TFCshop: Botiga virtual JavaEE

Jaume Junyent Rosich
ETIS

Consultor: Jose Juan Rodriguez

14/01/2009

Resum i paraules clau

Aquest treball de fi de carrera pertany a l'àrea J2EE. L'objectiu principal de l'àrea J2EE és aprofundir en l'ús de la tecnologia Java, i introduir a l'estudiant en l'arquitectura J2EE mitjançant l'anàlisi, disseny i implementació d'una aplicació basada en aquesta arquitectura.

L'aplicació en qüestió consisteix en un front-end web per a l'exposició al públic del catàleg de productes d'una empresa qualsevol, que permeti als clients localitzar i consultar els productes que li interessen i efectuar comandes si ho desitja. D'altra banda, també es pretén facilitar la gestió d'aquest catàleg als usuaris interns de l'empresa mitjançant la mateixa aplicació. La gestió del cicle de vida de les comandes queda fora de l'abast d'aquest projecte.

La implementació del projecte implica l'ús d'un conjunt de tecnologies de l'arquitectura J2EE (o Java Enterprise Edition, segons la denominació més recent) i de Java en general. Aquestes tecnologies són: JSP + Struts2 + Tiles per a la capa web, EJB3.0 + JPA per a la capa de negoci i l'accés a la base de dades relacional, i Jboss Application Server com a contenidor web i EJB.

Paraules clau: botiga virtual, Struts2, Tiles, EJB, JPA, B2C, JBoss

Àrea de TFC: J2EE

Índex de continguts

Índex de figures

1. Introducció.

1.1. Justificació i context del TFC

Des de fa uns anys, la plataforma Java s'ha convertit en una de les més utilitzades en diversos àmbits, sobretot en entorns distribuïts. La combinació d'un llenguatge independent de la plataforma subjacent, completíssimes APIs, l'existència de diverses edicions de la plataforma per a cobrir totes les necessitats (Java SE - Standard Edition, JavaEE - Enterprise Edition, JavaME - Micro Edition) i la disponibilitat tant d'implementacions propietàries com d'implementacions lliures de les especificacions la fan molt atractiva per a arquitectes i desenvolupadors. A això cal sumar-hi el gran nombre de projectes de codi obert desenvolupats en Java per una comunitat molt activa, que faciliten el desenvolupament eficient i, per tant, econòmic de noves aplicacions, contribuint així a consolidar encara més el seu domini. Entre altres, podríem citar com a exemples molt coneguts: Hibernate, Struts (Struts2), Spring, RichFaces. Tenint en compte que, a més, en els darrers anys JavaSE i JavaEE s'han anat renovant , amb l'aparició d'una nova versió de l'especificació EJB (EJB3.0) molt més prometedora i amb la incorporació de la Java Persistence API (JPA) a JavaSE com a estàndard de persistència d'objectes en base de dades relacionals, és sens dubte el moment de posar-se al dia en el coneixement d'aquest món.

Aquesta és la meva principal motivació per a realitzar el treball de fi de carrera en aquesta àrea: aprendre'n tant com pugui d'una tecnologia actual amb àmplia demanda en el mercat laboral. Amb aquesta motivació en ment, el projecte hauria de permetre'm integrar diversos elements de l'especificació de JavaEE, com poden ser Servlets, Java Sever Pages (JSP) i Enterprise JavaBeans (EJB). També seria molt interessant integrar-hi algun *framework* web com Struts(2) o Java Server Faces (JSF) i algun mecanisme de persistència d'objectes (Hibernate o JPA). Sembla que una aplicació web de comerç electrònic *business to consumer* , és a dir, una botiga virtual, dissenyada en tres capes seria ideal. Així, a banda de la tecnologia, espero aprendre sobre la problemàtica que s'oculta darrera d'aplicacions que, com a simples usuaris, utilitzem tant habitualment avui en dia per comprar llibres o altres productes amb total comoditat.

1.2. Objectius del TFC

La finalitat d'aquest treball de fi de carrera serà la realització d'una botiga virtual genèrica implementada mitjançant la plataforma Java Enterprise Edition (JavaEE). El capítol de requeriments explica amb més detall la finalitat i funcions de l'aplicació, però aquí podem resumir els seus objectius principals:

- ✓ Permetre la consulta del catàleg de productes d'una empresa qualsevol a través d'Internet
- ✓ Permetre la introducció de comandes per part dels clients

- ✓ Permetre el manteniment dels usuaris de l'aplicació i dels articles del catàleg per part del personal de l'empresa

D'altra banda, l'objectiu general és satisfer els requeriments mitjançant l'arquitectura JavaEE, amb la finalitat d'aprendre el màxim possible sobre aquesta. Això implicarà:

- ✓ Fer un disseny de l'aplicació en tres capes: capa de presentació, capa de lògica de negoci, i capa dades, implementant cadascuna aquells aspectes de la funcionalitat de l'aplicació que li són propis, i utilitzant els patrons de disseny més adequats.

- ✓ Estudiar els *frameworks* i estàndards disponibles per a les diferents capes de l'aplicació, tant els definits per les especificacions de JavaEE (JSP, Servlet, JPA, JSF) com els disponibles gràcies a la comunitat de software lliure (Hibernate, Struts, etc.) per tal d'escollir els més adequats.

- ✓ Proporcionar una interfície d'usuari web entenedora, pràctica i, ja que es tracta d'una botiga virtual, que sigui visualment mínimament atractiva.

- ✓ Estructurar la lògica de negoci i la persistència de dades per garantir el correcte funcionament en situacions de concurrència d'usuaris.

- ✓ Establir un conjunt de rols i definir els mecanismes d'autenticació i autorització que donaran seguretat a l'aplicació.

- ✓ Seleccionar un servidor d'aplicacions Java concret entre els disponibles en el mercat i configurar adequadament els seus serveis per a donar el suport necessari a l'aplicació.

Aquest objectius es concreten més detalladament mitjançant la descomposició en tasques donada en l'apartat corresponent a la planificació del projecte.

1.3. Enfocament i mètode seguit

El mètode de treball no s'allunyarà gaire del procés clàssic en cascada, amb la següent successió d'etapes:

- ✓ Establir els requeriments de l'aplicació
- ✓ Anàlisi i especificació dels requeriments, i disseny de les pantalles de l'aplicació
- ✓ Disseny de l'arquitectura, model de dades, lògica de negoci i presentació.
- ✓ Implementació
- ✓ Proves (funcionals executades manualment, segons el temps disponible es podrien fer proves unitàries automàtiques)

Amb el temps disponible per a l'elaboració del projecte, aquest no pot tenir una gran complexitat ni un gran abast funcional. A més, els requeriments són autoimposats. Per tant, malgrat que es puguin descobrir nous requeriments o necessitar algun canvi en el disseny quan ja ens trobem més avançats en el projecte, la realimentació entre les etapes serà poc significativa i es preveu seguir bastant fidelment aquest cicle de vida. Per al cicle de vida d'aquest projecte, no es consideren les etapes posteriors com poden ser l'operació i el manteniment del sistema.

1.4. Planificació del projecte

D'entrada es preveuen les següents tasques (les tecnologies indicades són només exemples del que es preveu utilitzar):

Fita: PAC2 – 05/11/2008

- ✓ Estudi de l'arquitectura JavaEE i *frameworks* disponibles
- ✓ Anàlisi detallat dels casos d'ús
- ✓ Anàlisi de la interfície gràfica – seqüència de pantalles
- ✓ Definició general de l'arquitectura de l'aplicació
- ✓ Disseny del model de dades
- ✓ Disseny de la lògica de negoci
- ✓ Realitzar l'informe de la PAC2
- ✓ Instal·lació i configuració del software (JBoss, MySQL, JDK, etc.)

Fita: PAC3 – 17/12/2008

- ✓ Implementació del model de dades (BBDD relacional i classes Java – Hibernate/JPA)
- ✓ Càrrega de dades de prova
- ✓ Implementació de la lògica de negoci (EJB)
- ✓ Implementació de la capa de presentació (JSP, Struts, JSF, Facelets). A falta de l'anàlisi i el disseny que es duran a terme per la PAC 2, sembla raonable implementar-la en paral·lel amb la lògica de negoci, d'acord amb la següent seqüència:
- ✓ Gestió d'usuaris: alta, baixa i edició d'usuaris per part dels administradors, auto-registre dels usuaris (els visitants es poden donar d'alta com a clients), gestió de dades personals (els clients poden modificar les seves dades personals)
- ✓ Gestió del catàleg: permetre als treballadors afegir/editar/esborrar articles del catàleg, i permetre als visitants consultar el catàleg
- ✓ Gestió de comandes: creació de comandes per part dels clients, i permetre als clients veure el seu historial de comandes i anul·lar les més recents pendents d'enviar.
- ✓ Informe PAC3

Fita: Lliurament final - 14/01/2009

- ✓ Proves i retocs diversos
- ✓ Elaboració de la memòria
- ✓ Elaboració de la presentació

A continuació, la figura *Figura 1. Planificació temporal del projecte* mostra la planificació temporal de les tasques:

	Nombre de tarea	Duración	Comienzo	Fin
1	**PAC2 - 05/11/2008**	**29 días**	**lun 29/09/08**	**mié 05/11/08**
2	Estudi JavaEE i frameworks	29 días	lun 29/09/08	mié 05/11/08
3	Anàlisi dels casos d'ús	10 días	lun 29/09/08	dom 12/10/08
4	Anàlisi de la interfície gràfica	10 días	lun 29/09/08	dom 12/10/08
5	Definició general de l'arquitectura	2 días	lun 13/10/08	mar 14/10/08
6	Disseny del model de dades	5 días	mié 15/10/08	lun 20/10/08
7	Disseny de la lògica de negoci	8 días	mar 21/10/08	jue 30/10/08
8	Informe PAC2	21 días	jue 09/10/08	mié 05/11/08
9	Instal•lació i configuració del software (JBoss, MySQL, JDK, etc.)	4 días	vie 31/10/08	mié 05/11/08
10	**PAC3 - 17/12/2008**	**31 días**	**jue 06/11/08**	**mié 17/12/08**
11	Implementació del model de dades (BBDD relacional i classes Java – Hibernate/JPA)	4 días	jue 06/11/08	mar 11/11/08
12	Càrrega de dades de prova	2 días	mar 11/11/08	mié 12/11/08
13	**Implementació de la lògica de negoci (EJB) i implementació de la capa de presentació (JSP, Struts, JSF, Facelets)**	**23 días**	**jue 13/11/08**	**dom 14/12/08**
14	Gestió d'usuaris	6 días	jue 13/11/08	jue 20/11/08
15	Gestió del catàleg	10 días	vie 21/11/08	jue 04/12/08
16	Gestió de comandes	7 días	vie 05/12/08	dom 14/12/08
17	Informe PAC3	3 días	lun 15/12/08	mié 17/12/08
18	**Lliurament final - 14/01/2009**	**20 días**	**jue 18/12/08**	**mié 14/01/09**
19	Proves i retocs d'estil	12 días	jue 18/12/08	dom 04/01/09
20	Elaboració de la memòria	12 días	jue 18/12/08	dom 04/01/09
21	Elaboració de la presentació	8 días	lun 05/01/09	mié 14/01/09

