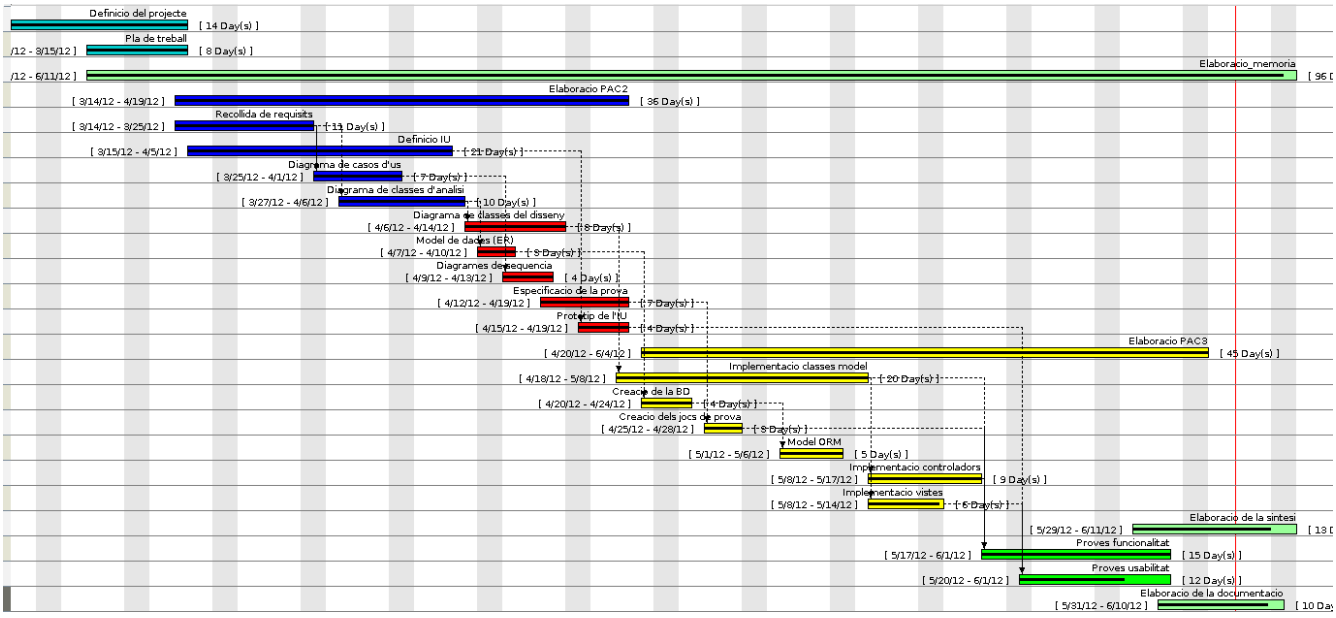
bases de dades es veu en les tres assignatures d'aquesta branca, Bases de dades I i II i Sistemes de gestió de bases de dades, fins i tot el disseny d'interfícies d'usuari s'estudia en Interacció humana amb els ordinadors. La planificació, però, és un tema que, fins que no arribem a aquesta assignatura, ens ve bastant demarcat pels respectius plans d'estudi i calendaris semestrals, amb el qual presenta el al·licient de la novetat.

Tot i la meva prèvia inexperiència, considero la planificació del semestre que vaig definir satisfactòria. Amb l'ajuda del pla d'estudi, es van definir quatre blocs de treball principal.

- Un primer bloc, amb termini 14 de Mars, en el qual s'havia de definir l'àmbit del projecte i la temporització de les tasques específiques a portar a terme. Corresponia a la PAC1.
- Un segon bloc, amb termini el 19 d'Abril, en el qual s'havia de portar a terme les fases de recollida i anàlisi de requisits així com el disseny de l'aplicació. Corresponia a la PAC2.
- Un tercer bloc, amb termini el 4 de Juny, en el qual s'havia de desenvolupar la major part de les funcionalitats especificades en el disseny (de manera orientativa es mencionava un 80%). corresponia a la PAC3.
- Un bloc final, amb termini el 18 de Juny, en el qual s'ha de terminar el desenvolupament de l'aplicació i s'han d'elaborar els documents de memòria i síntesi del treball portat a terme.

5.2.2 Detall de la planificació.

El detall de la planificació es pot obtenir en el document Pla de treball que es va enviar com primera PAC, seria redundant tornar a incloure totes les explicacions aquí i no aportaria res, inclourem però el diagrama de Gantt que s'ha utilitzat durant tot el semestre, el qual permet, de manera molt visual, fer-se una idea de la planificació dels objectius específics.



Il·lustració 1: Diagrama de Gantt de la planificació del projecte.

5.2.3 Conclusions.

Com en altres assignatures de la carrera, considero que una bona planificació és indispensable per tal de treure'n el màxim profit i assimilar els continguts d'una manera correcta, en aquest cas en particular, comptem amb l'al·licient de que és el mateix alumne l'encarregat de desenvolupar aquesta planificació, amb totes les conseqüències que això comporta durant la resta del semestre.

Considero que puc estar satisfet amb la planificació elaborada i la manera de portar-la a terme durant el semestre, si bé és veritat que per tal de portar tots els aspectes del desenvolupament en bon ordre i seguir la planificació he hagut de dedicar moltes més hores de les que pensava en un principi, això no és res de nou per a un estudiant de la UOC, i suposo que en general per a qualsevol estudiant, sobretot en el nostre camp de la informàtica, on qualsevol tasca pren sempre molta més dedicació de la que sembla en un principi.

6 Recollida i anàlisi dels requisits.

Per a desenvolupar el document de l'anàlisi vaig emprar la metodologia que s'estudia en l'assignatura Enginyeria del programari orientat a objectes de la UOC i que es recull en els documents [IEPOO04], [RIDR04], i [AOO04], segons aquesta, dividirem la fase d'anàlisi de requisit en quatre etapes.[3]

6.1 Etapa de recollida de requisits.

Estudiem el domini i les necessitats dels futurs usuaris, establim quins són els actors que interactuaran amb el SI. Obtenim el model del negoci, el del domini i una descripció informal del casos d'ús.

Es van determinar les necessitats que hauria de cobrir el programari, notablement permetre als usuaris publicar contingut visual, visualitzar i cercar continguts publicats per altres usuaris i comunicar-se mitjançant anuncis. També es van perfilar els actors que s'haurien de considerar, es van definir quatre actors amb la següent descripció:

Actor visitant, podrà registrar-se per tal de passar a ser un usuari registrat, és a dir, les seves dades es trobaran en el sistema i se li oferirà accés més funcionalitats que a un visitant.

- Podrà autenticar-se (si ja es troba registrat) per tal de tenir accés a la funcionalitat reservada als usuaris registrats.
- Podrà visualitzar imatges i decidir de comprar alguna, per tal de fer-lo s'haurà de registrar 1.
- Podrà realitzar cerques sobre la base de dades d'imatges mitjançant paraules clau.
- Podrà realitzar cerques sobre la base de dades d'anuncis i visualitzar-los.

Actor usuari registrat, podrà fer totes les mateixes accions que un visitant menys registrar-se (o autentificar-se) i, a més, les següents:

- Podrà publicar imatges (comercialitzar-les) a l'aplicació.
- Podrà marcar imatges com a interessants per a visualitzar-les més tard.
- Podrà eliminar imatges que hagi publicat anteriorment o restaurar d'eliminades.
- Podrà publicar anuncis que podran ser visualitzats per altres usuaris o visitants.
- Podrà marcar anuncis per a tornar a llegir-los en un moment posterior.
- Podrà eliminar anuncis que hagi publicat anteriorment o restaurar d'eliminats.
- Podrà comprar imatges.

3 Per tal d'obtenir informació més detallada sobre les tasques que es van portar a terme en aquesta fase es pot consultar el document *Fase d'anàlisi i disseny de l'aplicació Imatgina* que es va enviar com a part de la segona prova d'avaluació.

Actor administrador, podrà fer totes les mateixes accions que un usuari registrat i, a més:

- Crear etiquetes.
- Eliminar un anunci que no les-hi pertany.
- Eliminar una imatge que no les-hi pertany.
- Donar de baixa un soci.

Actor sistema, es considera aquest autor per a descarregar als administradors de tasques que poden portar-se a terme de manera automàtica i per a oferir als usuaris d'una funcionalitat de reciclatge que els permeti recuperar elements que han marcat com eliminats prèviament. L'únic cas d'ús en el que, inicialment es pensava, està involucrat aquest actor és la neteja de dades del sistema en cas de superar-se uns límits preestablerts.

6.2 Descripció de la interfície d'usuari.

Amb la informació recollida en la fase anterior, elaborem una descripció formal de la interacció que tindrà lloc entre el SI i els seus usuaris finals.

Els principals productes d'aquesta etapa van ser els següents:

- Identificació de les restriccions tècniques; un dels principals avantatges d'una aplicació web és que els usuaris poden accedir amb el dispositiu client de la seva preferència, aquest fet, però, presenta l'inconvenient de que en el disseny s'ha de tindre en consideració la gran diversitat que poden presentar aquests dispositius.

  Afortunadament avui les tecnologies que tenim al nostre abast per a portar a terme la construcció d'una aplicació web ens permeten una clara separació entre la funcionalitat i la presentació. Aquest és el cas, definitivament amb les tecnologies J2EE. A més, existeixen una gran quantitat de bastiments i eines dissenyades amb l'única finalitat de permetre al programador abstreure'ns de les particularitats de tots aquests dispositius d'accés.

  Com a conclusió podem dir que s'ha de tenir en compta la diversitat funcional i tecnològica dels usuaris de l'aplicació però que contem amb una bona quantitat d'eines per a ajudar-nos en aquesta tasca.

- **Perfils d'usuari.** No podrem fer suposicions sobre els coneixements tecnològics dels usuaris de l'aplicació i haurem d'esforçar-nos per a oferir un nivell molt bo d'usabilitat ja que aquest influirà notablement en la captació de nous clients i el manteniment dels existents.

  Per altra banda, tindrem un grup molt reduït d'usuaris administradors i, per a ells, la interacció amb Imatgina es tractarà d'una activitat remunerada, per tant se les-hi poden imposar uns certs requeriments, com que siguin capaços d'interactuar amb la base de dades mitjançant una eina específica. Tot i això, s'ha de tenir en consideració que si facilitem la seva tasca seran més productius i haurem de dissenyar amb aquest punt en consideració.

- **Documentació de les tasques.** Portar a terme una documentació completa de les tasques en aquest document seria una redundància innecessària[4], donat el caràcter visual del producte i la seva bona adaptació al medi digital, es pot adquirir una bona idea de les tasques d'usuari estudiant els casos d'ús que es presentaran a l'apartat següent.

- **Especificacions d'usabilitat.** Els 90% dels usuaris finals no administrador han de ser capaços de

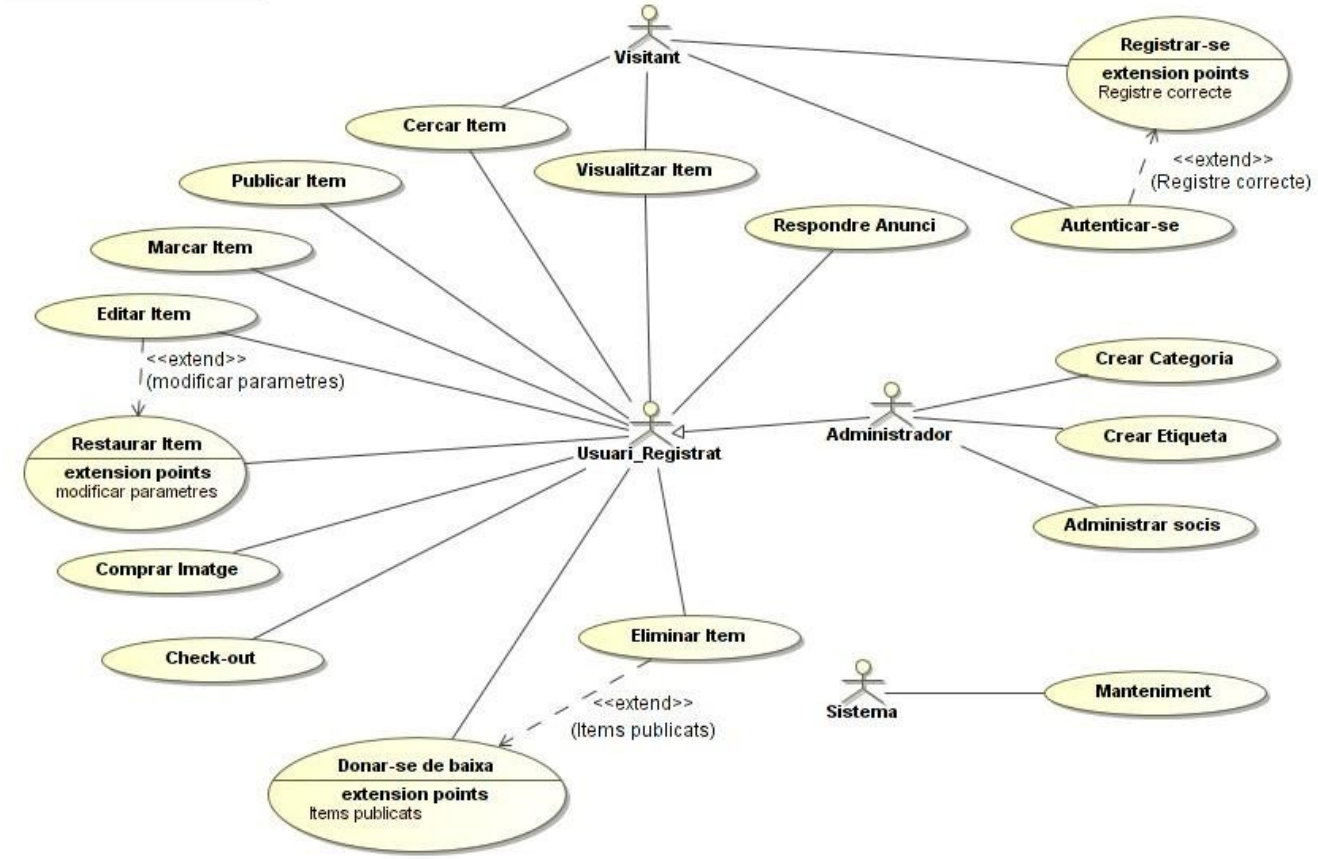
4 A qui pugui interessar una descripció detallada de les tasques d'usuari, en presentació tabular acompanyades d'una descripció textual, es pot consultar el document *Fase d'anàlisi i disseny de l'aplicació Imatgina* que es va enviar com a part de la segona prova d'avaluació.

fer sevir tota la funcionalitat de l'aplicació la primera vegada que l'utilitzen sense necessitar cap document d'ajuda. Un usuari hauria de poder publicar un ítem, la primera vegada que ho fa, en menys de 2 minuts o realitzar una compra, també la primera vegada, en menys de 5 minuts.

Els administradors han de poder utilitzar tota la funcionalitat del sistema després de dues hores de formació.

6.3 Descripció formal dels casos d'ús.

Utilitzarem tota la documentació generada fins a aquest punt per a documentar els casos d'ús del sistema, els casos que vam obtenir en la fase d'anàlisi es poden veure reflectits en el següent diagrama.



Il·lustració 2: Diagrama de casos d'ús de l'anàlisi.

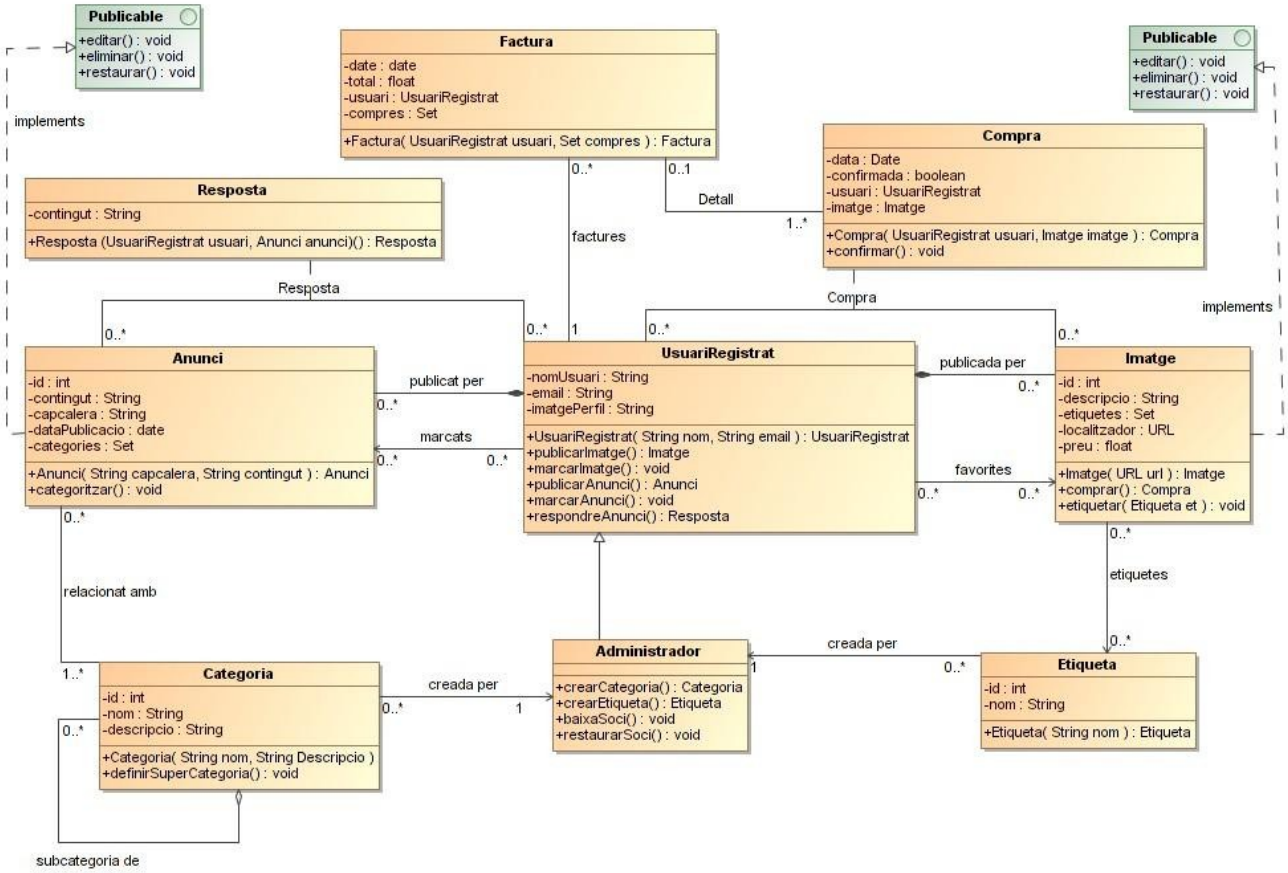
Per brevetat, i considerant que aquest és un aspecte que es veu sotmès a constants adaptacions i modificacions durant tot el procés de desenvolupament d'una aplicació, no inclouré en aquest document la descripció textual dels casos d'ús[5].

5 Es pot trobar aquesta descripció textual, elaborada durant la fase d'anàlisi, al document *Fase d'anàlisi i disseny de l'aplicació Imatgina* que es va enviar com a part de la segona prova d'avaluació.

6.4 Creació del diagrama de classes de l'anàlisi.

Definim les classes que existiran en la nostra aplicació i les seves interrelacions de manera estàtica. Es tracta d'un disseny conceptual, sense tenir en compte les limitacions de cap tecnologia concreta.

Amb tota la informació generada fins a aquest moment, s'han pogut determinar les següents classes, interfícies i interrelacions entre elles.



Il·lustració 3: Diagrama de classes de l'anàlisi.

En aquest diagrama s'identifiquen les classes que modelen els objectes del domini de l'usuari. Tot i que estrictament no hauríem de parlar de cap tecnologia en concret durant la fase d'anàlisi, en el nostre cas particular ja hem decidit prèviament que utilitzarem la tecnologia J2EE per a portar a terme el desenvolupament, aplicant el patró[6] de disseny MVC[7], es pot per tant afirmar que aquestes classes formaran el model i seran les que s'hauran de persistir en la base de dades.

A més haurem de implementar classes controladores i vistes per a presentar els resultats a l'usuari, la forma d'implementar unes i altres dependrà de les tecnologies emprades i serà un punt que estudiarem en l'apartat següent.

6 Segons alguns autors es tracta d'un agregat de patrons i no un patró. Enllaç.
7 Podem trobar una descripció en detall del patró MVC a [POSA96].

7 Disseny.

Per a portar a terme el disseny de l'aplicació he utilitzat la metodologia estudiada en l'assignatura Enginyeria del programari orientat a objectes, la qual es pot consultar en [DOO04].

Enfocarem el disseny de l'aplicació des de sis vessants diferents:

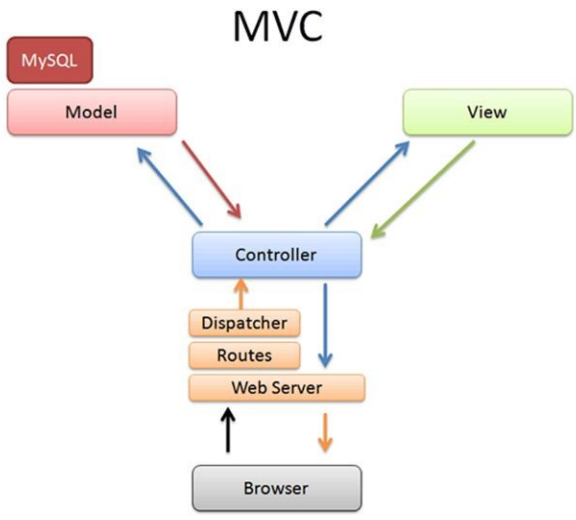
1. Disseny arquitectònic.
2. Disseny dels casos d'ús.
3. Disseny del model de l'aplicació (diagrama de classes del disseny).
4. Disseny de la persistència.
5. Disseny de la interfície gràfica d'usuari.
6. Disseny de la prova.

A continuació veurem cada un d'aquest components amb més detall.

7.1 Disseny arquitectònic.

En aquest apartat hem determinat que construirem l'aplicació seguint el model definit pel patró MVC[8], es a dir, tindrem una separació de la funcionalitat en tres capes:

- Una capa de model, encarregada de representar els objectes del domini de l'usuari. En aquesta capa trobarem les dades i la funcionalitat necessària per a operar amb aquestes dades. Els objectes d'aquesta capa normalment requeriran ser persistits en una base de dades.
- Una capa de control, encarregada de rebre les instruccions de l'usuari i processar-les, si aquestes instruccions ho requereixen la capa de control interactuarà amb la capa de model per tal de recuperar dades, crear noves dades o modificar alguna d'existent. Finalment, els objectes d'aquesta capa seran els responsables de seleccionar la vista que s'haurà de mostrar a continuació a l'usuari i de proporcionar a aquesta vista la informació necessària.



Il·lustració 4: Diagrama del patró MVC.

- Una capa de vista, encarregada de presentar informació a l'usuari. En una aplicació ben dissenyada aquesta capa no hauria de contenir cap lògica d'aplicació.

Una vegada hem definit l'estructura de l'aplicació ens cal **determinar quines eines s'hauran d'utilitzar per a portar a terme la construcció**.

D'entre totes les tecnologies actuals que ens ofereix *java*, en el marc actual, per a desenvolupar aplicacions J2EE hi ha tres que es poden considerar predominants.

- La implementació oficial de l'especificació J2EE, és a dir les EJB[9], les quals ,actualment en la seva

8 Es pot trobar una bona descripció del patró MVC a [DP94] pags 14-16.
9 Es pot trobar l'especificació EJB3 a http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html.

versió 3, semblen haver superat algunes deficiències anteriors i és una tecnologia digne de consideració, sobretot considerant la seva condició d'estàndard oficial.

- El bastiment *Struts*[10], concretament la versió 2, el qual és considerat per molts autors com la implementació del patró MVC.
- El bastiment *Spring*, el qual sembla ser l'estàndard *de facto* en el desenvolupament de aplicacions J2EE i **per el qual m'he decantat**.

Els motius d'aquesta elecció són diversos i variats però puc mencionar la utilització de l'orientació a aspectes que fa el bastiment *Spring* com un aspecte que m'ha impressionat positivament comparant-ho amb algunes de les opcions.

La raó és que veig l'orientació a aspectes com una solució general a problemes comuns en el desenvolupament d'aplicacions empresarials, i en general en la programació, en contra d'altres solucions alternatives als mateixos problemes, molt més lligades a una plataforma en concret i molt més difícils d'exportar entre plataformes diferents.

Un motiu de pes és l'existència de l'eina *SpringRoo*[11] la qual s'ha usat extensivament en el desenvolupament del projecte, especialment en les fases inicials. I la qual considero molt positivament com una eina d'aprenentatge i prototipatge.

Per a portar a facilitar l'administració de la **capa de persistència** s'ha triat la tecnologia *Hibernate*[12], molt reconeguda i utilitzada extensivament. Els motius principals d'aquesta elecció han sigut la seva reputació, una bona integració amb el bastiment *Spring* sense necessitat de massa configuració i l'existència de molt bona documentació.

Una altra preocupació que s'ha de tenir en compta en el desenvolupament d'aplicacions J2EE és el control de les dependències, les quals poden atenir mides monumentals fins i tot per a aplicacions senzilles. Per a facilitar aquesta tasca he decidit d'utilitzar l'eina *Maven*[13].

Finalment, d'entre totes les bones opcions existents com a gestors de bases de dades de codi obert, he triat *MySQL*[14] per la seva popularitat i existència de bona documentació.

Per a fer l'aplicació públicament accessible necessitarem contractar els serveis d'algun proveïdor d'hostatge web d'entre els que ofereixen contenidors d'aplicacions *Java.* En la fase de desenvolupament he utilitzat un servidor Tomcat7.

7.2 Disseny dels casos d'ús.

Amb la base de la documentació textual sobre els casos d'ús que es va desenvolupar en la fase d'anàlisi, en el disseny s'ha portat a terme la seva especificació en forma de diagrames de seqüència UML.

Tot i que aclaridors a l'hora d'entendre el treball fet i alguna de les decisions preses, he optat per no incloure aquests diagrames en el present document ja que es poden consultar en el document *Fase d'anàlisi i disseny de l'aplicació Imatgina* que es va enviar com a part de la segona prova d'avaluació.

A continuació detallem els casos d'ús que es van trobar en el disseny, cadascú acompanyat d'una breu descripció i de la referència als apartats d'aquest document que expliquen com s'ha portat a terme la seva implementació.

10 La pàgina oficial es troba a http://struts.apache.org/
11 Es pot trobar més informació sobre aquesta eina a http://www.springsource.org/spring-roo
12 La pàgina oficial és http://www.hibernate.org/
13 La pàgina oficial és http://maven.apache.org/
14 La pàgina oficial és http://www.mysql.com/

Cas d'ús número 1: “Registrar-se”.

Permet als visitants del lloc web crear una compta d'usuari, hauran de facilitar algunes dades que quedaran registrades.

Apartat 8.7.1.1 registre de nous usuaris.

Cas d'ús número 2: “Donar-se de baixa”.

Permet als usuaris eliminar les seves dades del sistema, a partir d'aquest moment perden la capacitat d'autenticar-se utilitzant les dades associades a aquesta compta d'usuari.

Apartat 8.7.1.2 Baixa d'un usuari.

Cas d'ús número 3: “Autenticar-se”.

Un usuari utilitza el seu *nom d'usuari* i *contrasenya* per a identificar-se davant el sistema.

Apartat 8.7.1.3 Autentificació d'usuaris.

Cas d'ús número 4: “Cercar ítem”.

Un usuari de l'aplicació indica certs criteris per tal de que se li mostrin ítems relacionats amb aquests.

Apartats 8.7.4.5 Cerca d'imatges segons una paraula clau. i 8.7.6.5 Cerca d'entitats anunci.

Cas d'ús número 5: “Visualitzar ítem”.

Un usuari centra la seva atenció en un *ítem* en particular. Es centra el focus d'atenció en aquest *ítem* i s'ofereixen enllaços per a portar a terme les accions possibles.

Apartats 8.7.4.3 Visualització d'una entitat imatge. i 8.7.6.3 Visualització d'entitats anunci.

Cas d'ús número 6: “Publicar ítem”.

Un **Usuari registrat** manifesta la seva intenció de *publicar* un *ítem.* El sistema recull la informació associada i el nou *ítem* queda inserit en la base de dades i accessible per a tots els usuaris.

Apartats 8.7.4.1 Creació d'entitats imatge. i 8.7.6.1 Creació d'entitats Anunci.

Cas d'ús número 7: “Editar ítem”.

Un usuari vol modificar alguns dels paràmetres relacionats amb algun dels *ítems* que ha publicat anteriorment.

Apartats 8.7.4.2 Edició d'entitats imatge. i 8.7.4.11 Eliminació i restauració d'entitats imatge.

Cas d'ús número 8: “Marcar ítem”.

Un **usuari registrat** marca un *ítem* per facilitar la seva posterior localització.

Apartats 8.7.4.9 Marcatge d'imatges. i 8.7.6.6 Marcatge d'entitats anunci.

Cas d'ús número 9: “Eliminar ítem”.

Un **usuari registrat** marca un ítem publicat per ell mateix per a la seva eliminació del sistema.

Apartats 8.7.4.11 Eliminació i restauració d'entitats imatge. i 8.7.6.3 Visualització d'entitats anunci.

Cas d'ús número 10: "Restaurar ítem".

Un usuari restaura una imatge que havia eliminat anteriorment.

Apartats Error: Reference source not foundError: Reference source not found i 8.7.6.3 Visualització d'entitats anunci.

Cas d'ús número 11: "Comprar imatge".

Un usuari selecciona l'acció compra sobre una *imatge* determinada. Aquesta queda inclosa dins la cistella de la compra associada a l'usuari.

Apartats 8.7.5.1 Creació de noves compres. i 8.8 Funcionalitat de compra d'imatges.

Cas d'ús número 12: "*Check-out*".

Un usuari sol·licita l'acció de realitzar el pagament dels objecte que té a la seva *cistella.*

Apartat 8.8.1 Procediment de checkout.

Cas d'ús número 13: "Respondre anunci".

Un usuari envia una resposta a un anunci publicat per un altre usuari.

Apartat 8.7.8.1 Creació d'entitats resposta.

Cas d'ús número 14: "Crear Etiqueta".

Un usuari administrador definix una nova *etiqueta* en el sistema.

Apartat 8.7.3.1 Creació d'entitats etiqueta.

Cas d'ús número 15: "Crear categoria".

Un usuari defineix una nova categoria que podrà ser utilitzada per a categoritzar anuncis.

Apartat 8.7.2.1 Creació d'entitats categoria.

Cas d'ús número 16: "Administrar socis".

Un administrador podrà gestionar altes, baixes i incidències de socis, així com enviar-les correus o respondre als seus.

Aquesta funcionalitat es porta a terme de manera no incorporada en l'aplicació, l'administrador ha de portar a terme les tasques manualment.

Cas d'ús número 17: "Manteniment".

Resum de la funcionalitat: El sistema porta a terme, automàticament, funcions de manteniment.

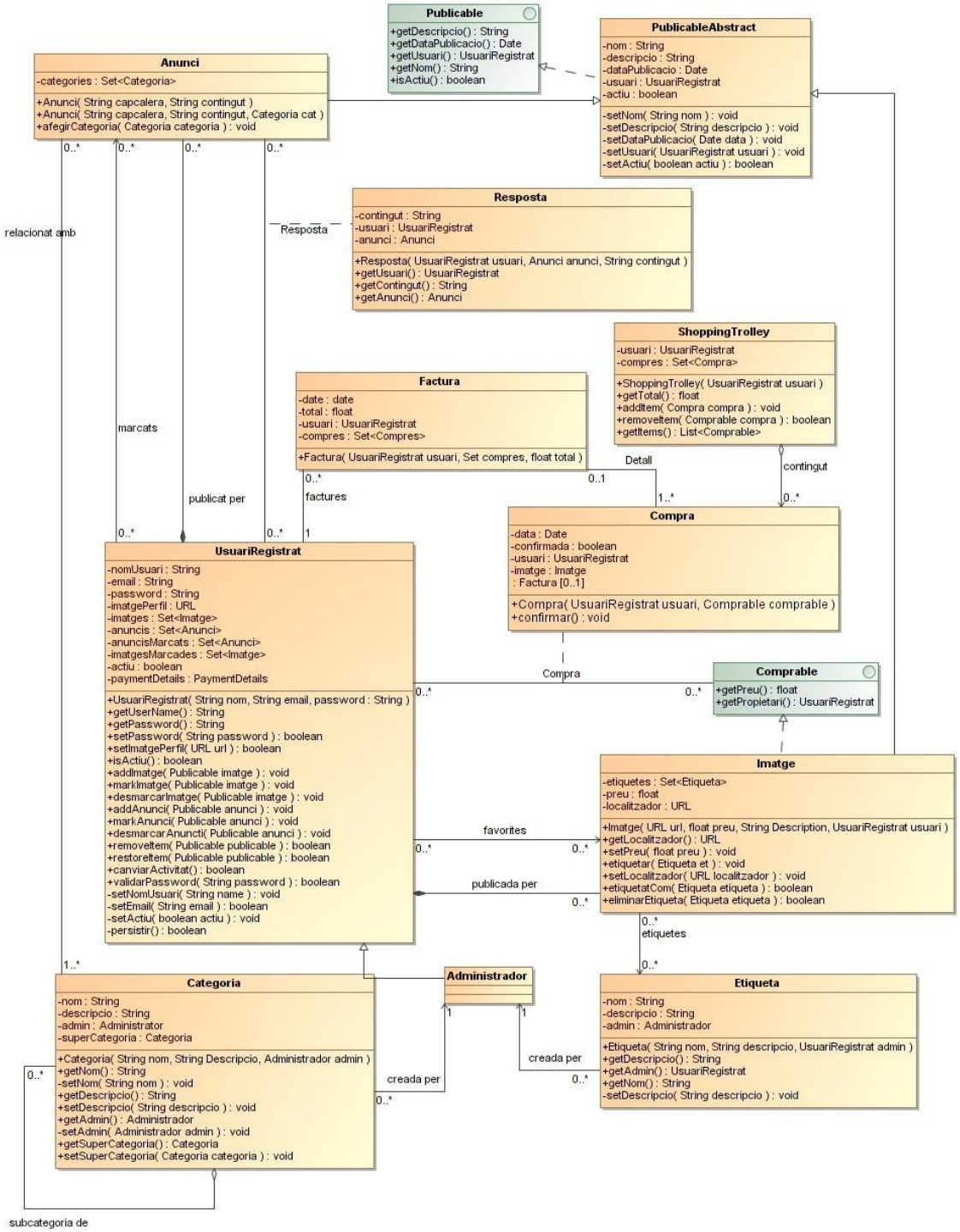
Aquesta funcionalitat s'ha de programar fora de l'aplicació, per exemple mitjançant un procediment

desat en la base de dades o un altre programa que s'ocupi d'aquesta tasca.

7.3 Disseny del model de l'aplicació (diagrama de classes del disseny).

Per a elaborar el diagrama de classes del disseny s'han seguit els passos especificats a [DOO04]. Concretament, normalització dels noms, reutilització de les classes i adaptació de l'herència.

El diagrama obtingut és el següent.



Il·lustració 5: Diagrama de classes del disseny.

7.4 Disseny de la persistència.

En el document *Fase d'anàlisi i disseny de l'aplicació Imatgina* que es va enviar com a part de la segona prova d'avaluació s'havia adjuntat un diagrama del model lògic-relacional de la base de dades que utilitzarem per a persistir les dades. Aquest diagrama no serà inclòs en la memòria ja que considero que no serveix per a aclarir cap aspecte relacionat amb el desenvolupament d'aquesta que no quedi aclarit pel diagrama de classes del disseny.

De fet, ni tans sols s'han utilitzat sentències de creació de la base de dades sinó que he aprofitat el mapatge que s'ha elaborat a *Hibernate* de les classes del model per tal que l'aplicació, en el moment de ser desplegada en un contenidor, generi les taules de la base de dades sinó les troba a la base de dades.

7.5 Disseny de la interfície gràfica d'usuari.

Per a modelar la interacció dels usuaris amb l'aplicació es va portar a terme la construcció d'un prototip amb l'eina *Axure RP*[15]. Aquest prototip va formar part del lliurament corresponent a la PAC2 i consisteix en una versió completament navegable de l'aplicació però que no ofereix cap funcionalitat.

Aquesta metodologia de disseny de la interfície d'usuari la vaig estudiar en l'assignatura *Interacció humana amb els ordinadors,* i considero que ofereix molts bons resultats, ens permet desenvolupar una versió completament interactiva de l'aplicació d'una manera molt eficient.

Aquesta versió de l'aplicació es pot utilitzar per a portar a terme proves d'usabilitat i per a proves de concepte, amb el qual les modificacions que s'hagin de portar a terme sobre el prototip enlloc del producte final redueix el cost de desenvolupament d'una manera notable.

7.6 Disseny de la prova.

Per a l'elaboració de proves aprofitarem, una altra vegada, les possibilitats d'autogeneració de codi que ens ofereix *SpringRoo*[16]. Comprovarem el funcionament de l'aplicació des de tres nivells.

- El nivell més bàsic, *unit testing*, comprovarem el bon funcionament de mètodes aïllats.
- Un nivell mitjà, *integration testing*, comprovem el correcte funcionament de la persistència, creem entitats, les persistim, les modifiquem, les eliminem de la base de dades i comprovem que tots aquests passos donen el resultat esperat.
- Un nivell superior, tests de funcionalitat*,* comprovem que la resposta a les accions d'usuari és l'esperada, d'entre les diverses maneres d'escriure i portar a terme tests de funcionalitat he triat la suite de tests *selenium*[17] pel fet de ser la que sembla millor adaptar-se a les aplicacions web.

A més d'aquests tests de funcionalitat, també serà interessant portar a terme tests sobre d'usabilitat i accessibilitat del lloc web creat. Els tests d'usabilitat s'hauran de portar a terme amb usuaris i, per tant, de moment, queden fora de l'abast d'aquest treball, per contra els tests d'accessibilitat es poden automatitzar amb eines com HERA[18].

15 Es pot trobar més informació sobre aquesta eina a http://www.axure.com/

16 Una bona explicació sobre les classes de prova que *SpringRoo* permet generar de manera automàtica i com adaptar-les a les nostres necessitats es pot trobar a [SRIA12] principalment al capítol 9.

17 Per a més informació sobre *selenium* veure http://seleniumhq.org/

18 Per a obtenir més informació sobre HERA consultar http://www.sidar.org/hera/

8 Elaboració del producte.

8.1 Creació de les entitats del model i definició de la persistència.

A l'hora de crear les entitats del model i la base de dades tenim diverses opcions. La més habitual seria definir les taules de la base de dades mitjançant sentències SQL, i crear una capa DAO per a facilitar l'accés a la base de dades de les entitats de la nostra aplicació.

L'eina roo pren una altra aproximació al mateix problema que pretén reduir la quantitat de línies de codi necessàries per a posar en funcionament la nostra aplicació i, donat el fet que ja l'estem utilitzant, podem donar-les una oportunitat.

Segons la metodologia de roo, utilitzarem la línia de comandes per a crear entitats de manera declarativa. Una vegada declarada una entitat, roo s'ocuparà de gestionar la creació de les estructures necessàries per a la seva persistència en la base de dades i de enviar a la BD les sentències SQL de definició.