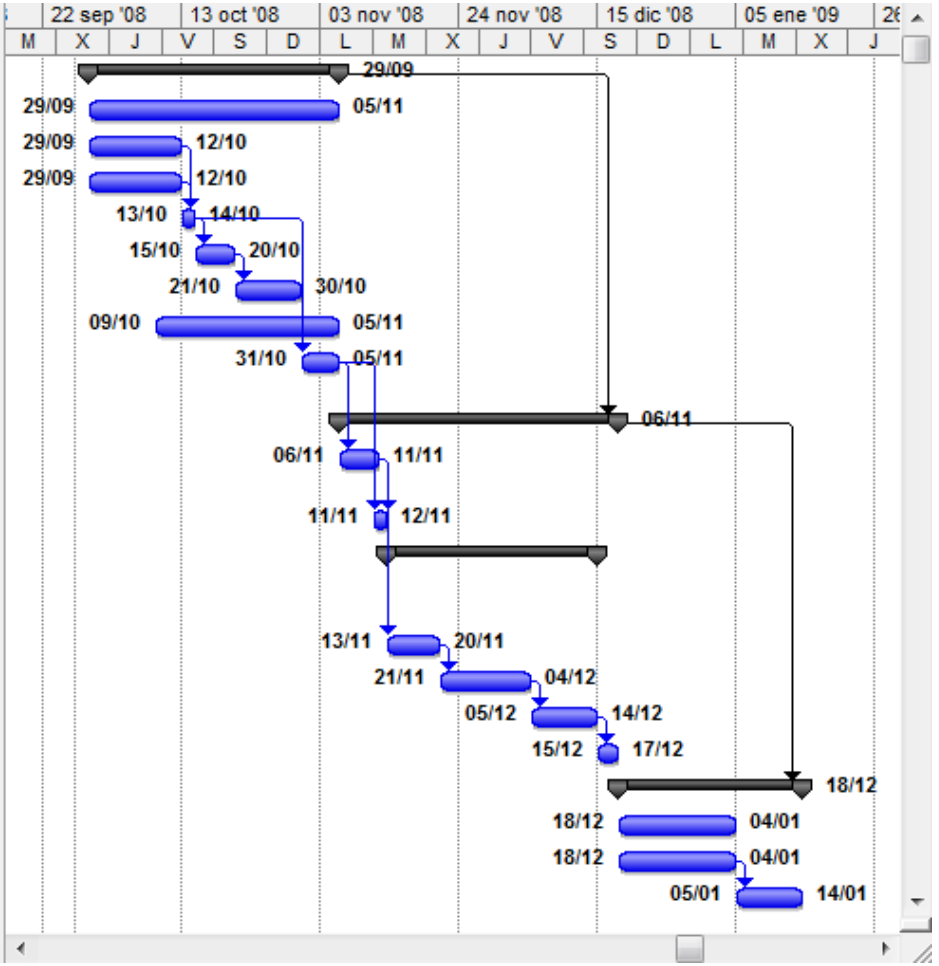


Figura 1. Planificació temporal del projecte

1.5. Productes obtinguts

Els productes obtinguts al final del projecte són:

- ✓ Aquesta memòria, que conté la descripció de la feina feta.
- ✓ Una presentació que sintetitza els diversos aspectes del projecte.
- ✓ L'aplicació en sí, consistent en el codi font i una versió compilada d'aquest preparada per a ser desplegada en un servidor d'aplicacions. Va acompanyada dels *scripts* de creació de la base de dades i algunes dades de prova. Les instruccions per a la instal·lació es troben a la pròpia memòria, però per comoditat també s'adjunten amb el producte.

1.6. Breu descripció dels altres capítols de la memòria

Els capítols següents expliquen les diferents fases del projecte seguint l'ordre cronològic, començant per l'anàlisi dels requeriments i les pantalles necessàries, continuant amb l'explicació de l'arquitectura i el disseny del model de dades, la lògica de negoci i els diagrames de seqüència corresponents, acabant amb la implementació i les instruccions per a la instal·lació (Annex I). L'Annex II conté les sentències de creació de les taules de la base de dades, i l'Annex III conté una mostra de com han quedat finalment algunes de les pantalles, que en general s'assemblen al disseny original, però millorades visualment.

2. Requeriments

La botiga disposarà d'un catàleg de productes que els usuaris podran consultar, ja sigui navegant per les categories de productes, com fent cerques mitjançant un cercador. Hi haurà un cistell de la compra, al qual l'usuari podrà anar afegint els productes que desitja comprar, i finalment els usuaris podran fer una comanda, sempre i quan s'hagin registrat prèviament donant les seves dades personals.

En el moment de fer la compra, el sistema autenticarà l'usuari, o bé permetrà de crear un compte d'usuari si el comprador no en té cap encara. Es demanarà a l'usuari que confirmi les dades de la comanda, i que indiqui l'adreça a la qual s'haurà d'enviar el producte. L'adreça d'enviament podrà ser una de les que constin entre les dades personals del comprador, o una altra que especifiqui en el moment de la compra (p.ex.: si es compra un regal per algú altre). També se li demanaran les dades de la seva targeta de crèdit. En principi, el processament de les comandes, incloent la gestió del pagament, es deixarà en mans d'un sistema extern.

Els usuaris registrats també podran consultar el seu historial de comandes, anul·lar les comandes que encara no hagin estat enviades, i modificar les seves dades personals.

Els empleats de la botiga també tindran una part privada exclusiva per a ells, en la qual podran mantenir el catàleg (afegir/modificar/esborrar productes). Els administradors podran gestionar la llista d'usuaris, que inclou els usuaris registrats, els treballadors i els administradors.

En funció del temps disponible, es podrien incorporar altres funcionalitats addicionals, per exemple:

- ✓ Que el sistema tingui en compte les possibles promocions vigents per a fer suggeriments al comprador basant-se en el contingut del carret de la compra.

- ✓ Que cada usuari pugui mantenir una llista de productes desitjats: "wish-list"

- ✓ Permetre als usuaris registrats és la de contactar amb el propietari de la botiga per a notificar alguna incidència (p.ex.: no haver rebut una mercaderia o haver-la rebut en mal estat). Caldria implementar també el sistema corresponent de gestió d'incidències per part dels treballadors.

- ✓ L'aplicació podria mantenir una llista dels productes visitats més recentment pel visitant, la qual serà mostrada en alguna part de les pàgines de cerca per a permetre comparar ràpidament els productes i així facilitar el procés de decisió de compra.

- ✓ Permetre als usuaris registrats donar la seva valoració sobre els productes comprats, amb la finalitat que serveixi de guia per als altres clients.

- ✓ La navegació per les parts privades de l'aplicació s'hauria de fer de manera segura mitjançant criptografia.

3. Anàlisi

3.1. Casos d'ús

Prèviament als casos d'ús, comencem per identificar els actors que intervenen en els diversos processos:

- ✓ **Visitant**: qualsevol persona que accedeixi a la botiga virtual mitjançant un navegador web.
- ✓ **Usuari registrat**: Visitant que disposa d'un nom d'usuari proporcionat pel sistema. Mentre no s'identifiqui com a tal, actua com a simple Visitant.
- ✓ **Client**: Usuari registrat que pot fer compres a la botiga.
- ✓ **Treballador**: Usuari registrat que treballa per al propietari de la botiga.
- ✓ **Administrador**: Treballador amb els màxims privilegis sobre l'aplicació.

L'anàlisi dels casos d'ús derivats dels requeriments funcionals es pot resumir en el següent diagrama:

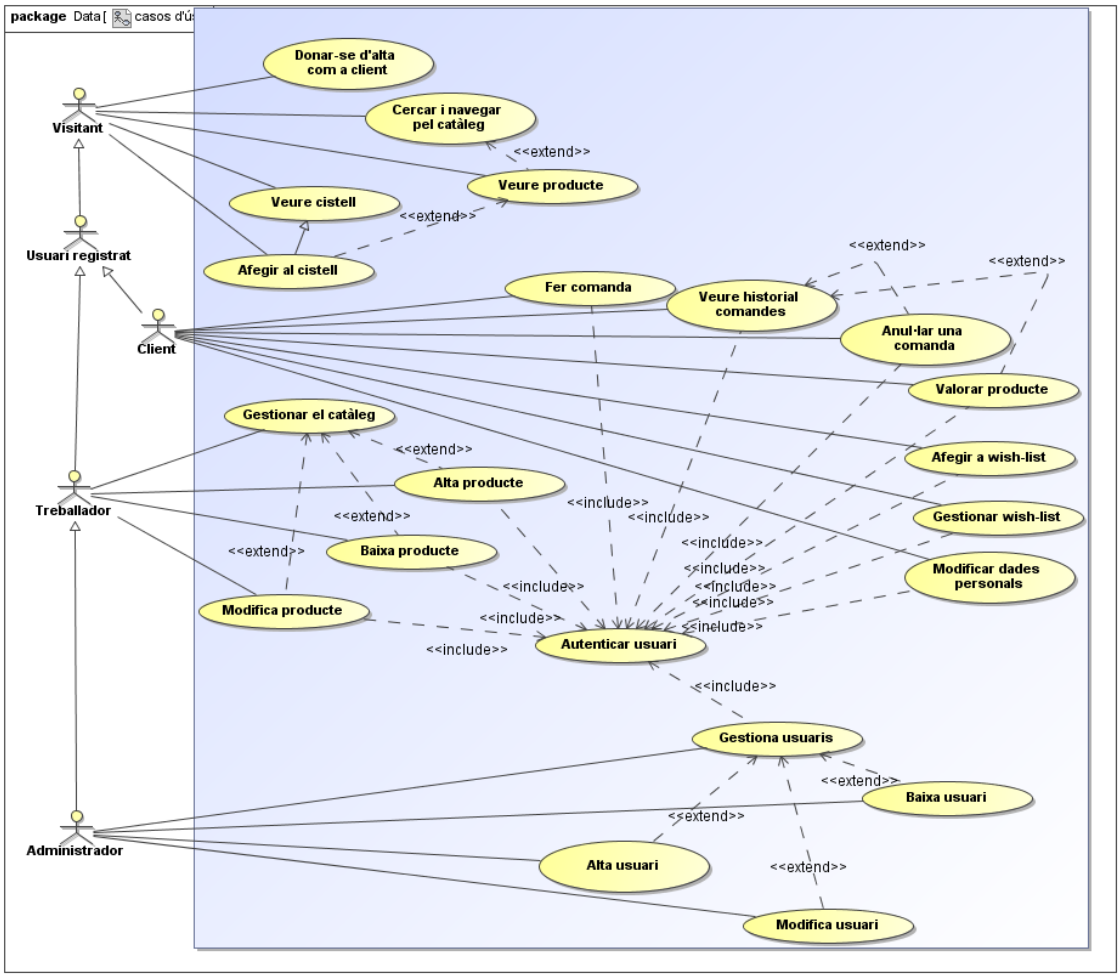


Figura 2. Diagrama de casos d'ús

A continuació, es descriuen els principals casos d'ús:

3.1.1. Donar-se d'alta com a client

Permet al visitant obtenir un nom d'usuari únic que l'identifiqui com a Client.

Actors: Visitant
Casos d'ús relacionats: "Autenticar usuari"
Precondicions: -
Postcondicions: -
Procés normal principal:

1. El sistema demana al visitant una sèrie de dades personals, que inclouen, com a mínim, nom i cognoms , una adreça de correu electrònic, i la paraula clau per autenticar-se en el sistema.
2. L'usuari entra les dades
3. El sistema comprova les dades per garantir que l'adreça de correu electrònic sigui única en el sistema, ja que servirà com a nom d'usuari.
4. Si no hi ha conflicte, registra l'usuari a la base de dades i el redirigeix a la seva àrea privada o al destí original si aquest cas d'ús s'executa dintre d'un altre.

Alternatives de procés i excepcions:

3b. Si hi ha conflicte o algun error en les dades, es torna al pas 1 mostrant els missatges d'error necessaris.

3.1.2. Autenticar usuari

Permet/obliga al visitant a subministrar el seu nom d'usuari i paraula clau.

Actors: Visitant, Usuari registrat (Client,Treballador,Administrador)
Casos d'ús relacionats: "Donar-se d'alta com a client", "Donar d'altra un treballador/administrador", "Fer una comanda"
Precondicions: -
Postcondicions: -
Procés normal principal:

1. Si el visitant no està autenticat, el sistema demana al visitant el seu nom d'usuari i la paraula clau. També li dóna la possibilitat d'iniciar el cas d'ús "Donar-se d'alta com a usuari registrat". Si ja està autenticat, s'ignoren els passos següents i es redirigeix l'usuari igual que en el pas 4.
2. El visitant/usuari entra les dades
3. El sistema comprova les dades.
4. Si la paraula clau correspon a l'usuari indicat, el visitant es converteix en usuari registrat i és redirigit a la seva àrea privada o al destí original si aquest cas d'ús s'executa dintre d'un altre.

Alternatives de procés i excepcions:

2b. Si s'inicia l'altre cas d'ús, es passa al pas 3 després de la seva execució.
4b. Si falla l'autenticació, es torna al pas 1 mostrant els missatges d'error necessaris. Si l'autenticació falla per tercera vegada dins d'una finestra temporal de 2 dies, es deshabilita l'usuari per precaució.

3.1.3. Donar d'alta un usuari registrat

Permet a un administrador crear un nom d'usuari únic que identifiqui una persona com a Usuari Registrat (Client, Treballador o Administrador).

Actors: Administrador
Casos d'ús relacionats: "Autenticar usuari"
Precondicions: -
Postcondicions: -
Procés normal principal:

1. El sistema demana a l'administrador una sèrie de dades personals, que inclouen, com a mínim, nom i cognoms , una adreça de correu electrònic, i la paraula clau per autenticar-se en el sistema. També pregunta quin tipus d'usuari es vol crear.
2. L'administrador entra les dades
3. El sistema comprova les dades per garantir que l'adreça de correu electrònic sigui única en el sistema, ja que servirà com a nom d'usuari.
4. Si no hi ha conflicte, registra l'usuari a la base de dades i acaba el procés.

Alternatives de procés i excepcions:

4b. Si hi ha conflicte o algun error en les dades, es torna al pas 1 mostrant els missatges d'error necessaris.

3.1.4. Navegar pel catàleg

Permet al visitant veure els productes organitzats per categories.

Actors: Visitant
Casos d'ús relacionats: "Afegir al cistell de la compra", "Fer una comanda", "Cercar en el catàleg"
Precondicions: -
Postcondicions: -
Procés normal principal:

1. El sistema presenta al visitant un menú jeràrquic de categories.
2. L'usuari selecciona una categoria (o una branca de categories)
3. El sistema mostra una llista de productes pertanyents a la categoria escollida.

3.1.5. Cercar en el catàleg

Permet al visitant cercar els productes que compleixin una sèrie de condicions.

Actors: Visitant
Casos d'ús relacionats: "Afegir al cistell de la compra", "Fer una comanda", "Navegar pel catàleg"
Precondicions: -
Postcondicions: -
Procés normal principal:

1. El sistema permet al visitant indicar unes condicions de cerca de productes. Aquestes condicions seran: la categoria en la qual cercar i un text lliure.
2. L'usuari entra les dades
3. El sistema mostra una llista de productes que satisfan els criteris de cerca.

3.1.6. Veure la informació detallada sobre un producte

Permet al visitant veure la informació ampliada sobre un producte.

Actors: Visitant
Casos d'ús relacionats: "Navegar pel catàleg", "Cercar en el catàleg", "Afegir un producte al cistell de la compra"
Precondicions: el visitant ha fet una cerca o ha navegat pel catàleg i ha obtingut una llista de productes, o bé disposa d'algun enllaç directe a un producte.
Postcondicions: -
Procés normal principal:

1. El visitant fa clic sobre l'enllaç corresponent
2. El sistema mostra els detalls del producte. També dóna l'opció d'executar el cas d'ús "Afegir un producte al cistell de la compra"

3.1.7. Afegir un producte al cistell de la compra

Permet al visitant confeccionar una llista de productes amb la finalitat de fer una comanda

Actors: Visitant
Casos d'ús relacionats: "Navegar pel catàleg", "Cercar en el catàleg", "Veure la informació detallada sobre un producte", "Fer una comanda"
Precondicions: el visitant està navegant o cercant pel catàleg i/o ha executat el cas d'ús "Veure la informació detallada sobre un producte".
Postcondicions: -
Procés normal principal:

1. El visitant fa clic sobre l'enllaç "Afegir al cistell"
2. El sistema actualitza les dades del cistell i les representa a la pantalla
3. El visitant segueix visualitzant els detalls del producte.

Alternatives de procés i excepcions:

3b. El visitant pot canviar la quantitat de cadascun dels productes del cistell.
3c. El visitant pot iniciar el procés de compra

3.1.8. Veure el cistell de la compra

Permet al visitant veure el contingut del cistell i iniciar el procés de compra

Actors: Visitant
Casos d'ús relacionats: "Navegar pel catàleg", "Cercar en el catàleg", "Veure la informació detallada sobre un producte", "Fer una comanda"
Precondicions: el visitant està navegant o cercant pel catàleg i/o ha executat el cas d'ús "Veure la informació detallada sobre un producte".
Postcondicions: -
Procés normal principal:

1. El visitant fa clic sobre l'enllaç "Veure el cistell"
2. El sistema presenta per pantalla el contingut del cistell
3. El visitant torna a navegar pel catàleg

Alternatives de procés i excepcions:

3b. El visitant pot canviar la quantitat de cadascun dels productes del cistell.
3c. El visitant pot iniciar el procés de compra

3.1.9. Fer una comanda

Permet al visitant fer una comanda

Actors: Client
Casos d'ús relacionats: "Veure el cistell de la compra", "Afegir un producte al cistell de la compra", "Autenticar usuari"
Precondicions: el visitant ha afegit els productes que desitja comprar en el cistell de la compra
Postcondicions: la comanda queda registrada a la base de dades, pendent de ser confirmada un cop comprovat el pagament.
Procés normal principal:

1. El visitant fa clic sobre l'enllaç "Comprar"
2. El sistema llegeix les dades del cistell i determina si el visitant és un Client.
3. Si el visitant és un Client, se li demana que esculli l'adreça a la qual enviar el paquet d'entre les que consten entre les seves dades personals, o que n'entri una de nova.
4. L'usuari escull l'adreça.
5. El sistema li demana que entri les dades de pagament per targeta bancària.
6. L'usuari entra les dades i continua amb el procés de compra.
7. Es demana una última confirmació de la comanda, que inclou un desglòs del total a pagar, l'adreça i les dades de pagament.
8. L'usuari confirma.
9. Es registra la comanda i se li comunica si hi ha hagut cap problema. Els passos següents en relació a la comanda són externs a aquest sistema. Un altre sistema validarà el pagament més endavant i ho notificarà al sistema. La preparació de la comanda i l'enviament també seran notificats des d'un sistema extern.

Alternatives de procés i excepcions:

3b. Si el visitant no s'ha autenticat, s'inicia el cas d'ús "Autenticar usuari". Si té èxit, es segueix el pas 3.

3.1.10. Veure historial de comandes

Permet al client veure les comandes que ha fet en el passat, incloent les més recents i que encara no li han enviat.

Actors: Client
Casos d'ús relacionats: "Anul·lar una comanda", "Valorar un producte adquirit"
Precondicions: el visitant s'ha autenticat com a Client.
Postcondicions: -
Procés normal principal:

1. El visitant fa clic sobre l'enllaç "Gestionar comandes" de la seva àrea privada.
2. El sistema mostra una llista amb totes les comandes fetes per l'usuari en el passat. De cada comanda s'indica la data, la data d'enviament, els articles i la quantitat de cadascun d'ells. Les comandes en curs de validació ofereixen la possibilitat de ser cancel·lades, iniciant el cas d'ús "Anul·lar una comanda".

3.1.11. Anul·lar una comanda

Permet a l'usuari registrat modificar o cancel·lar una comanda recent que encara no li han enviat.

Actors: Client
Casos d'ús relacionats: "Veure historial de comandes"
Precondicions: l'usuari ha executat el cas d'ús "Veure historial de comandes".
Postcondicions: -
Procés normal principal:

1. El visitant fa clic sobre l'enllaç "Anul·lar comanda" associat a la comanda que vol cancel·lar.
2. El sistema demana confirmació.
3. L'usuari confirma que vol cancel·lar la comanda.
4. El sistema actualitza l'estat de la comanda.

3.1.12. Valorar un producte adquirit

Permet a un usuari donar una puntuació i fer un comentari sobre un producte que hagi adquirit recentment.

Actors: Client
Casos d'ús relacionats: "Veure historial de comandes", "Actualitzar la valoració dels productes"
Precondicions: l'usuari ha comprat el producte en el passat, i ha executat el cas d'ús "Veure historial de comandes"
Postcondicions: -
Procés normal principal:

1. L'usuari fa clic sobre un enllaç relacionat amb un article d'una comanda.
2. Es presenta a l'usuari una pantalla que li permet donar una puntuació al producte i escriure un text. Si l'usuari ha valorat el producte anteriorment, apareix la informació que va entrar en el seu moment.
3. L'usuari entra les dades i fa clic a "Enviar" i s'enregistren/actualitzen les dades de la valoració.

3.1.13. Actualitzar la valoració dels productes

Aquest cas d'ús actualitza la puntuació mitjana dels productes de la base de dades d'acord amb totes les valoracions introduïdes pels usuaris.

Actors: cap (és un procés que s'executa periòdicament)
Casos d'ús relacionats: "Valorar un producte adquirit"

3.1.14. Afegir un producte a la wish-list

Permet a un client confeccionar una llista de productes que li agraden, per a poder consultar-la en el futur.

Actors: Client

Casos d'ús relacionats: "Veure la informació detallada sobre un producte"
Precondicions: el visitant ha executat el cas d'ús "Veure la informació detallada sobre un producte" com a usuari registrat.
Postcondicions: -
Procés normal principal:

1. El visitant fa clic sobre l'enllaç "Afegir a la wish-list"
2. El sistema afegeix el producte a la llista (si no hi és ja) i s'inicia el cas d'ús "Gestionar wish-list".

Alternatives de procés i excepcions:

2b. Si el visitant no s'ha autenticat, s'inicia el cas d'ús "Autenticar usuari". Després es segueix amb el pas 2.

3.1.15. Gestionar el catàleg

En realitat, són diversos casos d'ús en un: permet a un treballador afegir/eliminar/modificar productes del catàleg.