Veiem un exemple[19] de creació d'entitats amb el *shell* que ens ofereix roo, crearem l'entitat Imatge la qual representa una imatge que un usuari ha publicat a Imatgina.

```
// Create Imatge
entity jpa --class ~.domain.Imatge --table IMATGE --testAutomatically
field string --fieldName nom --column nom
field string --fieldName description
field date --fieldName dataPublicacio --type java.util.Date --notNull
field reference --fieldName usuari --type ~.domain.Usuari --notNull
field boolean --fieldName actiu --notNull --value true
field set --fieldName etiquetes --type ~.domain.Etiqueta --cardinality MANY_TO_MANY
field number --fieldName preu --type float --notNull --column preu
```

Per defecte l'aplicació generada per roo regenera l'estructura de la base de dades nova cada vegada que la despleguem en un contenidor, per a canviar aquest comportament modifiquem l'arxiu de configuració `/src/main/resources/persistence.xml` de la manera següent.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.0"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
<persistence-unit name="persistenceUnit" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <properties>
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5InnoDBDialect"/>
            <!-- value="create" to build a new database on each run; value="update" to modify an
existing database; value="create-drop" means the same as "create" but also drops tables when
Hibernate closes; value="validate" makes no changes to the database -->
            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.ejb.naming_strategy"
value="org.hibernate.cfg.ImprovedNamingStrategy"/>
            <property name="hibernate.connection.charSet" value="UTF-8"/>
            <!-- Uncomment the following two properties for JBoss only -->
            <!-- property name="hibernate.validator.apply_to_ddl" value="false" /-->
            <!-- property name="hibernate.validator.autoregister_listeners" value="false" /-->
        </properties>
    </persistence-unit>
</persistence>
```

Amb aquest canvi informem que volem mantenir els canvis introduïts, fet que ens permetrà desar un cert nombre de files de cada tipus en la nostra base de dades per tal de fer proves de funcionament.

19 Es pot trobar l'*script* que genera l'aplicació original a: Annex A : Script roo de creació de l'aplicació..

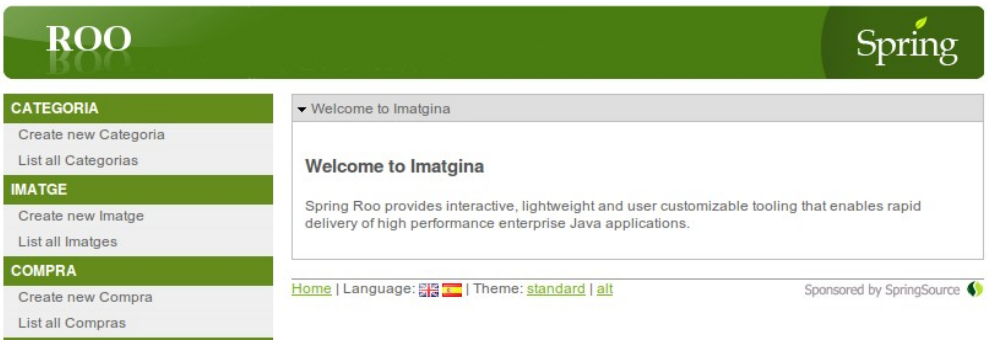
8.2 Modificació de la interfície d'usuari.

La interfície d'usuari constituirà el punt de contacte entre aquests i la nostra aplicació, és a dir, la part visible d'aquesta. Com a tal, li hem de donar una màxima importància ja que no ens servirà de res tenir una implementació dels requeriments perfecta si els usuaris no poden utilitzar la funcionalitat oferta.

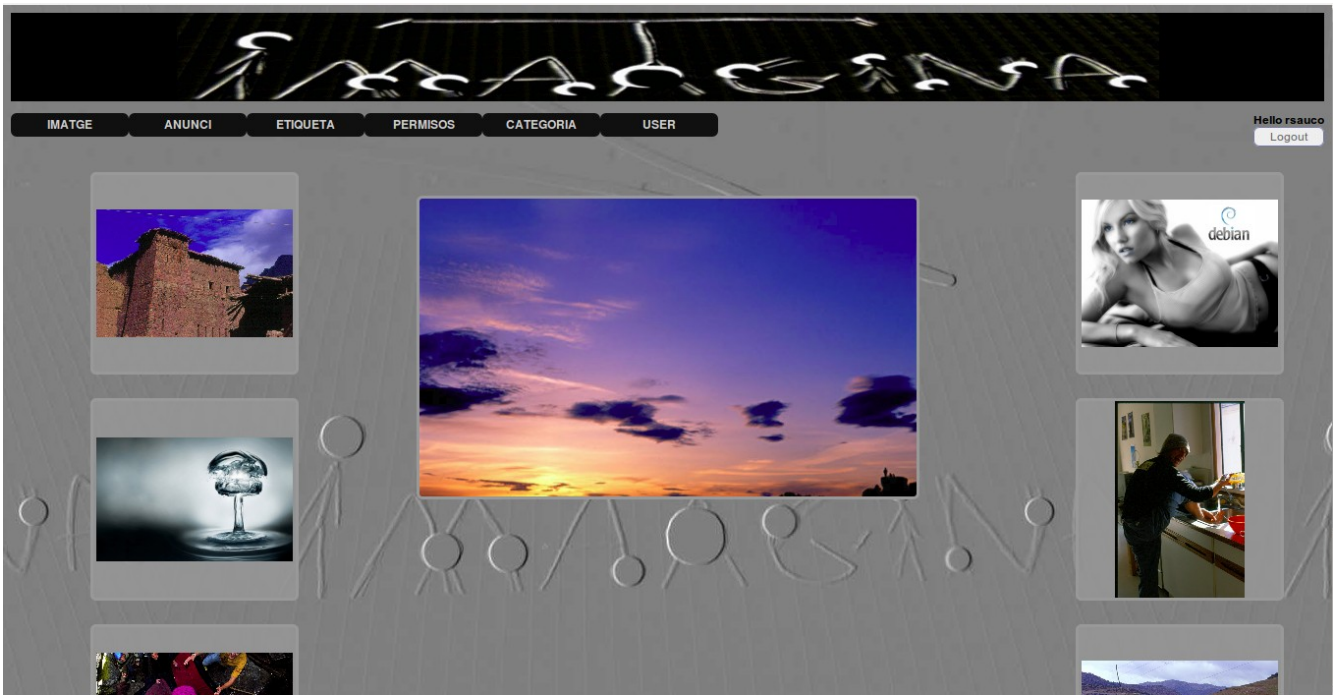
8.2.1 Modificació de la pàgina principal.

Un dels aspectes sobre els quals s'ha de treballar inicialment és en l'adaptació de la interfície que ofereix roo a una de més adequada a les nostres necessitats, en aquest cas, una que permeti a l'usuari portar a terme la interacció de la manera que s'ha dissenyat anteriorment.

Aquesta tasca requereix un cert coneixement del llenguatge de marcatge HTML i de l'estructura i funcionament de les fulles d'estil CSS. Mostrem la pàgina principal de l'aplicació que genera per nosaltres l'eina roo, a continuació la pàgina final que hem obtingut i, posteriorment, expliquem les modificacions més significatives que hem introduït per a portar a terme el canvi.



Il·lustració 6: Pantalla principal generada automàticament per l'eina roo.



Il·lustració 7: Pàgina principal de l'aplicació final.

Per a adaptar la pantalla principal s'han portar a terme tot un seguit de canvis. Els explicarem segons l'àrea a la qual afecten.

- Modifiquem l'arxiu `standard.css` per tal d'estilar la pàgina principal de la manera convenient. Com a referència per a portar a terme els canvis requerits he utilitzat [HFHTML06] i [UCSSR08].
- Afegim una nova pàgina `home.jspx` i fem que aquesta sigui la pàgina principal de l'aplicació, per defecte les peticions a `/imatgina` són servides de manera estàtica, nosaltres canviarem aquest comportament i crearem un controlador que s'ocupi de servir-les per tal de poder carregar un cert nombre d'imatges al model i mostrar-les a la pàgina d'inici. Definim un controlador al `package cat.imatgina.web` de la manera següent.

```
@RequestMapping("/")
@Controller
public class HomeController {

	public static int N..S = 4;

	@RequestMapping("/")
	public String doGet(Model uiModel) {

		// Add all the images to show in the home page to the model.
		uiModel.addAttribute("left_margin_images", Imatge.getHomePageImages(N..S));
		uiModel.addAttribute("right_margin_images", Imatge.getHomePageImages(N..S));
		uiModel.addAttribute("center_image", Imatge.getHomePageImages(1));

		return "home";
	}
}
```

- Crearem els mètodes necessaris a l'entitat Imatge, dels quals únicament `Imatge.getHomePageImage` serà públicament accessible. Aquest mètode obté un cert nombre d'imatges aleatòries i les retorna.
- El controlador afegeix les imatges obtingudes al model i retorna el nom lògic de la vista que ha de presentar aquestes imatges.
- L'arxiu `home.jsp` ocupada de mostrar aquesta vista no té res d'especial, remarcar principalment que s'ha reestructurat en profunditat la pàgina utilitzant mitjançant estils `CSS` i que s'han afegit al menú dos elements condicionals, un àrea de salutació i logout que es mostra en cas de tractar-se d'un usuari registrat i un àrea que mostra informació sobre les compres actives en cas d'existir alguna.

8.3 Configuració de paràmetres en temps de desenvolupament.

Alguns dels requeriments funcionals de l'aplicació seran molt diferents en temps de desenvolupament que una vegada es posi en servei.

Un clar exemple és el mecanisme de *caching*, una vegada en funcionament, ens interessarà que els navegadors web i servidors *proxy* desin una còpia temporal dels recursos estàtics de la nostra aplicació, per exemple la imatge de fons, es tracta d'un ítem d'una mida considerable a 42KB, i serà convenient que no s'envii en cada petició `GET`, especialment si tenim en consideració que no varia mai.

Per contra, durant la fase de desenvolupament, el fet que un navegador desi recursos ens pot portar a confusió ja que no veurem reflectits els canvis que portem a terme i no tindrem la certesa sobre si el navegador ens presenta amb la versió més actual del recurs o amb una versió recuperada del *cache*.

Per tal de enviar capçaleres sol·licitant als navegadors web de refrescar tot el contingut de l'aplicació en cada petició podem utilitzar un interceptor *`WebContentInterceptor`* definit de la següent manera al arxiu `webmvc-config.xml`. [STWI10]

```
    <!-- Register "global" interceptor beans to apply to all registered HandlerMappings -->
    <mvc:interceptors>
        <bean class="org.springframework.web.servlet.theme.ThemeChangeInterceptor"/>
        <bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"
p:paramName="lang"/>

        <!-- Intercept all requests to set no catching, only during development time -->
        <bean id="webContentInterceptor"
class="org.springframework.web.servlet.mvc.WebContentInterceptor">
        <property name="cacheSeconds" value="0"/>
        <property name="useExpiresHeader" value="true"/>
        <property name="useCacheControlHeader" value="true"/>
        <property name="useCacheControlNoStore" value="true"/>
    </bean>
    </mvc:interceptors>
```

També, en el mateix arxiu, haurem de canviar la configuració per tal de que les peticions a `/imatgina` no es vegin filtrades cap al recurs estàtic `/index.jspx` sinó que vagin al controlador `HomeController.java` que serà l'encarregat d'atendre'ls. Eliminem l'element de `webmvc-config.xml`:

```
<mvc:view-controller path="/" view-name="index"/>
```

Així mateix, durant la fase de desenvolupament ajustarem constantment el nivell de *logging* a les nostres necessitats, per exemple, si estem desenvolupant un component del paquet `cat.imatgina.web` ens pot resultar interessant veure registres generats per aquest paquet sense haver de cercar-los entre tots els generats pel reste de l'aplicació.

Una vegada arribi l'hora de desplegar l'aplicació, ens interessarà reduir el nivell de registre a un mínim, segons les condicions aquest pot ser el nivell `WARN` o `ERROR`.

8.4 Configuració del logging.

La creació de registres en temps de funcionament de l'aplicació, o *logging*, és una de les àrees en les quals podem treure'n el màxim profit de la programació orientada a aspectes. Es tracta d'un tema que creua de manera transversal totes les àrees de qualsevol aplicació, en el nostre cas, per exemple, ens interessa generar registres de les classes del domini com de les de presentació o les de servei, tot això sense oblidar les llibreries que estem utilitzant, de les quals no ens hem de preocupar però ja que venen configurades per a generar registres que haurien de ser correctes en la majoria de situacions.

Per tal de configurar les crides, principals, a nivell trace, a l'API de *logging* de l'aplicació web imatgina, utilitzarem un patró de disseny ideat per Ramnivas Laddad. [AJI03]

Aquest patró consisteix en la creació de un aspecte abstracte, que defineix un patró de registre amb indentació per tal de facilitar la identificació visual de la jerarquia de crides aquest *advice* s'aplica a un *joinpoint* també abstracte. Posteriorment, procedirem a la creació d'aspectes específics per a registrar diferents àrees de l'aplicació, cada un dels quals s'ocuparà d'implementar el *joinpoint* de la manera convenient.

Aquesta manera d'implementar la funcionalitat de *logging* de l'aplicació ofereix moltes avantatges sobre incloure totes les crides a l'API de log4j[20] en el mateix codi de l'aplicació. Per exemple, imaginem que tenim un aspecte que registra amb nivell INFO totes les crides als mètodes *create* del paquet .web, si per algun motiu decidim crear un nou mètode amb aquesta signatura no ens haurem de preocupar del *logging* sinó que aquesta funcionalitat serà inclosa en temps de compilació mitjançant l'aspecte que ja tenim definit.

20 D'entre tots els bastiments disponibles per a implementar el *logging* he triat aquest per ser un dels més populars i oferir una bona integració amb la resta d'eines que utilitzarem.

Durant la fase de desenvolupament he trobat algunes situacions que requerien ser registrades i em trobava, probablement degut a la meva inexperiència, en la impossibilitat de fer-lo d'altra manera que incloent-hi la crida directament en el codi java, per tant es poden trobar algunes crides a mètodes de registre entramades amb el codi de l'aplicació. Es tracta d'un punt que crec que es podria millorar però m'ha faltat temps per a poder estudiar-lo en profunditat.

Per defecte roo no habilita l'escriptura en un fitxer, per tal d'obtenir aquest servei modificarem aquest paràmetre en el fitxer log4j.properties de la manera següent:

```
log4j.rootLogger=WARN, stdout, R
```

On R és el nom de l'*appender* configurat en el mateix arxiu.

8.5 Configuració de les llengües de la interfície d'usuari.

Per defecte roo no ens permet incorporar la llengua catalana a la interfície d'usuari de les aplicacions generades amb aquesta eina. En el nostre cas, donat que l'existència mateixa de la nostra aplicació està motivada per tal de fomentar el comerç en català i millorar les oportunitats dels productors catalana-parlants, aquest es tracta d'un inconvenient greu, per sort és bastant fàcil de solucionar.

Hi ha diverses maneres de fer-lo, d'entre elles he triat la creació d'un *add-on* de roo per semblar-me la millor manera de convertir el meu treball en reutilitzable, tant per a mi mateix en ocasions futures, com per a altres persones que puguin estar interessades.

Una vegada creem un arxiu `messages_ca_ES.properties`, i disposant d'una imatge adequada per tal de servir d'icona, engeguem la línia de comandes roo i introduïm el següent comandament.

```
addon create i18n --topLevelPackage i18n.ca_ES --locale ca_ES --messageBundle messages_ca_ES --flag-
Graphic cat.png --language Català
```

Una vegada generat *l'add-on,* l'empaquem en una forma utilitzable, podem utilitzar *maven* per a simplificar-nos la tasca.

```
mvn clean install
```

I finalment utilitzem roo per a instal·lar el llenguatge català i, de pas i considerant-ho convenient, l'espanyol.

```
addon install bundle --bundleSymbolicName
web mvc language --code ca_ES
web mvc language --code es
```

Per tal de maximitzar la possibilitat de que un usuari es vegi rebut en la seva pròpia llengua utilitzarem l'interceptor *org.springframework.web.servlet.i18n.LocaleChangeInterceptor* el qual s'ocuparà d'assignar la llengua correcta a la interfície segons el local de la maquina utilitzada per a accedir l'aplicació.

Com que un usuari pot desitjar, per motius diversos, utilitzar una llengua diferent a la del seu local, oferirem les opcions per a modificar la llengua en el peu de pàgina, a més, utilitzarem *cookies* per a recordar les preferències dels usuaris. Aquesta configuració ja ens ve donada per defecte amb l'aplicació generada per roo.

Com a darrer detall, podem modificar l'ordre de les llengües al peu de pàgina per tal de mostrar el Català en primer lloc.

```
<span id="language"> <c:out value=" | " /> <spring:message
```

```
            code="global_language" /> <c:out value=": " />
            <util:language label="Català" locale="ca_ES" />
            <util:language label="English" locale="en" />
            <util:language label="Español" locale="es" />
</span>
```

8.6 Configuració de la seguretat.

La seguretat és un aspecte cabdal en qualsevol aplicació, encara més en una aplicació web exposada a Internet i que conté dades privats dels usuaris, incloent-hi dades financeres.

Definirem la seguretat des-de dues vessants, d'una banda, en el descriptor de la seguretat indicarem el nivell d'accessibilitat de diferents URLs del nostre lloc web i d'una altra, restringirem l'accés a nivell de mètode mitjançant l'ús d'anotacions.

8.6.1 Configuració bàsica de la seguretat.

Spring Roo ens ofereix la possibilitat de posar en funcionament un sistema bàsic d'autentificació el qual, posteriorment, podrem adaptar a les nostres necessitats, per a fer-lo, executem la següent comanda des de la línia d'ordres:

```
~.web roo> security setup
Created SPRING_CONFIG_ROOT/applicationContext-security.xml
Created SRC_MAIN_WEBAPP/WEB-INF/views/login.jspx
Updated SRC_MAIN_WEBAPP/WEB-INF/views/views.xml
Updated ROOT/pom.xml [added property 'spring-security.version' = '3.1.0.RELEASE'; added dependencies org.springframework.security:spring-security-core:${spring-security
.version}, org.springframework.security:spring-security-config:${spring-security.version}, org.springframework.security:spring-security-web:${spring-security.version},
org.springframework.security:spring-security-taglibs:${spring-security.version}]
Updated SRC_MAIN_WEBAPP/WEB-INF/web.xml
Updated SRC_MAIN_WEBAPP/WEB-INF/spring/webmvc-config.xml
```

Aquesta comanda porta a terme les modificacions necessàries en el projecte per tal de tenir un sistema d'autentificació bàsica basat en un fitxer d'usuaris.[21] També ens proveeix d'una pàgina d'autentificació bàsica i d'una funcionalitat de *logout* que afegeix a la barra de peu de pàgina.

8.6.2 Adaptació de la seguretat a les necessitats específiques de l'aplicació.

La configuració bàsica de la seguretat generada per roo és un bon punt de partida però normalment, com en el nostre cas, ens caldrà adaptar la configuració a les nostres necessitats, ho farem per parts.

8.6.2.1 Configuració de l'origen de les dades d'autentificació.

Un primer pas serà modificar el fitxer `applicationContext-security.xml`, originalment, els usuaris autoritzats i els seus drets es llegeixen del mateix fitxer[22]:

```
<user-service>
      <user name="admin" password="8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918"
            authorities="ROLE_ADMIN" />
      <user name="user" password="04f8996da763b7a969b1028ee3007569eaf3a635486ddab211d512c85b9df8fb"
            authorities="ROLE_USER" />
</user-service>
```

Nosaltres canviarem aquest comportament per tal de recuperar els usuaris de la base de dades d'imatgina definirem la següent propietat del component `authentication-manager`:

```
<jdbc-user-service id="userService"
      data-source-ref="dataSource"
      users-by-username-query="select name, contrassenya, actiu from usuari where name=?"
```

21 Per a una explicació detallada d'aquesta comanda veure [SRCB11] pàgines 310-316.
22 Es pot configurar

```
      authorities-by-username-query="select usuari.name, permisos.perms from usuari join
            permisos on usuari.id=permisos.usuari where usuari.name=?" />
```

De la mateixa manera, si ens és més convenient, es pot configurar l'autentificació per tal de recuperar les dades d'una altra font com una LDPA, un servei d'autentificació extern o d'altres[23].

8.6.2.2 Configuració de la funcionalitat recordar-me.

Als usuaris registrats de l'aplicació no els resultarà agradable haver d'autenticar-se cada vegada que accedeixen, serà convenient oferir-les la possibilitat de recordar les seves dades d'autenticació d'una sessió a la següent.

El bastiment Spring ens ofereix la possibilitat de configurar aquesta funcionalitat d'una manera senzilla i ràpida, únicament haurem d'afegir un element remember-me dins del component http al fitxer de configuració applicationContext-security.xml de la següent manera.