Actors: Treballador
Casos d'ús relacionats: -
Precondicions: el treballador ha estat autenticat i es troba en la seva àrea privada.
Postcondicions: -
Procés normal principal:

1. El treballador fa clic sobre l'enllaç "Gestionar el catàleg"
2. El sistema mostra una pàgina que permet al treballador cercar productes del catàleg i mostra una llista de resultats, permetent editar/esborrar cada producte de la llista.
3. El treballador escull editar un producte de la llista
4. El sistema mostra un formulari amb els camps precarregats amb les dades del producte, que poden ser modificades. Entre les dades s'ha d'especificar la categoria a les qual s'associarà el producte.
5. El treballador entra les dades i fa clic a "Guardar"
6. El sistema enregistra les dades.

Alternatives de procés i excepcions:

3b. El treballador escull esborrar un producte de la llista
4b. El sistema demana confirmació i en cas afirmatiu marca el producte com a "esborrat del catàleg/no disponible", però es manté tota la informació intacta. El producte no és esborrat completament de la base de dades per tal de mantenir la integritat de l'historial de comandes i de les relacions amb altres entitats en general.

1c. El treballador fa clic sobre l'enllaç "Afegir un article".
2c. Es salta al pas 4, però el formulari de la pantalla no conté dades.

3.1.16. Gestionar wish-list

Permet a un usuari registrat visualitzar i modificar la seva wish-list.

Actors: Usuari registrat
Casos d'ús relacionats: "Afegir un producte a la wish-list", "Veure la informació detallada sobre un producte"

Precondicions:
Postcondicions: -
Procés normal principal:

1. El visitant fa clic sobre l'enllaç "Gestionar wish-list"
2. Es mostra la llista. Els elements de la llista permeten iniciar el cas d'ús "Veure la informació detallada sobre un producte" per al producte que representen. També donen l'opció d'esborrar-los de la llista.
3. L'usuari registrat escull esborrar un element.
4. L'element és esborrat i es mostra la llista actualitzada.

Alternatives de procés i excepcions:

2b. Si el visitant no s'ha autenticat, s'inicia el cas d'ús "Autenticar-se com a usuari registrat". Després es segueix amb el pas 2.

3.1.17. Gestionar usuaris

En realitat, són diversos casos d'ús en un: permet a un administrador cercar un usuari, modificar-ne les dades personals i habilitar-lo/deshabilitar-lo.

Actors: Administrador
Casos d'ús relacionats: -
Precondicions: l'administrador ha estat autenticat i es troba en la seva àrea privada.
Postcondicions: -
Procés normal principal:

1. L'administrador fa clic sobre l'enllaç "Gestionar els usuaris existents"
2. El sistema mostra una pàgina que permet l' administrador cercar usuaris i que permet editar/esborrar cada usuari de la llista resultant de la cerca.
3. L'administrador escull editar un producte de la llista
4. El sistema mostra una pantalla en què es demanen les dades corresponents a l'usuari. Els camps apareixen precarregats amb les dades de l'usuari seleccionat. Entre les dades a proporcionar, hi ha el tipus d'usuari.
5. L'administrador entra les dades i fa clic a "Guardar"
6. El sistema enregistra els canvis.

Alternatives de procés i excepcions:

3b. L'administrador escull donar de baixa un usuari de la llista
4b. El sistema demana confirmació i en cas afirmatiu marca l'usuari com a "deshabilitat", però es manté tota la informació intacta.

1c. L'administrador fa clic sobre l'enllaç "Afegir un nou usuari"
2c. Es salta al pas 4, però es mostra la pantalla sense dades.

3.1.18. Modificar dades personals

Aquest cas d'us permet a un usuari registrat modificar les seves dades personals, a través d'una opció de menú de la seva àrea privada.

3.2. Interfície gràfica

3.2.1. Estructuració de les pàgines de l'aplicació

El següent diagrama representa la navegació entre les pàgines principals de l'aplicació:

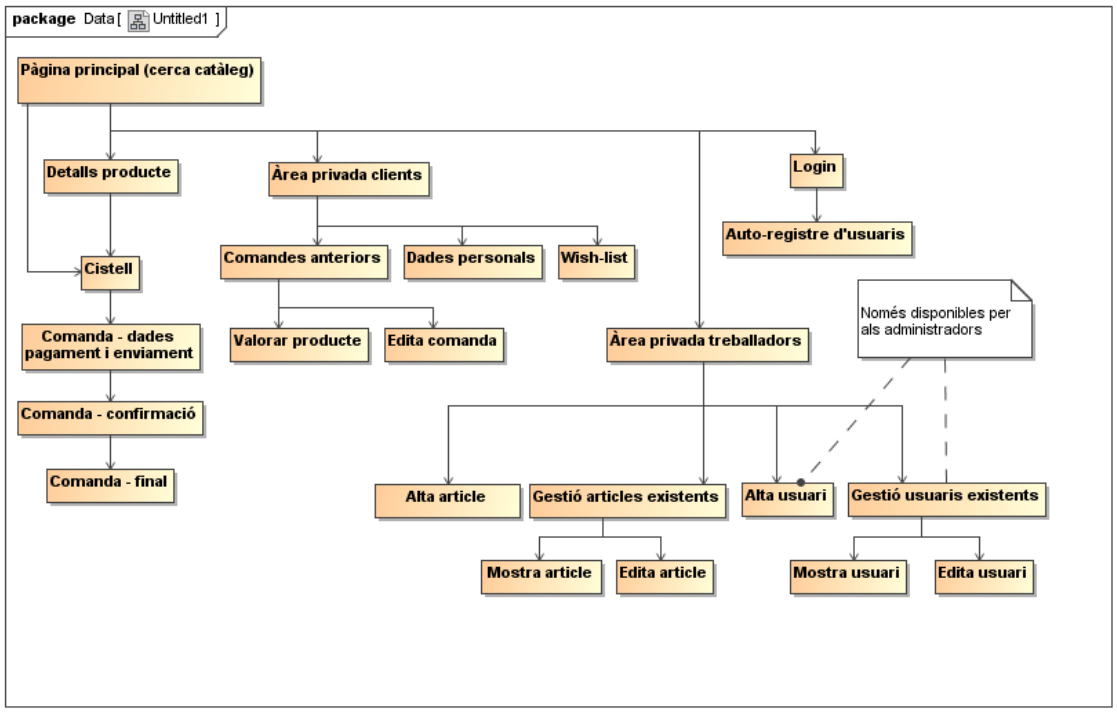


Figura 3. Estructuració de les pàgines de l'aplicació

El visitant arribarà sempre a la pàgina principal, que és la de navegació/cerca del catàleg.

A partir de la pàgina principal, l'usuari podrà veure els productes segons les categories i les condicions de cerca, afegir-los al cistell i veure el contingut del cistell.

Des de la pàgina de visualització del cistell, es pot iniciar el procés de compra, format per tres pantalles que recullen les dades necessàries i demanen la confirmació del client.

Des de qualsevol pàgina, el visitant podrà entrar a la seva à*rea privada*. Si es tracta d'un client, aquesta pàgina li permetrà veure les comandes anteriors, que li pot servir per a anul·lar-les i donar una valoració dels productes comprats. L'àrea privada també li permetrà anar a modificar les seves dades personals i gestionar el contingut de la seva wish-list. Si l'usuari és un treballador, l'àrea privada li permetrà accedir a les pàgines per donar d'alta nous articles en el catàleg o modificar/donar de baixa els articles existents. Si l'usuari és un administrador, a través de l'àrea privada també podrà gestionar els usuaris accedint a les pàgines corresponents.

Per entrar a l'àrea privada es demanarà el nom d'usuari i la contrasenya, sempre que l'usuari no s'hagi identificat prèviament. Això es farà a través de la pantalla de *login*, que després de recollir les dades redirigirà cap a l'àrea privada corresponent si l'autenticació de l'usuari és correcta. També es passarà per la pantalla de *login* quan l'usuari encara no s'hagi identificat i intenti fer una comanda. El diagrama anterior, per simplicitat, no reflecteix totes les possibilitats que porten a la pantalla de login.

Els apartats següents mostren orientativament l'aspecte i la funcionalitat que tindran algunes d'aquestes pàgines.