```
<remember-me key="imatginaKey" token-validity-seconds="15552000" use-secure-cookie="true" />
```

També haurem d'afegir al formulari un camp, el més apropiat serà de tipus checkbox, per tal de que pugin declarar el seu desig de ser autenticats automàticament en properes sessions. L'element següent ens servirà:

```
<div>
      <label for="remember_me" class="inline">
            <spring:message code="security_login_form_remember_me_message" />
      </label>
      <input id="remember_me" name="_spring_security_remember_me"
            type="checkbox" />
</div>
```

8.6.2.3 Utilització d'un canal segur

Imatgina haurà de processar informació sensible dels usuaris, tal i com informació bancària, números de targeta de crèdit o simplement les contrasenyes dels usuaris, per tal de garantir que aquesta informació no cau en mans de persones no autoritzades, requerirem la utilització d'un canal segur per a la seva transmissió. El bastiment *Spring* ens facilita considerablement aquesta tasca al permetre'ns d'obtenir aquesta funcionalitat de manera declarativa amb l'atribut `requires-channel`.

```
<intercept-url pattern="/login" access="isAnonymous()" requires-channel="https" />
```

Afegint aquest element a totes les `URLs` que requereixen encriptació el servidor requerirà al client l'establiment d'una connexió segura abans d'enviar la pàgina.

NOTA: La configuració del servidor per tal de donar suport a connexions segures m'ha resultat feixuga. Com que desconec si el contenidor en el qual s'avaluarà el treball fet tindrà aquesta característica habilitada, m'abstindré de configurar la seguretat en cas de que un mal funcionament o una excessiva complicació del desplegament puguin influir negativament en la imatge que ofereix el producte.

Basti dir que la configuració de seguretat és realment senzilla amb Spring i que, una vegada triat el servidor en el qual es desplegarà permanentment l'aplicació, es pot configurar i provar en unes hores.

23 La configuració utilitzada per a llegir les dades de la base de dades així com detalls sobre altres opcions de configuració de l'autentificació es poden consultar a [SIA11] pàgines 238-245.

8.7 Modificacions introduïdes a les classes autogenerades.

Com és lògic, les classes generades automàticament per l'eina roo no seran aptes per a les necessitats finals de la majoria d'usuaris i aplicacions sinó que haurem de modificar-les per tal d'adaptar-les als requeriments particulars de cada situació, en el cas d'Imatgina, hem portat a terme un seguit de modificacions en la majoria de les classes que s'exposen a continuació.

8.7.1 Usuari.

Per tal de tenir accés a tota la funcionalitat que ofereix Imatgina, els usuaris han de registrar-se i autenticar-se davant l'aplicació. Per la seva flexibilitat, s'ha triat un sistema d'autentificació amb contrasenya per a portar el control d'accés als recursos.

Quan un usuari intenta accedir a una zona reservada de l'aplicació, o manifesta la seva intenció d'autenticar-se, se li presenta amb una pàgina de *login* en la qual ha d'introduir el seu nom d'usuari i la seva contrasenya. Els detalls del funcionament d'aquesta part de l'aplicació han quedat reflectits a l'apartat Configuració de la seguretat.

8.7.1.1 registre de nous usuaris.

La configuració de seguretat que hem establert rep la contrasenya de l'usuari del formulari i porta a terme un *hash* del valor obtingut per tal de comparar-ho amb el valor existent a la base de dades. De moment roo no ens ofereix la possibilitat de definir un camp *contrasenya* i ocupar-se de manera automàtica d'aquestes tasques així que, en el controlador que processa el formulari de creació d'un nou usuari, ens haurem d'ocupar de codificar el valor abans d'enviar-lo a la base de dades. Ho farem amb l'ajuda d'una classe de servei per tal de poder reutilitzar el codi més fàcilment si ho necessitem posteriorment.

El mètode ocupat d'encriptar la contrasenya queda definit a la classe `cat.imatgina.services.Encoder`.

```
public static String encodeSha256Sum (String input, String algorithm) {
      MessageDigest messageDigest = null;
      try {
            messageDigest = MessageDigest.getInstance(algorithm);
      } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
      }

      messageDigest.update(input.getBytes(),0, input.length());
      String hashedPass = new BigInteger(1,messageDigest.digest()).toString(16);
      if (hashedPass.length() < 32)
            hashedPass = "0" + hashedPass;
      return hashedPass;
}
```

Des del controlador hem de cridar aquest mètode i recollir el valor en el camp `usuari.contrasenya`.

```
// Hash the user password before persisting in the database.
usuari.setContrassenya(Encoder.encodeSha256Sum(usuari.getContrassenya(), Encoder.sha256));
usuari.persist();
```

També ens hem d'assegurar de donar a l'usuari les autoritzacions pertinents, en el cas de nous usuaris les-hi atorgarem el permís `'ROLE_USER'`.

```
// Allow all new users the role 'ROLE_USER'
Permisos p = new Permisos();
p.setUsuari(usuari);
p.setPerms("ROLE_USER");
p.persist();
```

A més, serà convenient que, una vegada registrats, els usuaris queden autenticats directament davant del sistema, per a fer-lo, recuperarem l'`authentificationManager` que ens proporciona el context, aprofitarem les anotacions per a injectar-lo en el controlador, i amb l'ajuda d'aquest `bean`, autentificarem a l'usuari i l'enviarem cap a la seva pàgina d'acollida.

```
public class UsuariController {

      @Autowired
      @Qualifier("authenticationManager")
      protected AuthenticationManager am;
      ...
      @RequestMapping(method = RequestMethod.POST, produces = "text/html")
      public String create(@Valid Usuari usuari, BindingResult bindingResult,
             Model uiModel, HttpServletRequest httpServletRequest) {

             ...

        Authentication request = new UsernamePasswordAuthenticationToken(usuari.getNom(), oldPass);
        Authentication result = am.authenticate(request);
        SecurityContextHolder.getContext().setAuthentication(result);

        return "redirect:/usuaris/" + usuari.getId();
    }
      ...
}
```

8.7.1.2 Baixa d'un usuari.

Un altre punt a tenir en consideració és que sempre podrà donar-se el cas que un usuari vulgui donar-se de baixa i, per tant, que les seves publicacions deixen de ser accessibles en Imatgina, aquesta funció serà accessible a tots els usuaris autenticats però, donada la seva gravitat, demanarem una confirmació extra. Aprofitarem aquest fet per tal de recuperar l'identificador de l'usuari de la base de dades.

```
@RequestMapping(value="/unsubscribe")
public String unsubscribe (HttpServletRequest request, Model uiModel) {

      /* Retrieve the current's user's id and add it to the model*/
      final String currentUserName =
SecurityContextHolder.getContext().getAuthentication().getName();
      final Long userId = Usuari.getUserIdByName(currentUserName);
      uiModel.addAttribute("userId", userId);

      return "usuaris/unsubscribe";
}
```

Si l'usuari confirma, enviarem un `DELETE` a l'`URL /usuaris/{id}`, per a poder utilitzar aquest mètode, no suportat en `html`, utilitzarem el recurs mostrat a continuació.

```
<form id="command" method="post" action="${urlProceed}" >
      <input type="hidden" value="DELETE" name="_method"/>
      <!-- ... -->
</form>
```

El mètode que s'ocupa de donar a un usuari de baixa és el següent.

```
/* Change the user's activity level to inactive. */
Usuari usuari = Usuari.findUsuari(id);
usuari.setActiu(false);
usuari.persist();

/* Disable all users images */
Imatge.setAsInactiveByUserId (usuari);
```

```
/* Disable all users adds */
Anunci.setAsInactiveByUserId(usuari);
```

Es pot veure que aquest mètode, segons va quedar recollit en l'especificació del disseny, marca l'usuari i totes les seves publicacions com a inactius. En un moment posterior, el sistema podrà aprofitar aquest fet per a eliminar les seves dades o un administrador podrà restaurar l'usuari a un estat d'actiu, per exemple segons una petició d'aquest. La implementació del mètodes `setAsInactiveByUserId()` serà molt similar en les dues classes. D'entre totes les possibilitats he triat la que optimitza l'ús que es fa de la connexió a la BD enviant únicament una sentència SQL.

```
@Transactional
public int setAsInactiveByUser (Usuari user) {
	EntityManager em  = Imatge.entityManager();

	return em.createQuery("UPDATE Imatge SET actiu=false WHERE usuari= :user")
		.setParameter("user", user).executeUpdate();
	}

public static int setAsInactiveByUserId (Usuari user) {
	return new Imatge().setAsInactiveByUser(user);
}
```

Una altra opció hagués sigut utilitzar roo per a generar un *Roo_Finder* que ens retornés totes les publicacions d'un usuari i iterar sobre aquestes, marcant-les com a inactives i tornant a persistir-les. Tot i que aquesta possibilitat és la que s'adapta millor a l'eina amb la qual estem treballant i que permetria més flexibilitat enfront futurs canvis en el disseny de la BD, m'ha semblat que això no justificava el cost extra d'haver de recuperar i tornar a enviar a la BD totes les publicacions de l'usuari.

8.7.1.3 Autentificació d'usuaris.

L'autentificació d'usuaris davant del sistema ens ve quasi donada per les eines amb les que treballem[24], únicament requereix una mica de configuració, canviarem la pàgina d'autenticació per a adaptar-la a les nostres necessitats, per a definir la nova configuració m'he basat fortament en [SIA11] pàgina 230.

La pàgina d'autenticació s'ocupa de mostrar a l'usuari tres camps que es mostren a continuació.

La informació que aquest introdueix en aquests camps s'envia a l'`authentification-manager` que hem declarat en l'arxiu `applicationContext-security.xml`. Aquest contrasta la informació obtinguda contra la que figura a la base de dades, la qual recupera gràcies al `bean jdbc-user-service` configurat en el mateix arxiu i, segons el resultat de la validació, registra a l'usuari com autentificat davant el sistema o el retorna a la pàgina d'autenticació tot afegint al model els necessaris missatges que expliquen el motiu, per a ser mostrats.

8.7.2 Categoria.

Aquesta entitat, juntament amb la seva entitat germana, etiqueta, patirà més modificacions que algunes altres donat que no es tracta d'una entitat final, en el sentit que els usuaris no interactuaran directament amb ella sinó que ens serveix de categoritzador d'entitats anunci.

8.7.2.1 Creació d'entitats categoria.

Per a la creació d'entitats categoria utilitzarem els mateixos trucs que hem utilitzat en la creació d'imatges i anuncis, és a dir, afegirem al model la informació necessària que s'ha d'incloure en aquesta entitat de manera automàtica i l'afegirem a l'objecte que estem creant mitjançant la utilització de camps ocults.

24 Veure 8.6.1 Configuració bàsica de la seguretat.

En el mètode que prepara el formulari per a la seva presentació a l'usuari afegim el codi.

```
public String createForm(Model uiModel) {
      Usuari user = Usuari.findUsuari(Usuari.getUserIdByName(SecurityContextHolder.getContext()
                  .getAuthentication().getName()));

      Categoria cat = new Categoria();
      cat.setCreador(user);

      uiModel.addAttribute("categorias", Categoria.findAllCategorias());
      uiModel.addAttribute("categoria", cat);
      return "categorias/list";
}
```

I en la vista que el presenta inclourem un camp amb l'usuari que recuperem de amb el valor assignat, ocultarem els camps generats per roo i l'instruirem que no els ha de regenerar.

```
<input name="creador" type="hidden" value="${categoria.creador.id}" />
<field:select field="creador" render="false" ...
```

8.7.2.2 Actualització d'una categoria.

El mètode per a actualitzar una categoria és el mateix que per a crear-la, i les modificacions introduïdes al codi són quasi idèntiques i no les reflectirem aquí.

8.7.2.3 Visualització d'una categoria.

Per defecte roo ens genera el codi i una vista per a visualitzar els detalls d'una categoria determinada, com que aquest és un punt que no ens interessa massa, el modifiquem per tal que un accés a una categoria determinada ens mostri un llistat d'anuncis relacionats amb aquesta.

Modificarem el codi del controlador per tal que recuperi els anuncis relacionats amb la categoria i que retorni el nom lògic de la vista que presentarà els anuncis.

```
    @RequestMapping(value = "/{id}", produces = "text/html")
    public String show(@RequestParam(value = "page", required = false) Integer page,
                    ... {

        Categoria cat = Categoria.findCategoria(id);
        ...
            uiModel.addAttribute("anuncis", Anunci.findAnuncisByCategoria (cat, sizeNo, page));
        ...
        return "anuncis/list";
    }
```

Per brevetat no s'ha mostrat el codi relacionat amb la paginació i el control d'errors. Els mètodes de la classe anunci són molt semblants als que es mostraran amb relació a la funcionalitat de cerca d'imatges, es poden consultar aquests o el codi font de l'aplicació.

8.7.3 Etiqueta.

La classe etiqueta comparteix moltes similituds amb la classe categoria, es tracta bàsicament d'un classificador per a una classe més important en l'aplicació, en aquest cas la classe imatge, peça clau d'Imatgina. La seva implementació tindrà molts punts en comú amb les categories que podem aprofitar per a agilitzar el procés.

8.7.3.1 Creació d'entitats etiqueta.

A l'igual que en el cas de les categories, els únics usuaris que poden crear etiquetes són els administradors, portarem a terme els mateixos canvis que anteriorment. Modificarem el mètode `EtiquetaController.createForm` per tal que crei una nova `Etiqueta` i assigni l'usuari actual com creador.

```
@RequestMapping(params = "form", produces = "text/html")
public String createForm(Model uiModel) {

	Etiqueta et = new Etiqueta();
	et.setCreador(Usuari.findUsuari(Usuari.getUserIdByName(SecurityContextHolder.getContext()
			.getAuthentication().getName())));
	uiModel.addAttribute("etiqueta", et);

	return "etiquetas/create";
}
```

Modificarem la vista que presenta el formulari de creació per tal que marqui aquest usuari com el creador de l'etiqueta d'una manera transparent per a l'usuari.

```
<input name="creador" type="hidden" value="${etiqueta.creador.id}" />
<field:select field="creador" ... render="false" z="user-managed" />
```

8.7.3.2 Edició d'una entitat etiqueta.

Els mètodes relacionats amb l'edició d'una etiqueta seran molt semblants als de creació i no es reflectiran en aquest document. La única diferència resideix en que enlloc de crear una nova etiqueta es recupera l'existent de la base de dades. Consultar el codi font per a més informació.

8.7.3.3 Eliminació d'entitats etiqueta.

No es permetrà l'eliminació d'aquestes entitats una vegada creades per tal de mantenir la consistència dels etiquetatges portats a terme pels usuaris.

8.7.4 Imatge.

Donat el caràcter clau de peça central que aquest component ha de representar en el desenvolupament que estem portant a terme, serà un dels que s'haurà de modificar i redissenyar més completament, des de la seva creació a la seva presentació, a més, també s'haurà de desenvolupar un mecanisme de cerca d'imatges segons diferents criteris.

8.7.4.1 Creació d'entitats imatge.

Per defecte, roo exposa tots els camps de la imatge a l'usuari en el moment de la seva creació, haurem d'introduir alguns canvis ja que a nosaltres ens interessa que alguns dels camps s'omplin de manera automàtica.

També haurem de modificar el formulari original, el qual únicament pot processar entrades en format text, per tal de permetre als usuaris enviar al servidor les imatges que formaran la part central d'Imatgina.

El primer pas serà crear una nova imatge i omplir els camps adequats amb els valors necessaris per tal de que l'usuari no l'hagi de fer manualment, aquests camps seran la data de publicació i l'usuari, el qual es correspondrà amb l'usuari registrat corresponent a la sessió activa. El mètode de la classe `ImatgeController` que s'ocupa d'aquestes tasques és el següent.

```
@RequestMapping(params = "form", produces = "text/html")
public String createForm(Model uiModel) {

    // Get the principal object
    Usuari user = Usuari.findUsuari(Usuari.getUserIdByName(SecurityContextHolder.getContext()
                    .getAuthentication().getName()));

    // Get an Imatge object to populate the model, and populate it.
    Imatge image = new Imatge();
    image.setUsuari(user);

    Calendar c = GregorianCalendar.getInstance();
    StringBuffer sb = new StringBuffer();

    sb.append(new SimpleDateFormat("MMM").format(new Date(c.getTimeInMillis()))+" ");
    sb.append(c.get(Calendar.DAY_OF_MONTH)+", ");
    sb.append(c.get(Calendar.YEAR));

    uiModel.addAttribute("now", sb.toString());
    uiModel.addAttribute("etiquetas", Etiqueta.findAllEtiquetas());
    uiModel.addAttribute("usuari", user);

    List<String[]> dependencies = new ArrayList<String[]>();
    if (Usuari.countUsuaris() == 0) {
        dependencies.add(new String[] { "usuari", "usuaris" });
    }
    uiModel.addAttribute("dependencies", dependencies);
    uiModel.addAttribute("imatge", image);

    addDateTimeFormatPatterns(uiModel);
    return "imatges/create";
}
```

Les parts rellevants d'aquest mètode són:

- Crear una nova instància de la classe `Imatge` la qual utilitzarem per a recollir les dades, tant les generades automàticament com les introduïdes per l'usuari. Aquest objecte s'afegirà posteriorment al model.
- Obtenir l'objecte corresponent a l'usuari que es troba autenticat davant del sistema i afegir una referència a aquest objecte.
- Obtenir la dada actual i, amb aquesta dada com a referència, generar un nou objecte `String` i afegir-lo al model. Aquest objecte s'utilitzarà per a omplir de manera automàtica el camp `dataPublicacio` de la imatge que es crearà.

Una vegada tenim la informació en el model, ens hem d'ocupar de mostrar-la adequadament i recollir altra informació que l'usuari pugui considerar important, per exemple el arxiu d'imatge! D'aquestes tasques s'ocuparà la vista `/views/imatges/create.jspx`.

En aquest cas podem aprofitar quasi completament la vista generada per roo, degut però al tractament que els *tags* generats per roo fan dels camps que no es visualitzen[25], és a dir, quan posem l'atribut `render="false"`, ens caldrà incloure un parell de camps ocults per tal de que la imatge enviada per l'usuari passi la validació.

```
<input type="hidden" name="dataPublicacio" value="${now}" />
<input type="hidden" name="usuari" value="${imatge.usuari.id}" />
```

També ens caldrà incloure un camp per tal de que l'usuari pugui introduir l'arxiu d'imatge i, per tal de

25 Si els camps d'un formulari no es visualitzen, roo posa el valor `null` a l'atribut corresponent, fins i tot si aquest camp ja conté algun valor diferent d'aquest. Aquest comportament no és sempre ideal i s'ha generat un informe Jira sol·licitant que es modifiqui. De moment, tenim l'opció de solucionar-lo d'altres maneres, com la inclusió d'un camp ocult que ha sigut la triada.

tenir accés a aquest objecte des del nostre controlador, haurem modificar la capçalera del formulari de manera que declari que es tracta d'un formulari multipart.

```
<form:create id="fc_cat_imatgina_domain_Imatge" modelAttribute="imatge"
      multipart="true" path="/imatges" render="${empty dependencies}" z="user-managed">
...
      <div class="input_field">
            <label for="image"><spring:message code="label_cat_imatgina_domain_compra_imatge"/>
            </label> <input name="image" type="file" /></div>
...
</form:create>
```

Quan l'usuari cliqui sobre enviar, s'enviarà un mètode `POST` amb la informació introduïda que serà processat pel següent mètode d'`ImatgeController`.

```
    @RequestMapping(method = RequestMethod.POST, produces = "text/html")
    public String create(@Valid Imatge imatge, BindingResult bindingResult,
            Model uiModel, HttpServletRequest httpServletRequest,
            @RequestParam(value="image", required=false) MultipartFile image) {

        ...

        imatge.persist();

        try {
            if (!image.isEmpty()) {
                  Imatge.validateImage(image);
        ...
                  imatge.saveImage(path, image);
            }
        } catch (ImageUploadException e) {
            bindingResult.reject (e.getMessage());
        }
        ...
      return "redirect:/imatges/" + imatge.getId().toString();
    }
```

Aquest mètode s'ocupa de:

- Comprovar si hi ha algun error en el nou objecte imatge creat, si aquest és el cas, es rebutjarà la imatge i es tornarà a mostrar el formulari de creació tot donant la informació adequada sobre els errors que s'hagin produït. Aquesta validació inclou la comprovació del tipus d'arxiu d'imatge utilitzat, inicialment Imatgina suportarà únicament imatges en el format jpg, una vegada desenvolupat i testada l'aplicació, inclourem suport per a altres formats d'imatge.
- Si no hi ha errors de validació, desarem la imatge en el servidor, d'entre les diferents opcions per a fer-lo, com per exemple desar les imatges en la mateixa base de dades, he triat la utilització del sistema de fitxers del propi sistema operatiu.
- L'estratègia serà desar dues còpies, una en format *thumbnail* i una altra en una mida major. Cada arxiu d'imatge es trobarà en el seu directori *(thumbnails* o *mida_completa*) i serà identificada per el valor de l'atribut `id` de l'objecte `imatge` al qual es troba associat.
- Finalment retornarem el nom lògic de la pàgina que ens permetrà visualitzar els detalls de la imatge creada.

8.7.4.2 Edició d'entitats imatge.

Com és fàcil d'imaginar, el cas d'edició d'imatges és molt semblant al de creació. S'ha de prendre la decisió, a nivell de negoci, de si permetrem a un usuari alterar una imatge que s'ha pujat al servidor anteriorment, la decisió que s'ha pres és que si, dotant als autors d'una certa capacitat d'edició d'imatges cre-

ades anteriorment sense necessitat de eliminar-les i recrear-les.

Al controlador aprofitarem el mètode `ImatgeController.create`, i, per a diferenciar entre la creació d'una nova imatge i l'edició d'una existent, comprovarem el mètode `HTTP`, una creació enviarà un `POST` i una edició serà un `PUT`. El codi que ho fa és el següent.

```
if (method == null)
    imatge.persist();
else if (method.equals("PUT"))
   imatge.merge();
```

La vista serà molt semblant a la utilitzada en la creació de noves imatges, amb el qual no la mostrarem.

8.7.4.3 Visualització d'una entitat imatge.

Per defecte, la vista que roo ha generat per a visualitzar l'entitat imatge és la següent.

Il·lustració 8: JSP original de vista d'una imatge.

Es pot apreciar que aquesta presentació de la informació dista bastant de la desitjada, per a començar, no ens mostra la part que ens resulta més important, la mateixa imatge. Per tal de solucionar-lo, crearem una nova *jsp*, basada en el *tile default* per tal d'incloure capçalera, menú i fons de pantalla.

Aquesta nova pàgina creada **rep com a paràmetres un objecte imatge** i el valor del seu camp id[26], estructuralment ens interessa dividir la pàgina en tres camps, un per a la imatge mateixa la qual recuperarem del sistema de fitxers del servidor mitjançant el seu valor ID, un per la informació associada a aquesta, la qual es pot recuperar del mateix objecte imatge que es rep com a paràmetre i un altre per a informació suplementària tal i com poden ser uns *thumbnails* de imatges de característiques semblants que poden interessar a la persona que visualitza la imatge.

Respecte al camp que mostra la informació associada a la imatge, segons l'usuari actual sigui o no qui la ha publicat, ens interessarà mostrar una informació o una altra, per exemple, si l'usuari actual és el propietari, ens pot interessar mostrar un enllaç per a editar dades de la imatge, i per contra, si es tracta d'una imatge publicada per un altre usuari, ens interessarà mostrar un enllaç per tal d'afegir la imatge a la cistella de la compra.

Per tal de no haver-hi d'escriure el mateix condicional diverses vegades dins de la mateixa pàgina, farem la comprovació al començament d'aquesta i assignarem el resultat a la variable de tipus booleà `isPublisher`.

```
<!-- Create a boolean value to register if the user is also the publisher. -->
<c:choose>
      <c:when test="${pageContext['request'].userPrincipal != null}">
```

26 Tot i que aquest comportament és redundant no l'he modificat per a mantenir la consistència a través dels controladors.

```
            <security:authentication property="principal.username" var="userName" />
            <c:set value="${userName == imatge.usuari.nom}" var="isPublisher" />
      </c:when>
      <c:otherwise>
            <c:set value="false" var="isPublisher" />
      </c:otherwise>
</c:choose>
```

Per tal de controlar possibles errors haurem de portar a terme l'assignació dins d'un bucle condicional que faci una comprovació prèvia per tal de veure si hi ha algun usuari autentificat o es tracta d'un visitant.

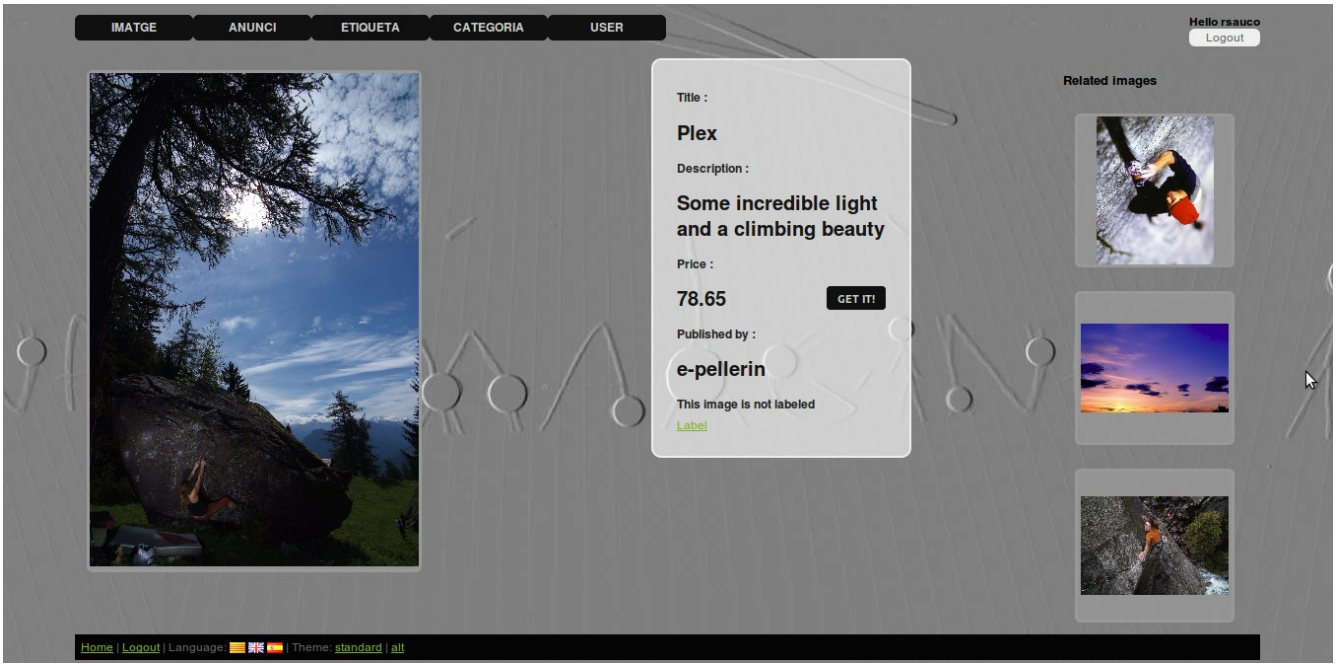
A partir d'aquest moment utilitzarem aquesta variable per tal de fer les comprovacions, per exemple, per tal de mostrar el preu d'una imatge a tots els usuaris excepte aquell qui la va publicar podem utilitzar el codi a continuació.

```
<security:authorize access="${!isPublisher}">
      <H1><spring:message code="image_show_price" />:</H1>
      <H2>${imatge.preu}</H2>
</security:authorize>
```

Finalment, recuperarem un cert nombre d'imatges relacionades amb la que estem visualitzant per tal de mostrar una barra lateral amb imatges similars.

```
    public List<Imatge> findRelated (Imatge image, int howMany) {
      List<Imatge> images = new ArrayList<Imatge>();
      // Iterate over the labels
      Iterator<Etiqueta> it1 = image.getEtiquetes().iterator();
      while (it1.hasNext() && images.size() < howMany) {
            Iterator<Imatge> it2 = Imatge.getImagesByLabel(it1.next()).iterator();
            while(it2.hasNext() && images.size() < howMany) {
                  images.add(it2.next());
            }
      }}
```

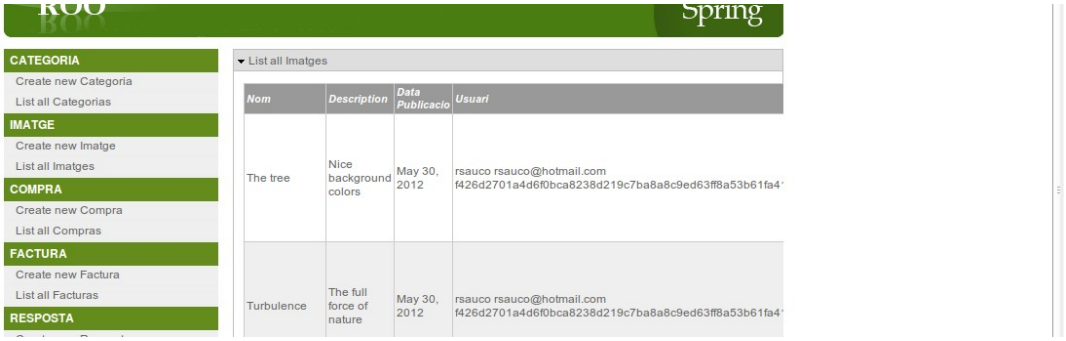
El resultat es pot veure a continuació.



Il·lustració 9: Pàgina de visualització d'una imatge.

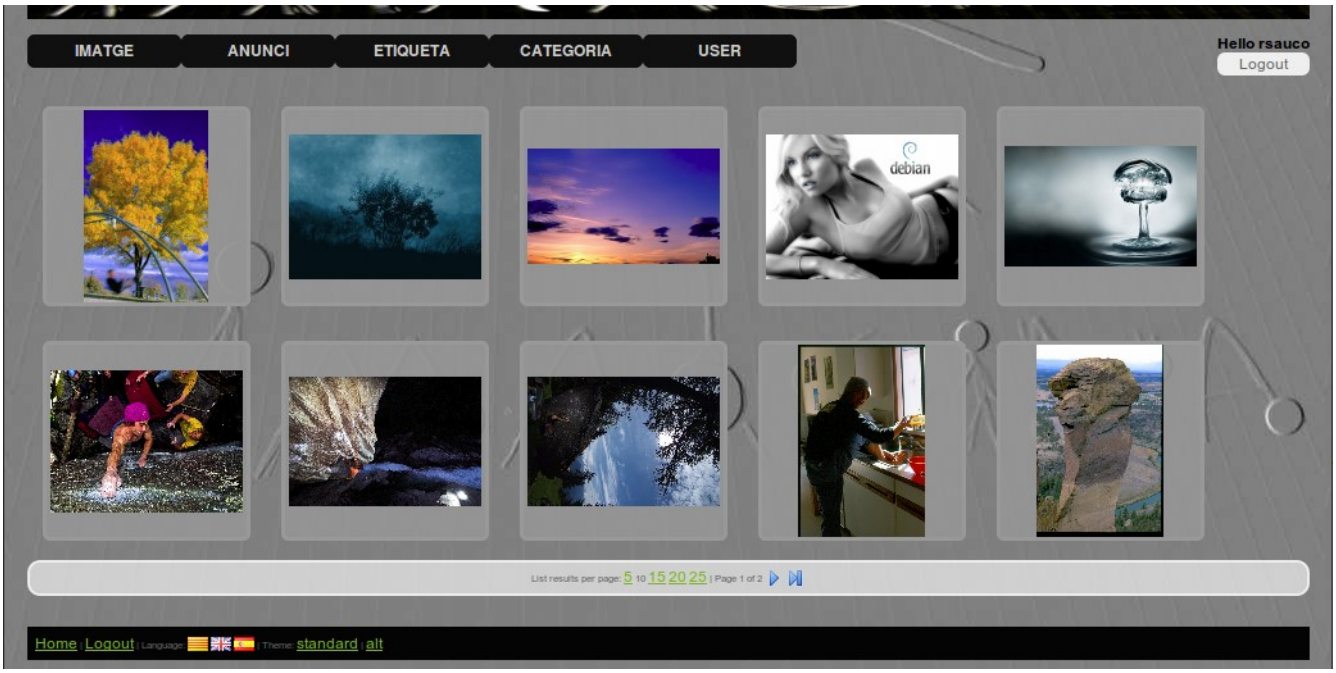
8.7.4.4 Visualització de múltiples entitats imatge.

Ens trobem en un cas semblant a l'anterior, per defecte, la pàgina que genera roo únicament mostra informació textual mentre que als usuaris de l'aplicació els interessarà principalment el contingut visual d'aquesta, haurem de modificar la imatge original que veiem a continuació.



Il·lustració 10: Presentació original del llistat d'imatges.

I obtenir aquesta presentació.



Il·lustració 11: Pàgina de visualització de múltiples imatges.

En aquest cas ens ha calgut únicament modificar la presentació ja que la sola tasca que porta a terme el controlador és afegir una `List<Imatge>` al model i cridar la vista lògica corresponent. A continuació podem veure la pàgina jsp que s'ocupa de mostrar un llistat d'imatges.

```
<ul id="show_all_images_list">
      <c:forEach items="${imatges}" var="image">
            <spring:url value="/resources/images_DB/small/{imageId}.jpg"
                  var="imageSRC">
                  <spring:param name="imageId" value="${image.id}" />
            </spring:url>
            <spring:url value="/imatges/${image.id}" var="imageURL" />
            <li class="thumbnail_container"><a href="${imageURL}"
                  name="veure_detalls" title="Veure ${image.nom}">
```

```
                    <div class="thumbnail_frame">
                                <img alt="image:${image.id} " src="${imageSRC}" />
                    </div></a>
                    <div class="thumbnail_footer"><SMALL>...</SMALL></div></li>
</c:forEach>
</ul>
<c:if test="${not empty maxPages}">
      <div id="image_paginator" class="white_panel" clear="left">
            <util:paginationPlus maxPages="${maxPages}" page="${param.page}"
                  size="${param.size}" /></div>
</c:if>
```

Aquest codi recupera un llistat d'imatges, que el controlador ha posat al model, i itera per cada una mostrant-les per pantalla. Mantenir la generalitat d'aquesta vista ens permetrà posteriorment utilitzar-la per a mostrar els resultats d'una cerca, ja sigui per paraula clau, etiqueta o qualsevol altre tipus de funcionalitat que puguem desenvolupar més tard.

També podem observar que s'ofereix un mecanisme de paginació, el qual treballarà amb el controlador per tal de permetre'ns veure més o menys imatges per pàgina i, en cas que hi hagi més imatges de les que entren en una pàgina, ens permeti moure'ns entre aquestes.

8.7.4.5 Cerca d'imatges segons una paraula clau.

A mesura que s'incrementi el nombre d'imatges a la base de dades d'Imatgina, deixarà de tindre sentit visualitzar un llistat d'aquestes, de fet aquesta funcionalitat ni tans sol se'ls ofereix als usuaris[27]. Per a substituir-la oferirem un servei de cerca dissenyat i implementat de forma modular.

Inicialment implementarem un servei de cerca per paraula clau, que comparará una cadena de text introduïda per l'usuari contra el continguts dels camps descripció i nom de les imatges registrades. També oferirem una funcionalitat de cerca segons etiqueta ja que als usuaris les hi pot resultar còmode de clicar sobre una etiqueta determinada per tal de trobar imatges relacionades amb aquesta.

Per tal de dirigir els usuaris cap al servei de cerca inclourem un enllaç al menú principal.

```
<menu:item id="i_imatge_search" messageCode="global_menu_search" url="/imatges/searchForm" />
```

També s'haurà de definir en un dels arxius de definició de `tiles`, l'associació entre el nom lògic d'aquesta vista i la seva implementació.

```
<definition extends="default" name="imatges/search">
        <put-attribute name="body" value="/WEB-INF/views/imatges/search.jspx"/>
    </definition>
```

Aquest enllaç mostrarà el següent formulari.

```
<DIV id="image_search_description_like" class="white_panel">
      <spring:url value="/imatges/search" var="form_url"></spring:url>
      <form:form action="${form_url}" method="GET">
            <h1><spring:message code="search_images_by_keyword" /></h1>
            <spring:message code="image_search_text" var="${search_keyword}" />
            <LABEL for="search">${search_keyword}</LABEL>
            <input type="hidden" name="size" value="${empty param.size ? 20 : param.size}" />
            <input type="hidden" name="page" value="1" />
            <input name="search" type="text" placeholder="Search keyword" />
            <input type="submit" class="button" />
      </form:form></DIV>
```

Podem veure que es tracta d'un formulari molt simple que recull una cadena de text de l'usuari i l'envia

27 Per contra si que oferirem aquesta funcionalitat als administradors ja que les hi pot resultar útil.

a `/imatges/search` com a paràmetre d'un `HTTP GET`, hem triat la utilització del mètode `GET` enlloc del `POST` per tal de facilitar el funcionament de la paginació i el suport d'idiomes, més avall s'explica la manera en que s'ha fet. A més s'afegeixen els paràmetres necessaris per a controlar la paginació.

El mètode que s'ocupa de donar servei a aquesta petició és ImageController.search.

```
    @RequestMapping(value= "/search")
    public String search(
              @RequestParam(value = "search", required = true) String searchString,
              @RequestParam(value = "page", required = false) Integer page,
              @RequestParam(value = "size", required = false) Integer size,
              Model uiModel) {
        if (page != null || size != null) {
            int sizeNo = size == null ? 10 : size.intValue();
            List<Imatge> images = Imatge.searchImatgeByDescription(searchString, sizeNo,
page.intValue());
            uiModel.addAttribute("imatges", images);
            long imagesReturnedByQuery =  Imatge.countImagesWithDescription(searchString);
            float nrOfPages = (float) imagesReturnedByQuery / sizeNo;
            uiModel.addAttribute("maxPages", (int) ((nrOfPages >
                    (int) nrOfPages || nrOfPages == 0.0) ? nrOfPages + 1 : nrOfPages));

        } else {
            uiModel.addAttribute("imatges", Imatge.searchImatgeByDescription(searchString, 20, 1));
        }
        addDateTimeFormatPatterns(uiModel);
        return "imatges/list";
    }
```

Els punts claus d'aquest mètode són:

- Recull els paràmetres `size`, `page` i `searchString` de l'objecte request.
- Comprova si ha de portar a terme paginació.
- Crida el mètode `Imatge.searchImatgeByDescription` i afegeix el resultat obtingut, una llista d'imatges, com un atribut del model.
- Si s'ha de portar a terme paginació, calcula el nombre total de pàgines que ocuparien els resultats amb el nombre actual d'imatges per pàgina que es mostren i afegeix aquest valor al model.
- Aprofita que la vista `imatges/list` és generalista i la utilitza per a mostrar els resultats obtinguts en forma de llistat d'imatges. El fet d'aprofitar aquesta vista presenta tots els avantatges clàssics de la reutilització de codi, si volem canviar la manera de presentar les imatges o les opcions d'interacció que aquestes ofereixen únicament ho haurem de fer en un lloc.

El mètode `Imatge.searchImatgeByDescription`, ocupat de recuperar les imatges de la base de dades en funció d'una cadena de text donada és el següent.