3.2.2. Navegació del catàleg/Cerca en el catàleg

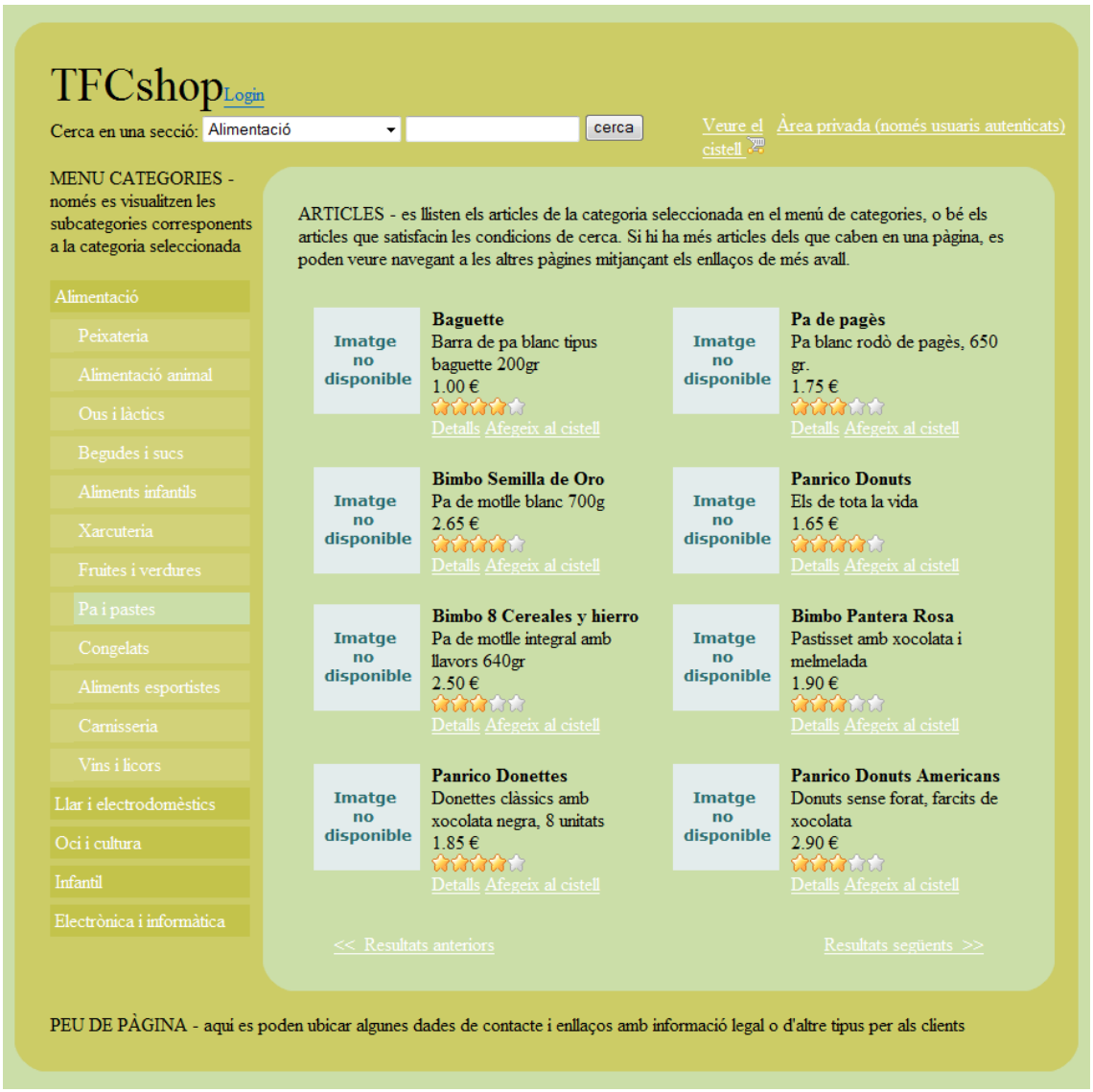


Figura 4. Pàgina principal de cerca/navegació del catàleg

3.2.3. Detalls del producte

Figura 5. Pàgina de detalls del producte

3.2.4. Cistell de la compra

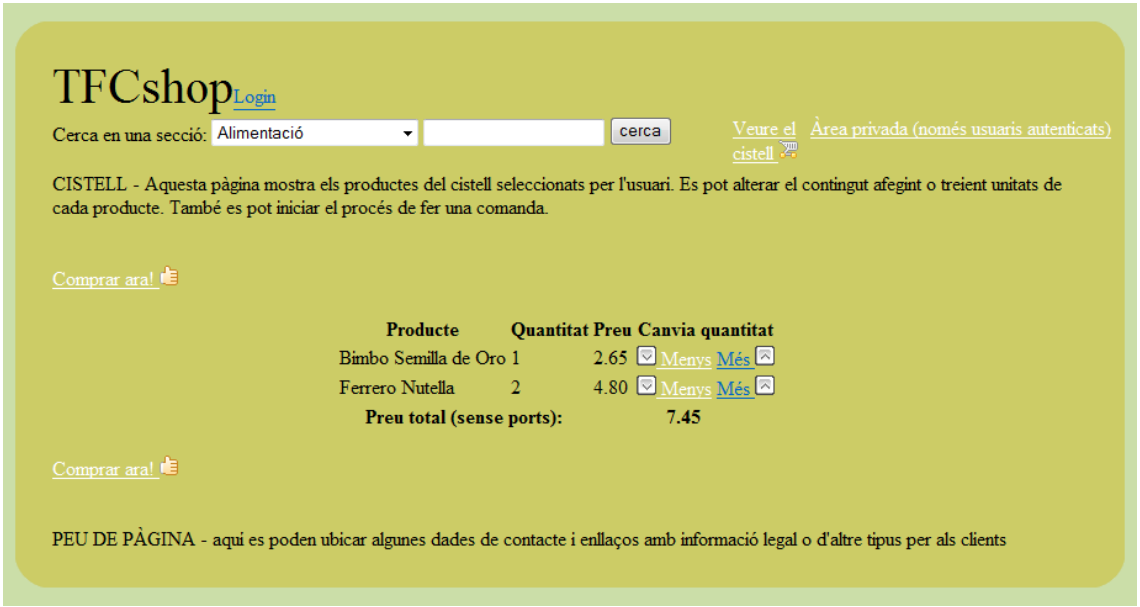


Figura 6. Pàgina del cistell de la compra

3.2.5. Comandes

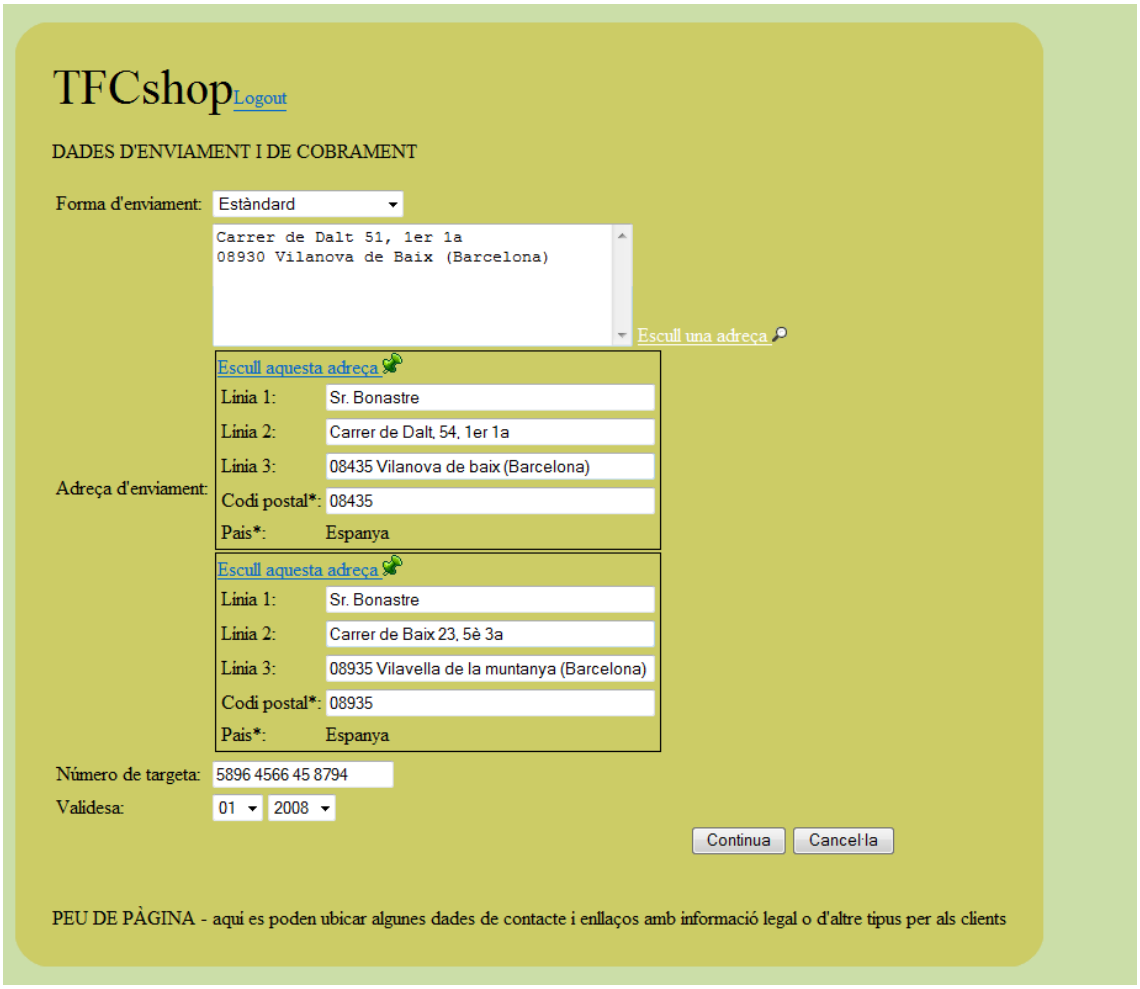


Figura 7. Pàgina amb el formulari per efectuar una comanda

Per a fer una comanda, hi ha una seqüència de tres pàgines: la primera, en què es demanen les dades d'enviament i de pagament (veure Figura 7. Pàgina amb el formulari per efectuar una comanda); una segona pàgina, en què es mostren altre cop aquestes dades, juntament amb els articles sol·licitats i el preu, i es demana a l'usuari que confirmi la comanda (veure Figura 8. Pàgina de confirmació de les dades de la comanda); la darrera pàgina és una simple notificació a l'usuari que la seva comanda ha estat enregistrada correctament.