```
public static List<Imatge> searchImatgeByDescription(String description, int maxResults, int page) {
    if (description == null || description.length() == 0)
        throw new IllegalArgumentException("The description argument is required");

        EntityManager em = Imatge.entityManager();
        TypedQuery<Imatge> q = em.createQuery("SELECT o FROM Imatge AS o " +
                    "WHERE (LOWER(o.description) LIKE LOWER(:description)  " +
                    "OR LOWER(o.nom) LIKE LOWER(:description))" +
                    "AND o.actiu = true", Imatge.class).setFirstResult
                    (page == 1 ? 0 : (page-1) * maxResults).setMaxResults(maxResults);
        q.setParameter("description", sqlStringConverter.convert(description));
        return q.getResultList();
}
```

Aquest mètode es limita a recuperar un subconjunt, delimitat per `maxResults` i `page`, del conjunt de

imatges que respon als criteris de cerca. Fixem-nos que s'inclou una crida al mètode `SqlStringConverter.convert`. Aquest mètode transforma la cadena de text facilitada per l'usuari en un format aplicable a una cerca en la base de dades. S'ha extret aquest mètode en una classe de servei per tal de poder reutilitzar-lo en altres classes.

```
public static String convert (String word) {
    word = word.replace('*', '%');
    if (word.charAt(0) != '%') word = "%" + word;
    if (word.charAt(word.length() - 1) != '%') word = word + "%";
    return word;
}
```

Per tal de que funcionin la paginació i el suport de llengües s'haurà d'afegir el paràmetre `search` a les `URL's` generades automàticament per l'etiqueta `<spring:url>`, ho farem mitjançant l'etiqueta `<spring-Param>`.

```
<c:if test="${not empty param.search}">
      <spring:param name="search" value="${param.search}" />
</c:if>
```

8.7.4.6 Cerca d'entitats imatges per etiquetes.

Una vegada hem decidit que utilitzarem l'`URL` d'una etiqueta per a mostrar un llistat d'imatges relacionades, només ens queda construir els enllaços apropiats en les vistes, per exemple amb el codi:

```
<spring:url value="/etiquetas/${label.id}" var="label_url" />
<a class="etiqueta" href="${label_url}">${label.nomEtiqueta}</a>
```

Aquestes peticions `GET` seran ateses pel mètode `EtiquetaController.show` que mostrem a continuació.

```
    @RequestMapping(value = "/{id}", produces = "text/html")
    public String show(@PathVariable("id") Long id, Model uiModel) {
        uiModel.addAttribute("imatges", Imatge.getImagesByLabel(Etiqueta.findEtiqueta(id)));
        return "imatges/list";    }
```

Aquest mètode és molt senzill, recull les imatges relacionades amb una etiqueta, les afegeix al model i retorna el nom lògic de la vista que mostra un llistat d'imatges.

8.7.4.7 Cerca d'imatges marcades.

La funcionalitat de marcatge no servirà de gran cosa sinó oferim als usuaris una manera ràpida d'accedir a aquelles imatges que han marcat prèviament, per a fer-lo, modificarem lleugerament el mètode `ImatgeController.search` per tal que pugui filtrar cerques segons paraules claus, trobo que aquesta és una bona manera d'aprofitar el codi ja desenvolupat per a oferir noves funcionalitats.

```
if (searchString.equals("MY_MARKED_IMAGES") || searchString.equals("MY_DELETED_IMAGES")) {

      Usuari user = Usuari.getUserByName(SecurityContextHolder...getName());
      List<Imatge> images;

      if (searchString.equals("MY_MARKED_IMAGES")) {
            images = new ArrayList<Imatge>(user.getImatgesMarcades());
      } else {
            images = Imatge.getDeletedImages(user);
      }...
      uiModel.addAttribute("imatges", images);
}...
```

Veiem que quan es rep una certa cadena de caràcters predefinida, enlloc de portar a terme una cerca per paraula clau, recuperem a l'usuari actual del context i trobem les seves imatges marcades, únicament ens queda afegir un enllaç al menú principal que doni accés a aquesta funcionalitat de manera transparent per als usuaris.

```
<menu:item id="i_imatge_marked" messageCode="global_menu_marked_images" url="/imatges/search?
      search=MY_MARKED_IMAGES&amp;page=1&amp;size=${empty param.size ? 10 : param.size}" />
```

8.7.4.8 Cerca d'imatges eliminades.

Per defecte les imatges que un usuari ha eliminat no són visibles, degut a aquest fet, haurem de proporcionar un enllaç a la seva visualització per tal de possibilitar la seva restauració. El mètode encarregat d'atendre aquestes peticions serà `ImatgeController.search`, el codi en concret que porta a terme la tasca es pot veure a l'apartat anterior.

Per a possibilitar l'accés a aquesta funcionalitat, oferirem un enllaç, visible únicament per als usuaris autenticats, al menú principal.

```
<menu:item id="i_imatge_marked" messageCode="global_menu_deleted_images" url="/imatges/search?
      search=MY_DELETED_IMAGES&amp;page=1&amp;size=${empty param.size ? 10 : param.size}" />
```

8.7.4.9 Marcatge d'imatges.

En aquest cas, com que el marcatge no es tracta d'una funcionalitat prevista per roo, ens caldrà crear un nou mètode en el controlador que s'ocupi d'aquest cas, per a mapar les peticions de l'usuari a aquest mètode, em sembla el més apropiat un mapatge principal a l'usuari amb un paràmetre indicatiu de que es tracta d'un marcatge que inclogui el número d'imatge a marcar, quedarà més clar amb un exemple.

```
<spring:url value="/usuaris/${user.id}?markImage=${image.id}" var="mark_url" />
      <form action="${mark_url}" method="POST"><input class="thumbnail_link"
      id="mark_${image.id}" type="submit" value="${mark_message}" /></form>
```

Per exemple, si l'usuari 4 vol marcar l'imatge 7 s'enviarà un `POST` a l'`URL /usuaris/4?markImage=7`. El mètode que processarà aquesta petició és el següent.

```
@RequestMapping(value="/{id}", params = "markImage", method = RequestMethod.POST)
public String markImage(@PathVariable("id") Long id, @RequestParam(value="markImage") Long imageId,
            Model uiModel, HttpServletRequest request) {
      Usuari user = Usuari.findUsuari(id);
      user.mark(Imatge.findImatge(imageId));
      user.merge();
        return "redirect:"+request.getHeader("referer");
    }
```

El mètode recupera l'usuari i l'imatge de la base de dades, genera el marcatge i envia una redirecció a la pàgina anterior per tal de que es refresquin les dades existents.

8.7.4.10 Visualització d'entitats imatge marcades.

Per tal d'aprofitar el codi que ja tenim podem utilitzar el mètode `ImatgeController.search` per tal de mostrar les imatges marcades, utilitzarem una paraula clau, `MY_MARKED_IMAGES` per tal d'indicar a l'aplicació que l'usuari vol veure les seves imatges marcades i no portar a terme una cerca.

Inclourem les següents modificacions en el mètode indicat.

```
if (searchString.equals("MY_MARKED_IMAGES")) {

      Usuari user=Usuari.getUserByName(SecurityContextHolder.getContext()
            .getAuthentication().getName());
      List<Imatge> images = new ArrayList<Imatge>(user.getImatgesMarcades());

      uiModel.addAttribute("imatges", images);
} else {... cercar imatges per paraula clau ...
```

També haurem d'incloure un enllaç, preferentment en el menú principal, per tal de permetre als usuaris accedir a aquesta funcionalitat. Afegim les següents línies a `menu.jsp`.

```
<security:authorize access="isAuthenticated()">...
      <menu:item id="i_imatge_marked" messageCode="global_menu_marked_images"
      url="/imatges/search?search=MY_MARKED_IMAGES&amp;page=1&amp;
            size=${empty param.size ? 10 : param.size}" /></security:authorize>
```

8.7.4.11 Eliminació i restauració d'entitats imatge.

Tal i com es va establir en el disseny de l'aplicació, es dotarà als usuaris d'una funcionalitat, tipus papelera de reciclatge, per tal que les eliminacions de ítems publicats no sigui irreversible. Amb tal fi modificarem el mètode `ImatgeController.delete` de dues maneres, d'una banda, quan s'hagi de portar a terme una eliminació, enlloc de eliminar la imatge de la base de dades, la marcarem com inactiva.

D'una altra banda, aquest mateix mètode serà l'encarregat de restaurar una imatge, quan es cridi aquest mètode, la imatge que es passa com a paràmetre veurà el seu estat d'activitat invertit.

```
    @RequestMapping(value = "/{id}", method = RequestMethod.DELETE, produces = "text/html")
    public String delete(@PathVariable("id") Long id,
            @RequestParam(value = "page", required = false) Integer page,
            @RequestParam(value = "size", required = false) Integer size, Model uiModel,
            HttpServletRequest request) {
        Imatge imatge = Imatge.findImatge(id);
        imatge.setActiu(!imatge.getActiu());
        imatge.merge();
        return "redirect:"+request.getHeader("referer");
    }
```

Per a cridar aquest mètode, el qual respon a peticions tipus `DELETE`, utilitzarem una tècnica que ja hem estudiat anteriorment. Ho podem veure a continuació.

```
<spring:message code="restore" var="restore_message" />
<spring:url value="/imatges/${imatge.id}" var="delete_url" />
<form id="command" action="${delete_url}" method="POST">
      <input type="hidden" value="DELETE" name="_method" />
      <input class="button" type="submit" value="${restore_message}" /></form>
```

Alternativament, també podríem utilitzar els *tags jsp* que ens proporciona el bastiment *Spring*[28] amb aquesta mateixa finalitat de la manera següent.

```
<form:form method="delete" ... ><input type="submit" value="..." /></form:form>
```

8.7.5 Compra.

Les entitat compra difereixen de la resta en que no són creades directament per els usuaris sinó que la seva creació es un subproducte de l'execució d'un acció tipus compra per part d'un usuari sobre una imatge determinada.

28 Es pot trobar més explicació sobre la construcció d'aplicacions segons les convencions REST a [SFRD12] pàgina 485.

Les altres etapes del seu cicle vital també evolucionen d'una manera diferent que la de la resta d'entitats, per exemple, una vegada creades no poden ser eliminades i el seu pas a l'estat confirmat es produeix una quan l'usuari porta amb èxit el pagament de la seva cistella.

8.7.5.1 Creació de noves compres.

Tal i com ja s'ha avançat, la creació de noves compres respon a una acció de compra de part d'un usuari, que es troba autenticat davant del sistema, sobre una imatge determinada. La petició que el client enviarà al servidor serà un `POST` a l'`URL` `/compras?image=id` on `id` serà l'identificador de la imatge determinada.

El mètode `CompraController.addToBasket` que s'ocupa de processar aquesta petició és el següent.

```
	@RequestMapping(params = "image", produces = "text/html")
	public String addToBasket (@RequestParam(value="image") Long id, HttpServletRequest request){

		ShoppingTrolley basket = null;
		HttpSession session = request.getSession();

		basket = getBasket(basket, session);

		Imatge image = Imatge.findImatge(id);
		Compra compra = new Compra(basket.getUsuari(), image);
		compra.persist();
		basket.getCompres().add(compra);
		session.setAttribute("basket", basket);

		return "redirect:"+request.getHeader("Referer");
	}
```

Aquest mètode delega en un mètode accessori privat per tal d'obtenir la cistella de la compra, si existeix una s'utilitzarà aquesta, altrament es crearà una de nova i es declararà com un atribut de la sessió i posteriorment afegeix a aquesta cistella una nova compra que reflecteix l'usuari comprador (recuperat de la cistella) i la imatge afegida.

Com que la compra d'imatges es pot produir des de diferents llocs de l'aplicació, retornarem el flux a la pàgina des de la qual s'ha realitzat la petició.

Sembla evident que seria molt més adequat portar a terme tota aquesta funcionalitat amb crides AJAX, ho he intentat i, tot i que el funcionament general del sistema no em resulta problemàtic, és a dir, puc sense problemes enviar les peticions i rebre els resultats mitjançant crides AJAX, no he pogut reflectir els canvis produïts en els llocs adequats mitjançant *JavaScript*, amb el qual he hagut d'implementar aquesta funcionalitat mitjançant crides i redireccionaments.

8.7.6 Anunci.

L'entitat `Anunci` és molt semblant a l'entitat `Imatge`, de fet, al disseny ja s'havia definit una superclasse abstracta `Publicable`, de les quals les dues heretaven. Finalment, després de portar a terme algunes proves, no s'ha implementat d'aquesta manera, i per tant s'ha introduït un cert grau de redundància en el codi, per **trobar que el suport que roo ofereix a les interfícies i l'herència no estava del tot madur**.

Aprofitarem les semblances entre aquestes dues entitats per a copiar alguns patrons de disseny utilitzats en `Imatge` amb `Anunci`.

8.7.6.1 Creació d'entitats Anunci.

A l'etapa de disseny vam determinar que els anuncis poden ser publicats únicament per usuaris regis-

trats, per a assegurar-nos que aquest és el cas, utilitzarem els *tags* que ens proporciona *SpringSecurity* per a mostrar l'enllaç de creació de manera condicional.

```
<security:authorize access="isAuthenticated()">
      <menu:item id="i_anunci_new" messageCode="global_menu_new" url="/anuncis?form"
            z="hbiiCGiwdy7+y3sSwwls4WjjdN4=" /></security:authorize>
```

Si l'usuari clica en aquest enllaç, la petició enviada serà atesa pel mètode `AnunciController.createForm`, aquest mètode és bàsicament anàleg al que podem veure en 8.7.4.1 amb el qual no detallarem, les úniques diferències seran que enlloc d'afegir un llistat d'etiquetes, el que afegim és un llistat de categories.

La pàgina `jsp` que mostra aquest formulari i el mètode que processa les dades enviades per l'usuari són molt semblant als que s'ocupen de processar la creació d'una imatge, sense haver de processar les dades binàries corresponents a l'arxiu d'imatge amb el qual utilitzem un formulari normal. Es poden veure els detalls de la creació d'entitats imatge a 8.7.4.1 i els de creació d'anuncis al codi font de l'aplicació.

8.7.6.2 Edició d'entitats anunci.

També en aquest cas l'edició d'anuncis s'implementarà d'una manera molt semblant a l'edició d'imatges, sense tenir en compta, per descomptat, el processament del arxiu binari que és la imatge mateixa. Per tal d'obtenir informació sobre l'edició d'entitats imatges es pot consultar l'apartat 8.7.4.2 d'aquest mateix document, per a obtenir informació detallada sobre com s'ha implementat l'edició d'anuncis, es pot consultar el codi font de l'aplicació.

8.7.6.3 Visualització d'entitats anunci.

Per a visualitzar els anuncis, com ja és habitual, ens quedarem amb els atributs que ens interessen i no mostrarem els que no seran rellevants per als usuaris finals. Els més interessant d'aquesta funcionalitat és que, tal i com ja vam veure en la visualització d'una imatge, mostrarem enllaços de manera condicional segons si l'usuari es troba autenticat davant del sistema, si es tracta del creador de l'anunci i si aquest es troba actiu o no en el sistema.

El codi que mostra o no els camps es simple codi condicional utilitzant els *tags* estàndards que ens ofereix l'especificació dels *Servlets*, pot ser interessant però, mostrar la manera d'obtenir la informació sobre l'estat de l'usuari i la seva relació amb l'anunci.

```
<c:choose>
      <c:when test="${pageContext['request'].userPrincipal != null}">
            <security:authentication property="principal.username" var="userName" />
            <c:set value="${userName == anunci.usuari.nom}" var="isPublisher" />
            <c:set value="true" var="is_registered_user" />
            <spring:eval expression="T(cat.imatgina.domain.Usuari).getUserByName(userName)"
                  var="user" /> </c:when><c:otherwise>
                  <c:set value="false" var="isPublisher" />
                  <c:set value="false" var="is_registered_user" />
            </c:otherwise></c:choose>
```

La funcionalitat que ofereixen la major part dels enllaços es troba explicada en els diferents apartats d'aquesta entitat, excepte la funcionalitat de respondre a l'anunci, la qual es troba explicada a l'apartat 8.7.8.1 .

Per tal d'agilitzar la navegació per els diferents anuncis, oferirem un llistat de categories a les quals pertany l'anunci visualitzat en format d'enllaços.

```
<c:forEach items="${anunci.categories}" var="cat">
```

```
<spring:url value="/categorias/${cat.id}" var="cat_url" />
<a href="${cat_url}"><c:out value="${cat.nom}" /></a><c:out value=", " /></c:forEach>
```

Quan un usuari seleccioni un enllaç es veurà presentat amb un llistat dels anuncis categoritzats segons aquesta. El mètode que processa aquesta petició es va explicar a l'apartat 8.7.2.3 .

8.7.6.4 Visualització de múltiples entitats anunci.

En aquest ens serà bastant convenient l'estructura tabular que ens ofereix roo.

Il·lustració 12: Visualització de múltiples entitats anunci.

Introduirem, com sempre, algunes modificacions per a adaptar la vista i les funcionalitats ofertes a les necessitats de l'aplicació. Les modificacions més significatives seran:

- Presentació del nom de l'anunci com un enllaç que ens permet accedir a la visualització d'aquest.
- Presentació dels noms de les categories dels anuncis com enllaços que ens porten a visualitzar una relació dels anuncis inclosos en la categoria seleccionada.
- Existència d'un enllaç que ens permet enviar una resposta al creador de l'anunci.
- Existència d'un espai funcional que ens permet portar a terme la resta d'opcions existents sobre un anunci, aquestes són funcionalitats condicionals i es mostrarà una u altra en funció de la relació entre l'anunci i l'usuari autenticat davant del sistema.

La funcionalitat d'aquests enllaços és la mateixa que es va detallar a l'apartat anterior.

8.7.6.5 Cerca d'entitats anunci.

A l'igual que en el cas de les imatges, el primer pas serà oferir un enllaç per tal de possibilitar la cerca d'anuncis en el menú principal.

```
<menu:item id="i_anunci_search" messageCode="global_menu_search" url="/anuncis/searchForm" />
```

Haurem de crear també el mètode `AnunciController.searchForm` que servirà aquesta petició.

```
@RequestMapping(value = "/searchForm", produces = "text/html")
public String showSearchForm () {
      return "anuncis/search";
    }
```

S'haurà d'associar aquest nom lògic a alguna tecnologia capaç de oferir la implementació, en el nostre cas a la jsp corresponent.

```
<definition extends="default" name="anuncis/search">
        <put-attribute name="body" value="/WEB-INF/views/anuncis/search.jspx"/>
    </definition>
```

I s'haurà de crear el formulari que recollirà la cadena de text de l'usuari i l'enviarà com a paràmetre de la petició `HTTP GET`.

```
<DIV id="anunci_search_description_like" class="white_panel">
      <spring:url value="/anuncis/search" var="form_url" />
      <form:form action="${form_url}" method="GET">
            <h1><spring:message code="search_anuncis_by_keyword" /></h1>

            <spring:message code="anunci_search_text" var="${search_keyword}" />
            <LABEL for="search">${search_keyword}</LABEL>
            <input type="hidden" name="size" value="${empty param.size ? 20 : param.size}" />
            <input type="hidden" name="page" value="1" />
            <input name="search" type="text" placeholder="Search keyword" />
            <input type="submit" class="button" />
      </form:form></DIV>
```

El mètode que s'ocuparà de portar a terme la cerca serà anàleg al de l'entitat `Imatge`.

```
@RequestMapping(value= "/search")
public String search(
      @RequestParam(value = "search", required = true) String searchString,
      @RequestParam(value = "page", required = false) Integer page,
      @RequestParam(value = "size", required = false) Integer size,
      Model uiModel) {
      if (page != null || size != null) {
            int sizeNo = size == null ? 10 : size.intValue();
            List<Anunci> anuncis = Anunci.searchAnunciByDescription
                  (searchString, sizeNo, page.intValue());
            uiModel.addAttribute("anuncis", anuncis);
            long anuncisReturnedByQuery =  Anunci.countAnuncisWithDescription(searchString);
            float nrOfPages = (float) anuncisReturnedByQuery / sizeNo;
            uiModel.addAttribute("maxPages", (int) ((nrOfPages >
                  (int) nrOfPages || nrOfPages == 0.0) ? nrOfPages + 1 : nrOfPages));

      } else {
            uiModel.addAttribute("anuncis", Anunci.searchAnunciByDescription(searchString, 20, 1));
      }
      addDateTimeFormatPatterns(uiModel);
      return "anuncis/list";
}
```

I també ens caldrà un mètode que contarà el nombre de resultats de la cerca.

```
public static long countAnuncisWithDescription (String description) {
     return entityManager().createQuery("SELECT COUNT(o) FROM Anunci AS o " +
                  "WHERE (LOWER(o.description) LIKE LOWER(:description)  " +
                  "OR LOWER(o.nom) LIKE LOWER(:description))" +
                  "AND o.actiu = true", Long.class).setParameter("description",

      cat.imatgina.services.SqlStringConverter.convert(description)).getSingleResult();
}
```

A més, a l'igual que s'ha fet amb les imatges, també suportarem la cerca d'anuncis eliminats prèviament, amb la idea de restaurar-los, i la cerca d'anuncis que s'hagin marcat. La funcionalitat de marcatge s'explica més detalladament en l'apartat següent.

8.7.6.6 Marcatge d'entitats anunci.

Els anuncis es podran marcar des dels enllaços existents, tan en la visualització d'un llistat d'anuncis com des de la visualització d'un únic. Qualsevol d'aquests enllaços enviarà un `POST` a, segons el cas, `/usuaris/id?markAnunci=anId` o `/usuaris/id?unmarkAnunci=anId` on `id` serà l'identificador de l'usuari que envia la petició i `anId` serà l'identificador de l'anunci que es vol marcar o desmarcar. Veiem un fragment de codi d'exemple.

```
<spring:url value="/usuaris/${user.id}?markAnunci=${anunci.id}" var="mark_url" />
      <spring:message code="mark_anunci" var="mark_message" />
      <form action="${mark_url}" method="POST"><input type="submit" value="${mark_message}"
            id="mark_${anunci.id}" class="button" /></form>
```

El mètode `UsuariController.mark` (`UsuariController.unMark` respectivament) serà l'encarregat de processar la petició, el mostrem a continuació.

```
@RequestMapping(value="/{id}", params = "markAnunci", method = RequestMethod.POST)
public String markAnunci(@PathVariable("id") Long id, @RequestParam(value="markAnunci") Long
            anunciId, Model uiModel, HttpServletRequest request) {
      Usuari user = Usuari.findUsuari(id);
      user.mark(Anunci.findAnunci(anunciId));
      user.merge();
      return "redirect:"+request.getHeader("referer");
    }
```

Es tracta d'un mètode molt senzill, recupera l'usuari i l'anunci, afegeix aquest darrer a la llista d'anuncis marcats del primer. Després utilitza el camp *referer*, per a mostrar la mateixa pàgina des de la qual s'ha enviat la petició.

8.7.6.7 Eliminació d'entitats anunci.

A l'igual que en el cas de les imatges, no permetrem l'eliminació total dels anuncis publicats als usuaris de l'aplicació, en el seu lloc, oferirem una funcionalitat que els permetrà marcar els anuncis com inactius, amb el qual no podran ser visualitzats per altres usuaris, tot i permetent al creador, en un moment posterior, la seva restauració si ho veu convenient.

L'enllaç que oferirà accés a aquesta funcionalitat es mostrarà, de manera condicional, en les vistes d'anunci i llistat d'anunci i enviarà un `DELETE`, mitjançant el patró definit anteriorment, a l'`URL /anuncis/id` on `id` serà l'identificador de l'anunci que es vol marcar com inactiu.

El mètode que s'ocupa de processar aquestes peticions serà el mateix que processi les peticions de restauració i es mostra a l'apartat següent.

8.7.6.8 Restauració d'entitats anunci eliminades.

La restauració d'anuncis es considera com l'acció oposada a l'eliminació i, per aquest motiu, el mètode `AnunciController.delete`, ocupat de processar-la, serà el mateix que el que s'ocupa de les eliminacions, la diferència la marcarà l'estat actual d'activitat de l'anunci sobre el qual es realitza la crida, en qualsevol dels dos casos s'inverteix el seu estat d'activitat. Mostrem el codi en qüestió.

```
@RequestMapping(value = "/{id}", method = RequestMethod.DELETE, produces = "text/html")
public String delete(@PathVariable("id") Long id,
            @RequestParam(value = "page", required = false) Integer page,
            @RequestParam(value = "size", required = false) Integer size,
            HttpServletRequest request, Model uiModel) {
            Anunci anunci = Anunci.findAnunci(id);
            anunci.setActiu(!anunci.getActiu());
            anunci.merge();
            return "redirect:"+request.getHeader("Referer");
      }
```

El funcionament del mecanisme que permet enviar una petició `DELETE`, tot i que `HTTP` únicament suporta `GET` i `POST`, s'ha detallat anteriorment[29].

29 Veure [SFRD12] pàgina 485 per a una explicació detallada del funcionament i l'apartat 8.7.1.2 d'aquest mateix document per a un exemple d'utilització en el context d'Imatgina.

8.7.7 Factura.

A l'igual que les compres, les factures són un tipus d'entitat transparent per als usuaris, serà l'aplicació l'encarregada de crear-les i processar-les internament en funció d'esdeveniments sobre altres elements del model.

Es tracta, en aquest cas d'una entitat molt simple que serà generada, en cas d'un pagament processat correctament, amb un set de compres obtingut del carro de la compra d'un usuari. La seva funció és mantenir un registre de les compres que es van portar a terme com una unitat, i, si escau, poder generar i enviar factures detallades als usuaris de l'aplicació.

Una altre utilitat que podem imaginar que tindran, en un futur, és servir com a base de treball a un possible departament de comptabilitat que s'ocupi de les finances d'Imatgina.

8.7.7.1 Creació d'entitats factura.

Com s'ha vist, les entitats factura es generen com a part del procediment de *checkout* portat a terme per un usuari, en concret, la crida al mètode que s'ocupa de processar el pagament[30] es realitza en l'estat d'acció[31] `process_payment` del flux `checkout`. Veiem la definició de l'estat.

```
<action-state id="process_payment">
     <evaluate expression="flowRequestContext.externalContext.sessionMap.basket.proccessPayment()"/>
      <transition to="show_exception_cause”
      on-exception="cat.imatgina.services.PaymentProcessingException" />
      <transition to="show_confirmation" />
</action-state>
```

Aprofitem que l'atribut de sessió `basket` ja té una referència a l'usuari, i per tant als seus detalls de pagament, i a totes les compres que s'han de processar per a poder invocar el seu mètode `ShoppingTrolley.proccessPayment()` sense paràmetres. Aquest mètode és l'encarregat de crear una nova entitat factura i afegir-hi totes les compres, una a una, després de marcar-les com a confirmades. Veiem-ho.

```
public boolean proccessPayment () throws PaymentProcessingException {

      // UserPaymentDetails pd = this.getUsuari().getPaymentDetail();
      // Here we would process the payment

      // Create a new bill, set the date and the user
      Factura fac = new Factura();
      fac.setDataFactura(new Date());
      fac.setUsuari(this.getUsuari());

      // Populate it with the items bought.
      for (Compra com : this.getCompres()) {
            com.setConfirmada(true);
            fac.getCompres().add(com);
            fac.setTotalFactura(fac.getTotalFactura() + com.getImatge().getPreu());
      }

      fac.persist();

      emptyBasket ();
      return true;
}
```

Aquest mètode no es troba complet ja que li falta processar el pagament, aquest processament es podrà incloure de manera senzilla, una vegada decidits els tipus de pagament que s'accepten i arribats a un acord amb les entitats que s'ocuparan del processament.

30 Recalcar en aquest punt que la funcionalitat de pagament no s'ha implementat.
31 Una descripció completa del l'estat d'acció es pot trobar a [SWFD12] pag 37.

El que si que podem veure és com es crea una nova factura i es van afegint les compres, recuperades del carro de la compra, a mesura que es van marcant com confirmades i es va calculant el valor total afegit de la suma d'ítems que l'usuari compra.

A posteriori, únicament ens haurem de preocupar de persistir la factura en la base de dades. Donat que hem definit les factures com objectes immutables, no podrem portar edicions ni eliminacions sobre aquesta entitat. Per a eliminar tots el codi innecessari que *SpringRoo* ha generat per a nosaltres en relació amb aquesta entitat podem, simplement, eliminar la classe `FacturaController` i *roo* s'ocuparà de portar a terme la neteja per nosaltres.

8.7.8 Resposta.

Com s'ha comentat, els usuaris podran respondre anuncis publicats per altres usuaris, una vegada respostos, aquestes respostes no seran modificables ni eliminables, amb el qual, a l'igual que va ser el cas amb les Factures, únicament ens preocuparem del cas de creació.

8.7.8.1 Creació d'entitats resposta.

La creació d'una resposta comença quan un usuari selecciona l'enllaç corresponent en la vista de visualització d'un, o diversos, anuncis[32]. Aquest enllaç les hi presentarà un formulari per a crear un anunci molt senzill; es visualitzen els detalls de l'anunci que s'està responent i hi ha un espai per a introduir la resposta que es vol enviar.

Il·lustració 13: Vista de creació d'una resposta.

El mètode que s'encarrega de processar aquest formulari és `RespostaController.create`, el mostrem.

```
    @RequestMapping(method = RequestMethod.POST, produces = "text/html")
    public String create(@Valid Resposta resposta, BindingResult bindingResult,
             Model uiModel, HttpServletRequest httpServletRequest) {
        uiModel.asMap().clear();
        resposta.persist();

        // Construct message
        StringBuffer sb = new StringBuffer();
              ...

        String mailTo = resposta.getAnunci().getUsuari().getEmail();
        String message = sb.toString();
        notificationService.sendMessage(mailTo, message);
        return "redirect:/";
```

32 La visualització d'un anunci es troba detallada a l'apartat 8.7.6.3 i la visualització de diversos a l'apartat 8.7.6.4 .

```
    }
```

Per brevetat s'han eliminat algunes parts del mètode no rellevants per a la discussió en curs, la part que ens interessa és l'enviament d'un correu electrònic mitjançant l'ús del mètode `notificationService.sendMessage()`. `NotificationService` és una interfície que hem definit al paquet serveis que defineix el mètode `sendMessage`, la seva implementació, `NotificationServiceImpl` ha estat definida, en el mateix paquet com un `@Component`. El codi corresponent a aquesta classe és el següent.

```
@Component
public class NotificationServiceImpl implements NotificationService {

    @Autowired
    private transient MailSender mailTemplate;

    @Autowired
    private transient SimpleMailMessage templateMessage;

    public void sendMessage(String mailTo, String message) {
        org.springframework.mail.SimpleMailMessage mailMessage =
             new org.springframework.mail.SimpleMailMessage(templateMessage);
        mailMessage.setTo(mailTo);
        mailMessage.setText(message);
        mailTemplate.send(mailMessage);
    }
}
```

Com que aquesta classe es troba definida com un component i tenim l'`autoScan` activat, podem injectar-lo en `RespostaController` d'una manera molt senzilla.

```
      @Autowired
      private NotificationService notificationService;
```

En l'arxiu `applicationContext.xml` hem afegit les següents definicions de components.

```
    <bean class="org.springframework.mail.javamail.JavaMailSenderImpl" id="mailSender">
        <property name="host" value="${email.host}"/>
        <property name="protocol" value="${email.protocol}"/>
        <property name="port" value="${email.port}"/>
        <property name="username" value="${email.username}"/>
        <property name="password" value="${email.password}"/>
        <property name="javaMailProperties">
            <props>
                <prop key="mail.smtp.auth">true</prop>
                <prop key="mail.smtp.starttls.enable">true</prop>
            </props>
        </property>
    </bean>
    <bean class="org.springframework.mail.SimpleMailMessage" id="templateMessage">
        <property name="from" value="${email.from}"/>
        <property name="subject" value="${email.subject}"/>
    </bean>
```

I en l'arxiu de propietats `email.properties` hem definit els paràmetres necessaris per a l'enviament de correus electrònics, tals i com el servidor `SMTP`, i nom d'usuari i contrasenya que s'utilitzen.

8.8 Funcionalitat de compra d'imatges.

Per a donar suport a aquesta funcionalitat ens caldrà implementar dues funcionalitats, una que permetrà als usuaris indicar que volen afegir un ítem al seu carro de la compra i una altra que els permetrà, una vegada satisfets amb el contingut d'aquest, procedir a l'etapa de pagament.

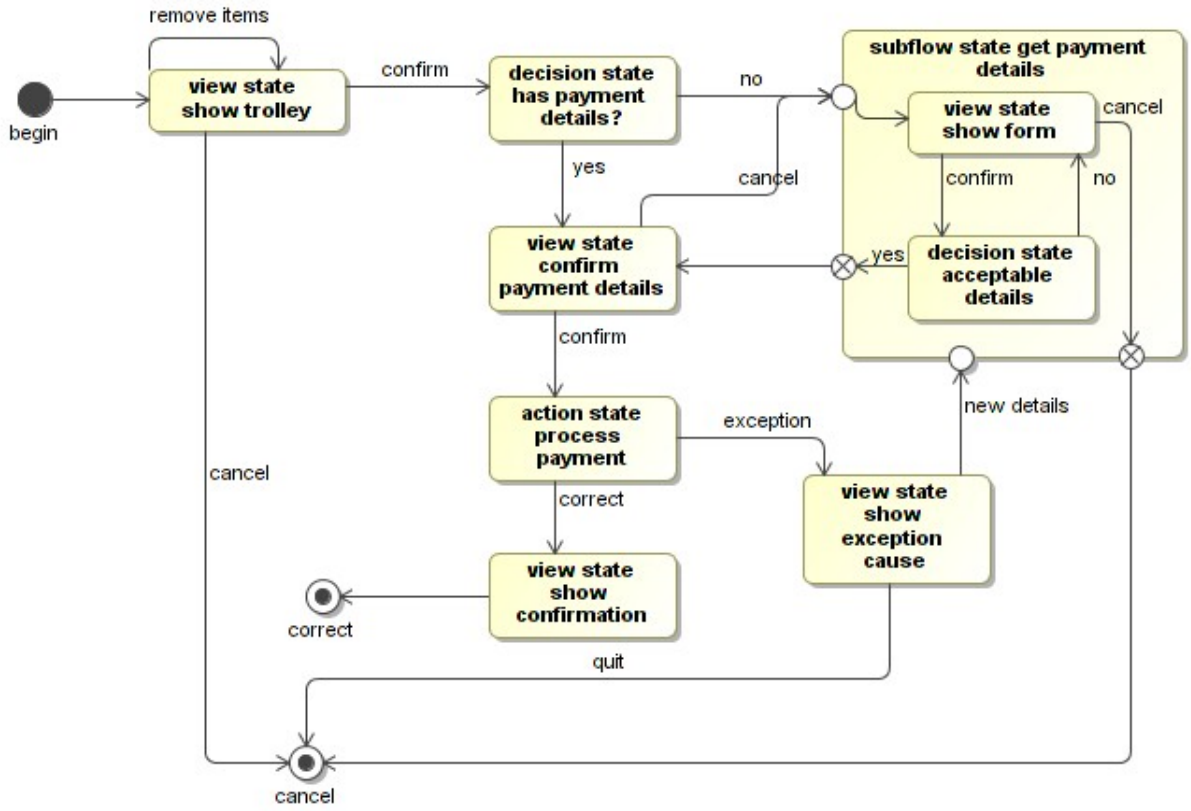
La primera de les dues s'ha vist a l'apartat 8.7.5.1 Creació de noves compres. serà per tant en la segona,

el procés de pagament, en la que ens centrarem en el següent apartat.

8.8.1 Procediment de *checkout.*

A l'hora de processar un pagament, serà convenient guiar a l'usuari a través de les respectives pantalles que s'hauran de visitar enlloc de deixar que aquest navegui lliurement i, com a conseqüència, es pugui perdre o no sàpiga com continuar.

En aquesta tasca ens serà de molta ajuda la tecnologia que ens ofereix *Spring Web Flow*[33], la qual ens permetrà definir de manera declarativa els diferents estats per els quals podrem passar i les transicions entre aquests. Per tal de fer-nos una idea general de l'estructura del flux a desenvolupar ens serà útil crear amb antelació un pseudodiagrama de estats[34], a continuació mostro el diagrama utilitzat.



Il·lustració 14: Diagrama d'estats del procediment de checkout.

Per a començar a treballar amb SWF podem utilitzar l'eina roo, la qual, tal i com ja estem acostumats, porta a terme una gran part de la configuració per nosaltres. Des de la consola roo executem la següent comanda.

```
web flow --flowName checkout
```

Amb el qual roo afegeix totes les dependències necessàries al projecte, i genera un flux d'exemple sobre el qual ens podem posar a treballar.

33 Podem trobar més informació sobre *Spring Web Flow* a [SWFD12] i la documentació en línia aquí.

34 Es tracta d'un pseudodiagrama ja que no s'utilitzen el llenguatge UML correctament sinó que utilitzem els estats en el sentit que les hi dona SWF, és a dir, tenim cinc tipus d'estat; d'acció, de decisió, de vista, subfluxos i finals. Una bona referència, un tant menys feixuga que la documentació, és [SIA11] pàgines 203-204.

El primer pas[35] serà generar el fitxer de definició de flux principal.

```
<flow ...schema definitions here... >

	<view-state id="show_basket" view="checkout/show_basket">
		<transition on="success" to="check_user_payment_details" />
		<transition on="cancel" to="cancel" />
	</view-state>

	<decision-state id="check_user_payment_details">
		<if	test="flowRequestContext.externalContext.sessionMap.user.hasPaymentDetails()"
			then="confirm_details" else="get_payment_details" />
	</decision-state>

	<view-state id="confirm_details" view="checkout/confirm_details">
		<transition on="change" to="get_details" />
		<transition on="accept" to="process_payment" />
	</view-state>

	<action-state id="process_payment">
		<evaluate expression=
			"flowRequestContext.externalContext.sessionMap.basket.proccessPayment()" />
		<transition to="show_exception_cause"
			on-exception="cat.imatgina.services.PaymentProcessingException" />
		<transition to="show_confirmation" />
	</action-state>

	<view-state id="show_exception_cause">
		<transition on="success" to="get_details" />
		<transition on="cancel" to="cancel" />
	</view-state>

	<subflow-state id="get_details" subflow="getDetails">
		<transition on="payment_details_ok" to="confirm_details" />
		<transition on="cancel" to="cancel" />
	</subflow-state>

	<end-state id="show_confirmation" view="checkout/show_confirmation" />

	<end-state id="cancel" view="checkout/cancel" />

	<global-transitions>
		<transition on="cancel" to="cancel" />
	</global-transitions>
</flow>
```

Aquest fitxer té poc d'especial, bàsicament es tracta de una transcripció directa del diagrama que s'havia realitzat anteriorment.

Per a fer-lo possible, s'hauran d'enviar les dades de pagament de l'usuari a aquest mètode, això no representarà cap problema ja que podem obtenir l'usuari de la variable `user`, que té *scope* de sessió, i recuperar els detalls de pagament d'aquesta variable.

Tampoc s'ha desenvolupat tota la lògica relacionada amb el pagament, la classe `PaymentDetails.java` s'ha posat enlloc únicament per a oferir una interfície a les crides que porta a terme el procediment de pagament, a les quals sempre respon de la mateixa manera. Per contra si que s'ha desenvolupat completament la part relacionada amb l'usuari, si Imatgina no té detalls de pagament registrats per a aquest, se li demanarà de facilitar-los, com que no hem definit el tipus de pagament que acceptem, l'únic que l'usuari haurà de fer és acceptar i continuar en aquesta vista, veurem els detalls a continuació.

El proper pas serà definir el subflux que hem anomenat `get_payment_details`, que s'ocupa, en cas que al sistema no hi consten detalls de pagament de l'usuari, de presentar un formulari per a recollir-los, validar-los i retornar-los al flux principal.

35 Per a treballar amb el bastiment *Spring Web Flow* m'he basat en la referència [SWFD12] i en [SIA11] capítol 8.