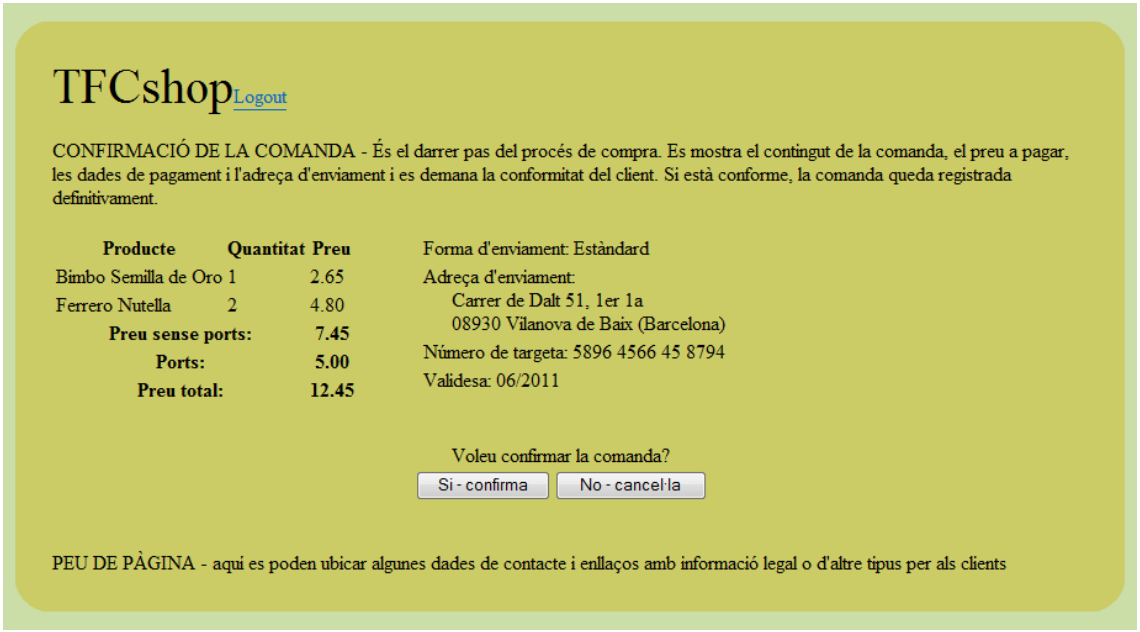


Figura 8. Pàgina de confirmació de les dades de la comanda

3.2.6. Gestió del catàleg

Figura 9. Pàgina principal de gestió dels articles del catàleg

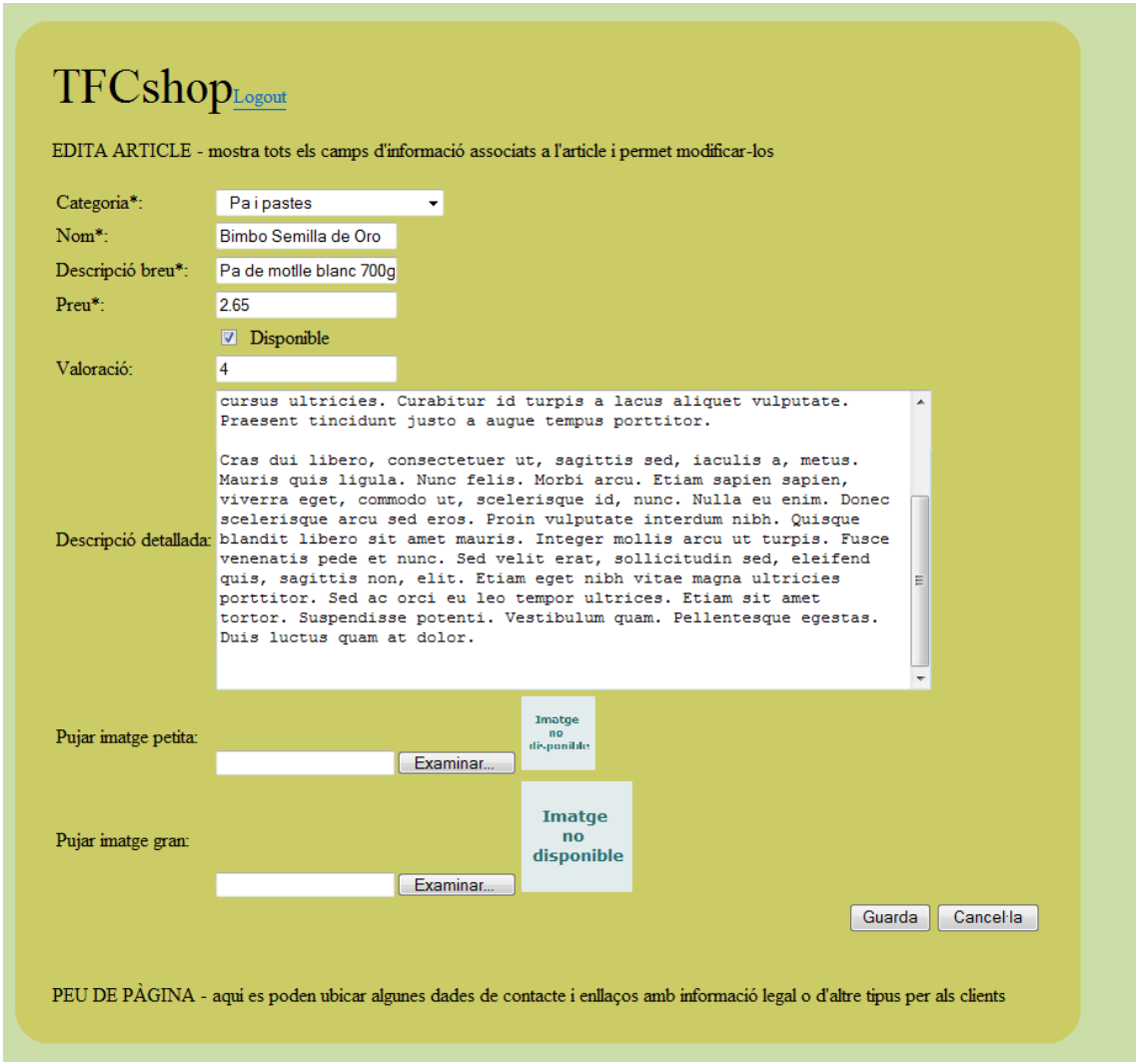


Figura 10. Pàgina de modificació d'articles del catàleg

La pàgina per a donar d'alta un article és idèntica a la d'edició, i la pàgina de mostrar un article també, però no permet editar els camps.

4. Disseny

4.1. Descripció general de l'arquitectura

Per ser un requisit implícit en aquesta àrea de TFC, el projecte es definirà basant-se en una arquitectura Java Enterprise Edition (JavaEE). En concret, s'utilitzarà la versió 5 de JavaEE.

L'aplicació es dissenyarà en tres capes: presentació (capa web), lògica de negoci (capa EJB) i persistència de dades (model de dades).

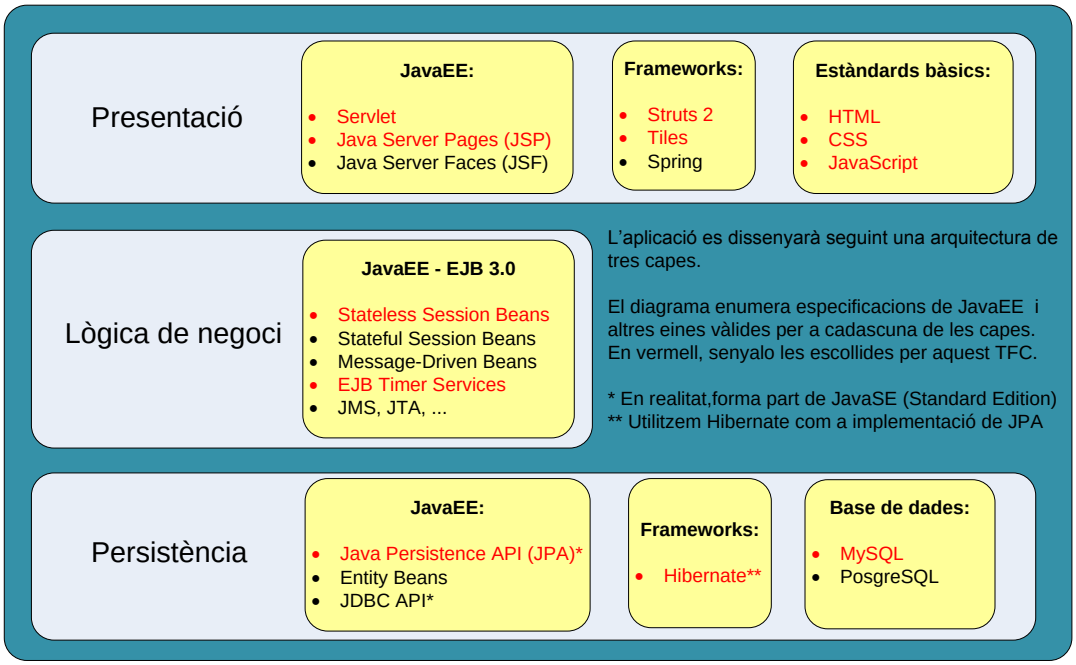


Figura 11. Descripció general de l'arquitectura

4.1.1. Capa de presentació

La capa de presentació consistirà en una interfície web implementada mitjançant pàgines JSP en el context d'un patró MVC (*Model-View-Controller*).

Per a facilitar la implementació d'aquest patró de disseny, s'utilitzarà el *framework* Struts 2. Aquest *framework* es configura en el servidor d'aplicacions com un filtre de *servlet*, que intercepta les peticions del client i actua com a *Front Controller* i com a *Dispatcher*, reunint en un sol punt la realització d'una sèrie de tasques habituals i delegant finalment les tasques específiques a uns altres components, les accions. Les accions són la pedra angular del *framework*, responsables d'actualitzar el model i finalment indicar al *framework* quina vista s'ha d'utilitzar per interactuar amb l'usuari (s'utilitza un fitxer de configuració per a relacionar el resultat de l'execució d'una acció amb la selecció de la vista adequada). Aquesta forma de funcionar forma un patró de disseny conegut com a *Service to worker,* que es resumeix en la següent figura (Figura 12. Patró de disseny "Service to worker"):

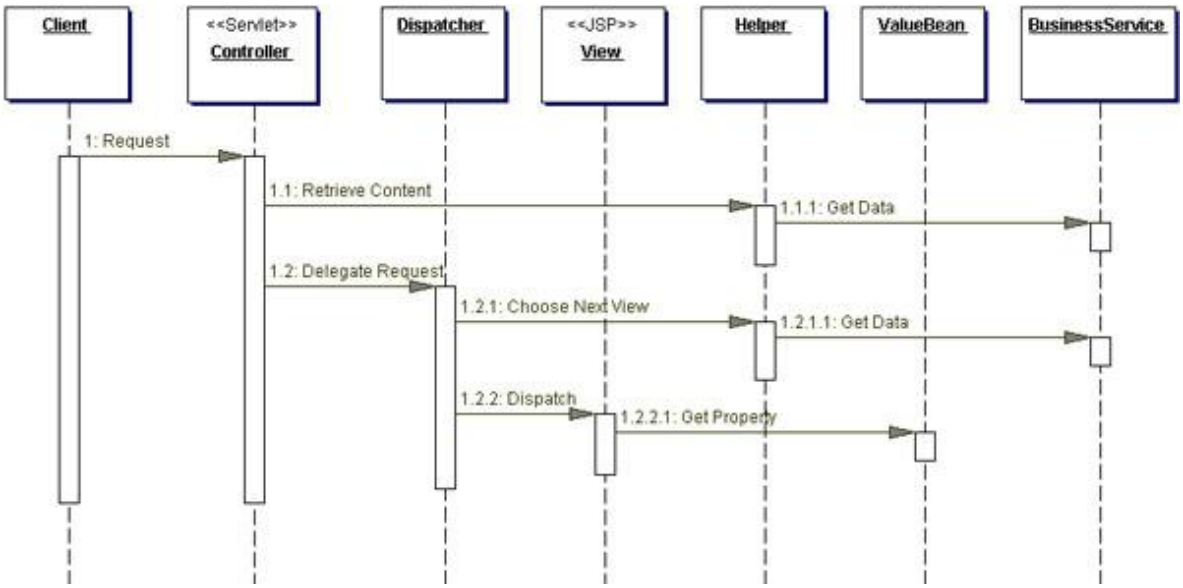


Figura 12. Patró de disseny "Service to worker"
(Extreta de http://java.sun.com/blueprints/corej2eepatterns/Patterns/ServiceToWorker.html)

Struts2 també incorpora la seva versió del patró *Intercepting Filter*. Prèviament a l'execució de l'acció, s'executa una cadena d'interceptors que realitzen una sèrie de tasques pròpies del *framework*, com és ara la recuperació dels paràmetres del formulari, la seva validació i el seu pas al bean de l'acció. A més, aquesta cadena és configurable, podent-s'hi afegir nous interceptors implementats per nosaltres mateixos o per tercers. La Figura 13. Patró de disseny "Intercepting Filter" il·lustra aquest patró:

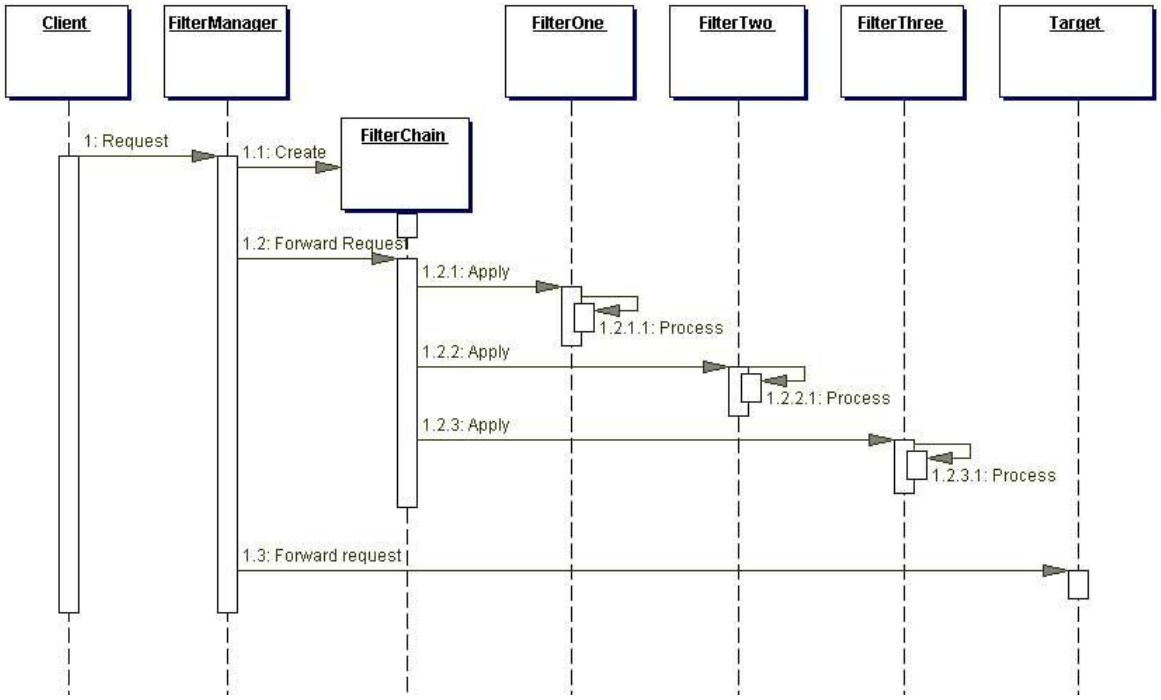


Figura 13. Patró de disseny "Intercepting Filter"
(Extreta de http://java.sun.com/blueprints/corej2eepatterns/Patterns/InterceptingFilter.html)

Per a estructurar les vistes, es farà servir el sistema de plantilles *Tiles*, que es troba disponible com un *plug-in* de Struts 2. Per a facilitar l'obtenció de les dades del model des de la vista, Struts2 proporciona una llibreria de *tags* JSP per a representar diferents tipus de camps de formulari, així com *tags* per al control del flux en la vista (estructures condicionals i iteracions).

D'aquesta manera, doncs, es simplificaran aspectes com la gestió de les dades dels formularis de les pàgines, la validació d'aquestes dades, la navegació entre pàgines, i la clara separació de l'accés a la lògica de negoci del que és pròpiament la presentació en HTML.

4.1.2. Capa de lògica de negoci

L'arquitectura JavaEE està pensada per a implementar la lògica de negoci mitjançant Enterprise JavaBeans (EJB), una especificació per al desenvolupament de components portables entre contenidors EJB de diferents proveïdors (fabricants de software). El contenidor proporciona de manera senzilla els serveis avançats que els components poden necessitar, com és ara gestió de transaccions, seguretat, gestió de fils d'execució (*threads*) davant la concurrència d'usuaris, gestió del cicle de vida per guanyar escalabilitat, injecció de dependències, etc.

En aplicacions senzilles, també és raonable implementar la lògica de negoci mitjançant classes normals (POJO) utilitzades des dels Servlets que actuen com a *Controller* en el patró MVC (o, més aviat, des de les classes auxiliars o *helpers* en les quals deleguen el processament de les peticions, anomenades accions en el context de Struts 2).

Els requeriments de l'aplicació dissenyada per aquest projecte potser no justifiquen plenament la sobrecàrrega que suposa l'ús d'EJB, ja que pràcticament no hi ha lògica de negoci, però pensant en un futur creixement de l'aplicació i en l'interès formatiu, la lògica de negoci s'implementarà mitjançant els diferents tipus de Session Beans especificats per EJB3.0.

4.1.3. Capa de persistència

El model de dades s'implementarà amb classes Java simples (POJO: *Plain Old Java Objects*) que representaran en memòria les entitats i relacions resultants del disseny del model de dades.
La persistència de les dades es farà en una base de dades relacional. Pel fet de ser gratuïta a la vegada que proporciona tota la funcionalitat necessària, s'utilitzarà una versió recent de MySQL.
El mapeig objecte-relacional (ORM: *Object–Relational Mapping*) es farà utilitzant l'API de persistència de Java (JPA: *Java Persistence API*), disponible a partir de la versió 5 de JavaSE, i que és el mecanisme de persistència recomanat per EJB3 en comptes de les antigues *Entity Beans* introduïdes per les especificacions anteriors de EJB . Per a fer les consultes, l'especificació posa a la nostra disposició el llenguatge EJBQL, una variant de SQL adequada a l'ús d'objectes.

Aquest sistema de persistència es basa en el patró Domain Store, que presenta l'avantatge, respecte el patró DAO (Data Acces Object), que permet a l'aplicació treballar directament amb les classes Java del model, sense necessitat que aquestes incloguin mètodes de persistència, resultant així més reutilitzables. L'element principal de JPA és la interfície EntityManager, que proporciona una sèrie de mètodes per a cercar i guardar entitats a la base de dades.

Com a proveïdor del SPI (Service Provider Interface) per a JPA s'utilitzarà la coneguda llibreria ORM Hibernate. També podria utilitzar-se directament, sense passar a través de la interfície JPA. No obstant, prefereixo veure com funciona l'estàndard de Java.

4.1.4. Servidor d'aplicacions

Com a servidor d'aplicacions JavaEE s'utilitzarà JBoss, un programari gratuït i de codi obert que ofereix una solució completa JavaEE (contenidor de Servlets, contenidor de EJB3.0, etc.).

4.2. Disseny del model de dades

Analitzant els requeriments de l'aplicació, bàsicament a partir dels casos d'ús, apareixen clarament una sèrie d'entitats interrelacionades. Com a evolució d'aquest primer anàlisi, arribem al disseny del següent model de dades per a l'aplicació:

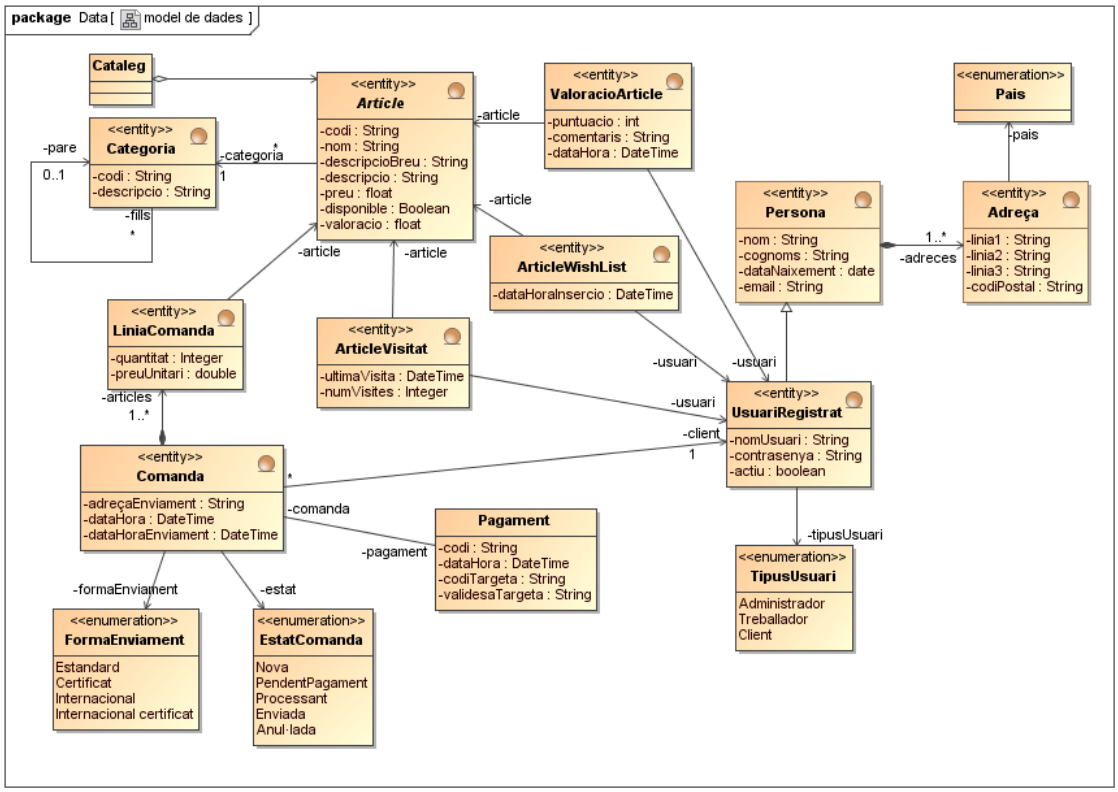


Figura 14. Diagrama del model de dades

Observacions:

- ✓ El camp adreçaEnviament de l'entitat Comanda no fa referència a una entitat Adreça, sinó que és de tipus String. Això té dos motius: 1) es permet enviar una comanda a adreces que no constin en les dades personals del client, i 2) si el client canvia les seves dades personals, les adreces de les comandes que hagi fet amb anterioritat romanen inalterades.

- ✓ ArticleVisitat no sorgeix de cap cas d'ús especificat anteriorment, però en els requeriments inicials sí que es considerava la possibilitat de mantenir un registre dels articles pels quals s'interessa un client.

- ✓ Com es pot veure, s'utilitza una mateixa classe d'entitat per als diferents tipus d'usuari, i s'utilitza el valor de l'atribut tipusUsuari per a diferenciar-ne el tipus. A partir dels actors presents en els casos d'ús, es derivaria la necessitat de tenir 3 tipus d'entitats: Client, Treballador i Administrador, amb els mateixos atributs llevat de les adreces, que només

serien necessàries per als clients. Per tant l'alternativa seria tenir una jerarquia de classes per als diferents tipus d'usuari. A partir d'una classe abstracta UsuariRegistrat, tindríem una subclasse Client, que afegiria un atribut per les adreces, i una subclasse Treballador (sense atributs addicionals) que al seu tron tindria una subclasse Administrador. En el diagrama de classes anterior, seria la classe Client la que estaria relacionada amb Comanda, ValoracioArticle, ArticleVisitat, etc. No obstant, per simplicitat, he descartat aquesta alternativa en favor del disseny exposat en el diagrama anterior. D'aquesta manera, es podran tractar els usuaris d'una manera totalment homogènia en el codi de l'aplicació.

4.3.Disseny de la lògica de negoci

4.3.1. Consideracions generals

Com ja s'ha argumentat, la lògica de negoci s'implementarà mitjançant EJB3.0, concretament, *Session Beans*, que seran classes POJO amb anotacions de Java per a identificar els mètodes relacionats amb el cicle de vida especificat per EJB. Aquests *beans* oferiran servei a altres capes mitjançant el patró *Session Façade*. La majoria de serveis d'aquesta capa tindran a veure amb la consulta de dades i persistència d'entitats, de manera que s'aprofitarà la capacitat d'injecció de dependències de EJB3 per a obtenir una instància del *EntityManager* de JPA necessari per a fer operacions amb la base de dades per als diferents tipus d'entitats. Per defecte, l'abast de les transaccions es limitarà a cadascun dels mètodes exposats pels *beans*.

Els components de negoci es podran obtenir des de qualsevol punt de l'aplicació mitjançant el patró *Service Locator*.

En la capa de presentació, l'accés als serveis de la capa de negoci es farà accedint directament als objectes retornats pel *ServiceLocator*, ja que els serveis definits són prou senzills (no s'utilitzarà, per tant, el patró *Business Delegate*).

En general, no caldrà una granularitat gaire fina en l'accés a les dades, de manera que els mètodes dels *beans* retornaran objectes seguint el patró *Data Transfer Object* (DTO) per a fer el mínim de crides. Normalment el conjunt de les dades transferides correspondrà directament a una entitat del model de dades, de manera que es podrà utilitzar la pròpia classe del model com a DTO (suposant que sigui serialitzable) per a evitar la proliferació d'adaptadors trivials. Per a gestionar la recuperació d'atributs de tipus *Collection* amb cardinalitat potencialment elevada, els *beans* oferiran mètodes per a obtenir les dades per intervals (alternativament, es podria utilitzar el patró *Value List Handler*).

Per altra banda, com que els *beans* i la capa web residiran en el mateix servidor, la implementació dels *beans* serà local (no remota).