```
<flow ... schema definitions go here ...>
	<view-state id="show_form" view="checkout/form">
		<transition on="success" to="payment_details_ok" />
	</view-state>

	<decision-state id="check_payment_details">
		<!-- Placeholder for some detail-checking logic -->
		<if then="payment_details_ok" test="true"
			else="show_form" />
	</decision-state>

	<end-state id="payment_details_ok" />

	<end-state id="end_state" />

	<global-transitions>
		<transition on="cancel" to="end_state" />
	</global-transitions>
</flow>
```

En aquest cas també tenim un codi sense funcionalitat, concretament en l'estat de decisió `check_payment_details`, per tal de permetre'ns integrar la lògica ocupada d'aquesta funció d'una manera fàcil quan es desenvolupi en un moment posterior.

Aquest estat de decisió tornarà l'execució del flux a l'estat de vista `show_form` mentre que el detalls introduïts per aquest no siguin vàlids. L'usuari sempre podrà escapar aquest bucle optant-hi per l'opció `cancel·lar`, la qual el portarà a l'estat final cancel del flux principal.

9 Proves.

Al disseny es va especificar que s'aprofitaríem les característiques de *SpringRoo* per a generar de manera automàtica els tests d'integració de les entitats de la nostra aplicació. Per a assolir aquest propòsit, ens cal únicament passar el paràmetre `--testAutomatically` en la creació d'entitats mitjançant el *shell* que ens ofereix *roo*.

```
entity jpa --class ~.domain.Usuari --table USUARI --testAutomatically
```

Per a portar a terme les proves, podem utilitzar el mateix *shell* que ens ofereix *SpringRoo*, l'eina *maven*, en la qual delega *roo* en qualsevol cas, o el nostre IDE favorit, des del qual podem invocar els tests Junit. Veiem el resultat d'executar els tests des del *shell* que ens ofereix *roo*.

```
roo> perform tests
Tests run: 81, Failures: 0, Errors: 0, Skipped: 0
~.web roo>
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 35.984s
[INFO] Finished at: Sat Jun 16 11:21:09 CEST 2012
[INFO] Final Memory: 11M/206M
[INFO] ------------------------------------------------------------------------
~.web roo> ~.web roo>
```

Per a afegir aquests tests a l'aplicació, *roo* genera, per a cada entitat persistent de la nostra aplicació, dues classes en el paquet `/src/test/java/cat/imatgina/domain`, `DataOnDemand` classes, encarregades de generar entitats per a portar a terme els tests i `IntegrationTest` classes, les quals contenen la lògica encarregada de portar a terme els tests.

En realitat en els fitxers java corresponents a aquestes classes no trobarem codi, sinó que aquests es trobarà, com és habitual amb les classes auto-generades per roo, en ITDs, i serà injectat en temps de com-

pilació. Veiem un exemple d'un test que comprova que la compta d'usuaris.

```
@Test
public void UsuariIntegrationTest.testCountUsuaris() {
    Assert.assertNotNull("Data on demand for 'Usuari' failed to initialize correctly",
dod.getRandomUsuari());
    long count = Usuari.countUsuaris();
    Assert.assertTrue("Counter for 'Usuari' incorrectly reported there were no entries", count > 0);
}
```

És l'anotació `@test` la que informa a `Junit`[36] que aquest mètode s'ha d'executar com un test.

Per manca de temps no s'han pogut portar a terme les proves amb la interfície d'usuari, proves de funcionalitat i proves d'accessibilitat abans de la data d'entrega, totes aquestes queden pendents.

Seria convenient així mateix portar a terme tot una redefinició de la interfície d'usuari, portada a terme per un equip de professionals d'aquest camp, abans de decidir publicar l'aplicació i fer-la accessible als usuaris finals.

10 Nivell de compleció del projecte.

Dels 17 cassos d'ús definits en l'etapa de disseny hi ha dos que no s'han implementat completament.

El **cas d'ús *checkout***; he posat en funcionament tota la lògica estructural del procés però no s'ha implementat cap funcionalitat específica que sigui capaç de processar un pagament per part d'un usuari. Per tal de fer-lo possible, s'hauria d'implementar correctament la classe `UserPaymentDetailsImpl`, o, fins i tot, si volem facilitar el pagament als usuaris, generar altres subclasses de la interfície `UserPaymentDetails` que s'adaptin a diferents maneres de pagament. Calculo un o dos dies de treball per a portar a terme aquest desenvolupament.

Una vegada desenvolupat la lògica ocupada de processar el pagament, s'hauria d'integrar en l'aplicació, aquest pas consistiria simplement en recollir els resultats de la crida a `basket.proccesPayment()`.

Tampoc no estic satisfet amb el disseny visual de la pàgina on es mostren els ítems comprats, s'hauria de modificar però encara no he decidit com fer-lo. En general, l'aplicació hauria de ser revisada per un equip de disseny abans de la seva posta en funcionament, llavors serien ells els encarregats de solucionar aquest punt.

El **cas d'ús manteniment**; després de pensar-lo he considerat que aquest cas d'ús trobarà una implementació més adequada com a un procediment intern[37] de la base de dades que com a una funcionalitat de l'aplicació. En tal cas, el seu desenvolupament s'haurà de portar a terme una vegada hem decidit l'entorn en el qual desplegarem l'aplicació per a la seva posta en funcionament.

Veient aquestes estadístiques, em considero satisfet amb el treball portat a terme. Tot i això no oferiré una valoració percentual del nivell de compleció assolit perquè, des del meu punt de vista, encara hi ha molts punts que es poden millorar.

Suposo que aquest és un punt comú en el desenvolupament de qualsevol tipus d'aplicació però, en el meu cas, amb la implementació de cada funció, han anat sorgint noves idees de funcionalitats que es podrien incorporar o altres que es podien modificar per a millorar l'experiència d'usuari amb el qual l'abast del projecte es troba en un constant creixement.

Per tal de mantenir la coherència m'han sigut d'una ajuda incomparable els documents portats a terme durant l'etapa de disseny, sense aquests crec que m'hagués passat tot el semestre intentant donar forma

36 Més informació sobre Junit es pot trobar a http://www.junit.org/

37 És a dir, com a un *stored procedure* desenvolupat i inserit a la base de dades per nosaltres.

al primer cas d'ús sobre el qual hagués començat a treballar, i a totes les noves idees derivades d'aquest.

Els tests d'unicitat i d'integració automàticament creats per roo ens han permès comprovar el bon funcionament de l'aplicació d'una manera eficient, sense haver de dedicar un esforç massa significant a desenvolupar les classes de prova, per contra, considero les proves d'usabilitat i accessibilitat insuficients i l'àrea de la interacció amb l'usuari s'ha de treballar i millorar abans de considerar la qualitat del producte satisfactòria.

11 Conclusions.

La utilització de plataformes de desenvolupament ràpid, com *SpringRoo* poden ser de molta ajuda a l'hora de millorar la productivitat dels desenvolupadors, ja que ens alliberen de una càrrega de treball considerable i repetitiva normalment necessària a l'hora de començar a implementar una nova aplicació.

En certa manera, podem considerar una eina com *SpringRoo* una evolució del concepte de plantilles, o com un intent de portar la reutilització de codi, patrons i paradigmes de disseny, un pas més enllà, amb l'aplicació de les millors pràctiques de disseny recopilades fins al moment actual.

Tots i aquestes avantatges, considero que la utilització d'aquest tipus d'eina pot no ser el més apropiat per a portar a terme un procés d'aprenentatge com hauria de ser el desenvolupament d'un treball de final de carrera, el motiu és que l'eina ens permet de treballar amb un nivell d'abstracció molt elevat, sense per força comprendre les accions que tenen lloc interiorment, i per tant, sense entendre realment l'arquitectura de l'aplicació que estem desenvolupant.

Aquest fet no seria un problema, de fet tota la informàtica està basada en la construcció d'abstraccions que ens permeten no haver de conèixer les capes inferiors sobre les quals treballem, sinó fos perquè l'abstracció no es troba prou evolucionada per a ser perdurar durant tot el procés de desenvolupament. Això vol dir simplement que, en el moment que necessitem adaptar l'aplicació a les nostres necessitats, haurem de interactuar amb un nivell lògic inferior que no hem dissenyat nosaltres i que té un nivell de complexitat bastant considerable. La sensació és la d'estar adaptant una aplicació creada amb altres propòsits i per altres programadors, a les nostres necessitats.

Personalment crec que l'eina *SpringRoo* és una bona base per a desenvolupar prototips i proves de concepte però que per a portar a terme el desenvolupament d'una aplicació completa, altament adaptada a les nostres necessitats particulars, obtindrem un millor resultat en un temps inferior si ho fem manualment o amb l'ajuda d'algunes de les plantilles existents, adoptant els patrons de disseny que siguin apropiats a les nostres necessitats.

Per contra, treballar amb el bastiment *Spring* m'ha resultat una experiència molt enriquidora i l'he trobat molt ben dissenyat i intuïtiu, queda definitivament afegit a la meva motxilla informàtica.

El disseny de la interfície d'usuari ha sigut l'aspecte al qual he dedicat més temps durant el desenvolupament, entenc perfectament que normalment hi hagi un equip de dissenyadors i programadors dedicats a aquesta tasca i he apreciat la completa separació entre funcionalitat i presentació que ens ofereix el patró MVC ja que m'ha permès desenvolupar i provar la funcionalitat, sense haver de preocupar-me per la interfície d'usuari, en primer lloc, per a, posteriorment, dedicar-me a dissenyar aquesta.

En general considero l'experiència obtinguda en aquest projecte com molt valuosa, si he de destacar un punt en concret, aquest seria que m'ha mostrat tenir, personalment, la capacitat per a portar a terme un projecte informàtic d'una certa envergadura des de la seva concepció al seu desplegament. En aquest punt trobo que el treball final de carrera difereix molt de la resta d'assignatures del pla d'estudis, totes les quals[38] es centren en un aspecte bastant concret de la informàtica.

38 No he cursat l'assignatura Tècniques de desenvolupament de programari, que podria ser semblant en aquest aspecte.

12 Glossari.

Administrador: figura encarregada de controlar el correcte funcionament del lloc web, podrà portar a terme totes les mateixes funcions que un usuari registrat i, a més, algunes extres com administrar socis.

Anunci: publicació per part d'un usuari d'una informació amb la finalitat de contactar amb altres usuaris. Pot tractar-se d'una cerca de feina, una oferta de material, per exemple venda de material fotogràfic, o de qualsevol altra tema a la discreció de l'usuari i adherint-se als criteris d'utilització del lloc web.

Compra: transacció econòmica per la qual un usuari registrat adquireix els drets d'utilització d'una imatge publicada per un altre usuari (o ell mateix tot i que no té gaire sentit).

Etiqueta: paraula clau definida per l'equip d'administració del sistema que intenta definir una categoria d'imatges per tal de facilitar la cerca d'imatges als usuaris del sistema. Una imatge pot trobar-se associada a un nombre il·limitat d'etiquetes les quals haurien de fer referència al seu contingut.

Imatge: principal focus d'atenció del lloc web, pot tractar-se de qualsevol tipus de contingut gràfic, fotografies, il·lustracions...

Ítem: Entitat amb una identitat pròpia i unes certes característiques que la diferencien de la resta que un usuari registrat ha publicat en el lloc web per tal de fer-lo accessible a la resta d'usuaris. Inicialment es preveuen dos tipus d'ítems, imatges i anuncis, tot i que no es descarta la inclusió de nous tipus en un futur.

Marcar: crear un vincle entre un usuari registrat i un ítem (inicialment una imatge o un anunci) per tal de facilitar la seva localització en un moment posterior.

Nom d'usuari: nom únic dins del sistema que ha de permetre la identificació d'un usuari registrat de manera unívoca.

Objecte: Nom que utilitzem per comoditat per a referir-nos a un Ítem.

Usuari (registrat): un usuari que s'ha registrat amb el lloc web per tal de tindre accés a totes les funcionalitats ofertes, és necessari conèixer un correu electrònic amb el qual se li podrà contactar, un nom d'usuari i la seva contrasenya. La resta de les dades seran opcionals, com per exemple les dades de pagament, tot i que li facilitarà la utilització de les funcionalitats del lloc web.

Visitant: usuari final de l'aplicació que no es troba autenticat davant del sistema i, per tant, no es pot associar amb cap informació existents. Els usuaris visitants tindran accés a una funcionalitat limitada dins el lloc web però tindran l'opció de registrar-se.

13 Bibliografia

AJI03: **Radmnivas Laddad**, *Aspectj in action, practical aspect-oriented programming*, 2003, Manning.

AOO04: **Benet Campderrich Falgueras**, *Anàlisi orientada a objectes*, 2004, Universitat Oberta de Catalunya.

DOO04: **Benet Campderrich Falgueras**, *Disseny orientat a objectes*, 2004, Universitat Oberta de Catalunya.

DP94: **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.**, *Design Patterns. Elements of reusable Object-Oriented software.*, 1994, .

HFHTML06: **Elisabeth Freeman, Eric Freeman**, *Head first HTML with CSS and XHTML*, 2006, O'Reilly.

IEPOO04: **Benet Campderrich Falgueras**, *Introducció a l'enginyeria del programari OO*, 2004, Universitat Oberta de Catalunya.

POSA96: **Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M.**, *A System of Patterns. Pattern-Oriented Software Architecture.*, 1996, Addison-Wesley.

RIDR04: **Benet Campderrich Falgueras**, *Recollida i documentació de requisits*, 2004, Universitat Oberta de Catalunya.

SFRD12: **Rod Johnson, Juergen Hoeller, Keith Donald et al.**, *Spring Framework reference documentation*, 2012, SpringSource.

SIA11: **Craig Wallis**, *Spring in Action Third Edition*, 2011, Manning.

SRCB11: **Ashish Sarin**, *Spring Roo 1.1 Cookbook*, 2011, Packt Publishing.

SRIA12: **Ken Rimple, Srini Penchikala**, *Spring Roo in action*, 2012, Manning.

STWI10: **Steven Willems**, *Spring MVC 3.0 Cache-Control headers*, http://www.i-develop.be/blog/tag/spring-mvc/.

SWFD12: **Keith Donald, Erwin Vervaet, Jeremy Grelle et alt...**, *Spring Web Flow Reference Guide*, 2012, Springsource.

UCSSR08: **Tomy Olsson, Paul O'Brien**, *The ultimate CSS reference.*, 2008, SitePoint Pty Ltd.

14 Annex A : Script roo de creació de l'aplicació.

```
// Spring Roo 1.2.1.RELEASE [rev 6eae723]

// Create project
project --topLevelPackage cat.imatgina

// Logging setup
logging setup --level DEBUG --package PROJECT

// Persistence setup HYPERSONIC
// jpa setup --provider HIBERNATE --database HYPERSONIC_IN_MEMORY

// Persistence setup MYSQL
jpa setup --provider HIBERNATE --database MYSQL --databaseName imatgina --hostName localhost --user-
Name i --password a

// Create Registered user
entity jpa --class ~.domain.Usuari --table USUARI --testAutomatically
field string --fieldName nom --notNull --unique --column name
field string --fieldName email --column email --regexp [A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]
{2,4}
field string --fieldName contrassenya --notNull --column contrassenya --sizeMin 6
field boolean --fieldName actiu --notNull --column actiu

// Create Authorizations table
entity jpa --class ~.domain.Permisos --table Permisos --testAutomatically
field reference --fieldName usuari --type ~.Domain.Usuari --cardinality MANY_TO_ONE --notNull
field string --fieldName perms --notNull

// Create Categoria
entity jpa --class ~.domain.Categoria --table CATEGORIA --testAutomatically
field string --fieldName nom --unique --column nom --notNull
field string --fieldName descripcio --notNull --column descripcio
field reference --fieldName creador --type ~.domain.Usuari --cardinality MANY_TO_ONE --notNull
field reference --fieldName superCategoria --type ~.domain.Categoria --cardinality MANY_TO_ONE

// Create Etiqueta
entity jpa --class ~.domain.Etiqueta --table ETIQUETA --testAutomatically
field string --fieldName nomEtiqueta --column nom_etiqueta --notNull --unique
field string --fieldName descripcio --column descripcio
field reference --fieldName creador --type ~.domain.Usuari --notNull --cardinality MANY_TO_ONE

// Create Imatge
entity jpa --class ~.domain.Imatge --table IMATGE --testAutomatically
field string --fieldName nom --column nom
field string --fieldName description
field date --fieldName dataPublicacio --type java.util.Date --notNull
field reference --fieldName usuari --type ~.domain.Usuari --notNull
field boolean --fieldName actiu --notNull --value true
field set --fieldName etiquetes --type ~.domain.Etiqueta --cardinality MANY_TO_MANY
field number --fieldName preu --type float --notNull --column preu

// Create Compra
entity jpa --class ~.domain.Compra --table COMPRA --testAutomatically
field date --fieldName dataCompra --type java.util.Date --column DATA_COMPRA --notNull --persisten-
ceType JPA_TIMESTAMP
field boolean --fieldName confirmada --notNull --column confirmada --value false
field reference --fieldName usuari --type ~.domain.Usuari --cardinality MANY_TO_ONE --notNull
field reference --fieldName imatge --type ~.domain.Imatge --cardinality MANY_TO_ONE --notNull

// Create Factura
entity jpa --class ~.domain.Factura --table FACTURA --testAutomatically
field date --fieldName dataFactura --column data_factura --type java.util.Date --notNull
field number --fieldName totalFactura --type float --notNull --column total_factura
field reference --fieldName usuari --type ~.domain.Usuari --cardinality MANY_TO_ONE --notNull

// Create Anunci
entity jpa --class ~.domain.Anunci --table ANUNCI --testAutomatically
```

```
field string --fieldName nom --column nom
field string --fieldName description
field date --fieldName dataPublicacio --type java.util.Date --notNull
field reference --fieldName usuari --type ~.domain.Usuari --notNull
field boolean --fieldName actiu --notNull --value true
field set --fieldName categories --type ~.domain.Categoria --cardinality MANY_TO_MANY

// Create Resposta
entity jpa --class ~.domain.Resposta --table RESPOSTA --testAutomatically
field string --fieldName contingut --notNull --column contingut
field reference --fieldName usuari --type ~.domain.Usuari --notNull --cardinality MANY_TO_ONE
field reference --fieldName anunci --type ~.domain.Anunci --cardinality MANY_TO_ONE --notNull

// Create ShoppingTrolley
class --class ~.domain.ShoppingTrolley
field reference --fieldName usuari --type ~.domain.Usuari --notNull --cardinality MANY_TO_ONE
field set --fieldName compres --type ~.domain.Compra --cardinality ONE_TO_MANY

// Update Usuari_Registrat
focus --class ~.domain.Usuari
field set --fieldName imatgesMarcades --type ~.domain.Imatge --cardinality MANY_TO_MANY
field set --fieldName anuncisMarcats --type ~.domain.Anunci --cardinality MANY_TO_MANY

// Update Compra
focus --class ~.domain.Compra
field reference --fieldName factura --type ~.domain.Factura --cardinality MANY_TO_ONE

// Web mvc setup
web mvc setup
web mvc all --package ~.web

// Add basic security
security setup
```

15 Annex B: problemes a resoldre, *bugs, wish-list.*

1. Els **zeros a l'esquerra en el camp de contrasenya** d'usuari es perden amb el qual les contrasenyes, el *hash* de les quals comença per aquest dígit, no coincideixen i els usuaris no poden autenticar-se tot i utilitzar els detalls correctes.

Aquest problema persisteix tot i haver inclòs el següent codi per a intentar prevenir-lo

```
if (hashedPass.length() < 32)
      hashedPass = "0" + hashedPass;
```

Es pot veure un exemple provant amb la contrasenya `usuari`.

2. Seria interessant implementar totes peticions que retornen a la mateixa pàgina, com el marcatge d'ítems o la compra d'imatges, com a crides AJAX per tal de no haver de recarregar tota la pàgina i, d'aquesta manera, reduir el tràfic generat.

3. S'hauria d'implementar el mètode de pagament, per a fer-lo s'hauria d'estudiar el mercat i decidir, en primer lloc, quines són les ofertes més interessants.