4.3.2. Components principals

El següent diagrama (Figura 15. Disseny dels components de negoci) mostra els components principals d'aquesta capa de d'aplicació, amb les seves interfícies i algunes classes auxiliars:

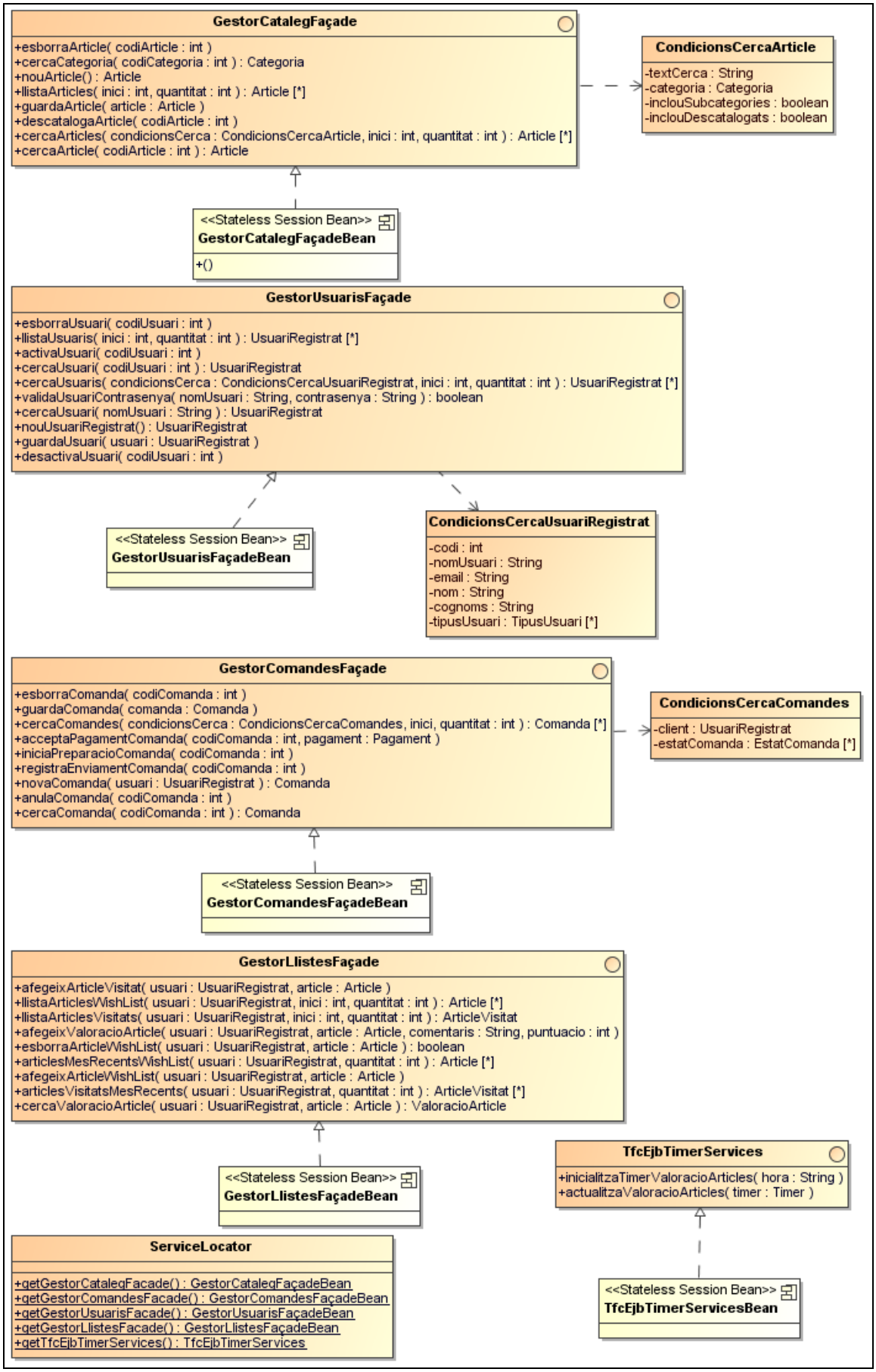


Figura 15. Disseny dels components de negoci

En el diagrama trobem els següents *beans*:

- ✓ *GestorCatalegFaçadeBean*: Stateless Session Bean que aglutina operacions relacionades amb la gestió d'articles i categories del catàleg.
- ✓ *GestorUsuarisFaçadeBean*: Stateless Session Bean que aglutina operacions relacionades amb la gestió d'usuaris
- ✓ *GestorLlistesFaçadeBean*: Stateless Session Bean que aglutina operacions relacionades amb la gestió de llistes (wish-list, historial de visites, valoracions dels articles)
- ✓ *GestorComandesFaçadeBean*: Stateless Session Bean que aglutina operacions relacionades amb la gestió de comandes.
- ✓ *TfcEjbTimerServicesBean*: Stateless Session Bean que conté el mètode que diàriament s'executarà per a actualitzar la puntuació mitjana dels productes a partir de les valoracions que n'hagin fet els clients. L'execució periòdica d'aquest mètode s'aconsegueix utilitzant l'*EJB Timer Service API* que forma de l'especificació EJB. El bean també disposa d'un mètode per a inicialitzar el temporitzador indicant-li a quina hora s'ha d'activar. El mètode d'inicialització s'ha de cridar quan es posa en marxa l'aplicació. Això es farà mitjançant una classe que implementi la interfície *ServletContextListener* per tal que, en carregar-se l'aplicació web, li sigui notificat aquest fet i pugui inicialitzar el temporitzador.

Tots els *beans* són de tipus *stateless*, ja que els seus mètodes no defineixen cap procés complex que requereixi mantenir l'estat entre les diferents crides d'un mateix client, sinó que són mètodes aïllats amb una finalitat molt concreta.

Pel que fa a les classes auxiliars, tenim:

- ✓ ServiceLocator, que permetrà accedir als *beans* des de qualsevol lloc de l'aplicació

- ✓ Classes que encapsulen paràmetres de cerca diversos. Quan un atribut té multiplicitat major que 1 vol dir que sobre aquell atribut es pot aplicar una condició de cerca de tipus OR.

Observació: en el diagrama, el modificador de tipus "[*]" no representa forçosament un *array* de Java, sinó que probablement serà algun tipus de *Collection* .

4.4. Model dinàmic

Els mètodes dels objectes de negoci anteriors pretenen respondre a la majoria de les necessitats detectades en l'anàlisi dels casos d'ús. En aquest apartat, s'inclouen alguns diagrames d'estat i d'interacció (seqüència) per a exemplificar i justificar alguns d'aquests mètodes. En el cas dels diagrames de seqüència, a causa de les limitacions d'espai, s'ha intentat simplificar-los al màxim, obviant tot allò que resulti més o menys transparent o evident (per exemple: l'ús del *ServiceLocator*, el controlador de Struts2, la creació d'alguns objectes, etc).

4.4.1. Cerca/Navegació del catàleg/Veure detalls del producte

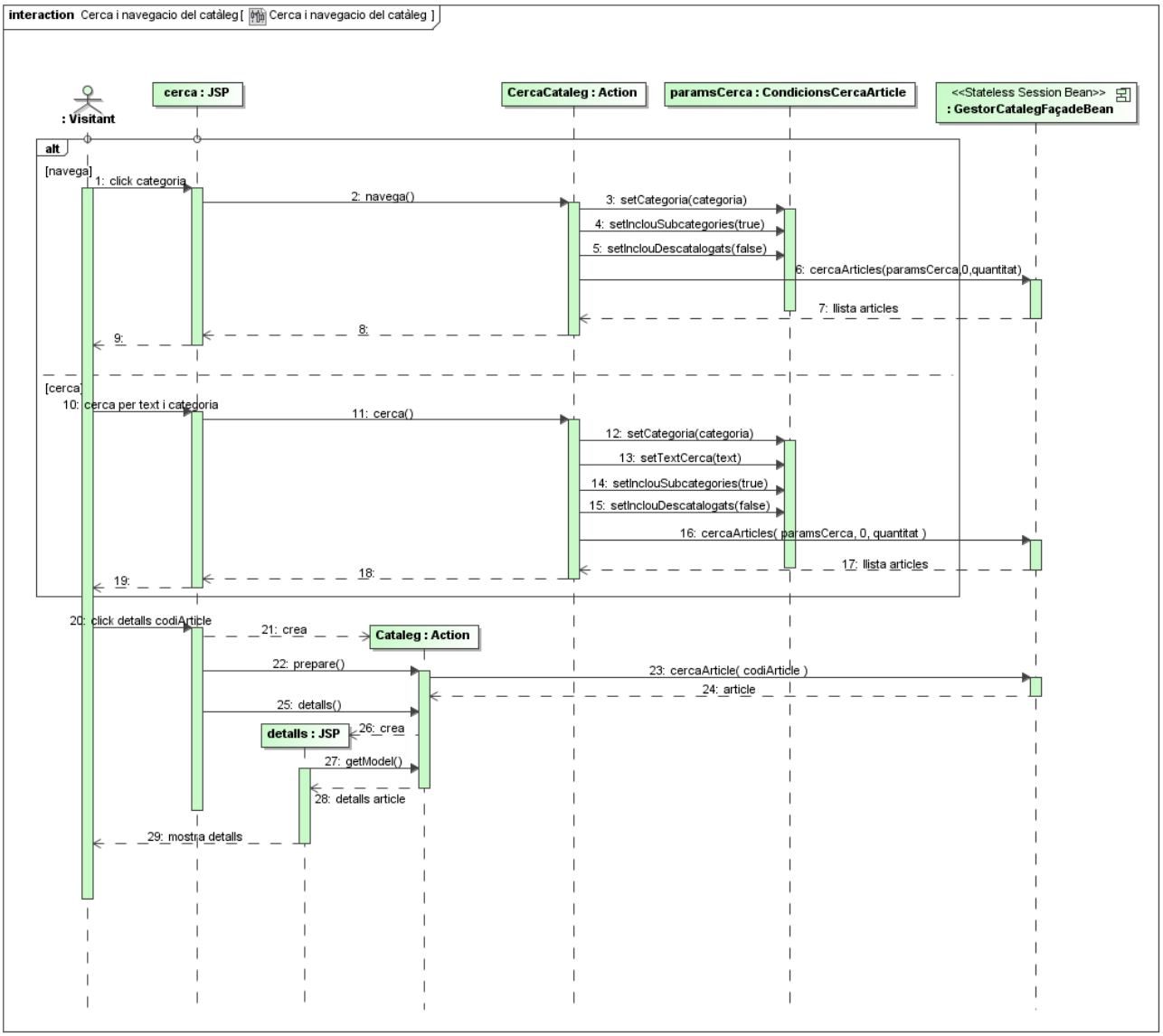


Figura 16. Diagrama de seqüència: cerca/navegació del catàleg i detalls del producte

En aquest diagrama, si l'actor principal fos un Client, en comptes d'un Visitant, la crida per visualitzar els detalls de l'article també enregistraria la visita a l'històric d'articles visitats pel client, mitjançant el mètode afegeixArticleVisitat del component GestorLlistesFaçadeBean. En el cas d'un Visitant, s'utilitzen cookies per a mantenir un historial recent que no es guarda de manera persistent.

4.4.2. Afegir al cistell

El següent diagrama es pot veure com una continuació de l'anterior, i mostra com un usuari afegeix al cistell l'article que estava visualitzant. El cistell és un objecte que es manté a nivell de sessió web i s'actualitza cada cop que s'afegeix un ítem o es modifica la quantitat dels existents. El diagrama també mostra com l'usuari incrementa en una unitat la quantitat d'aquest article que vol comprar.

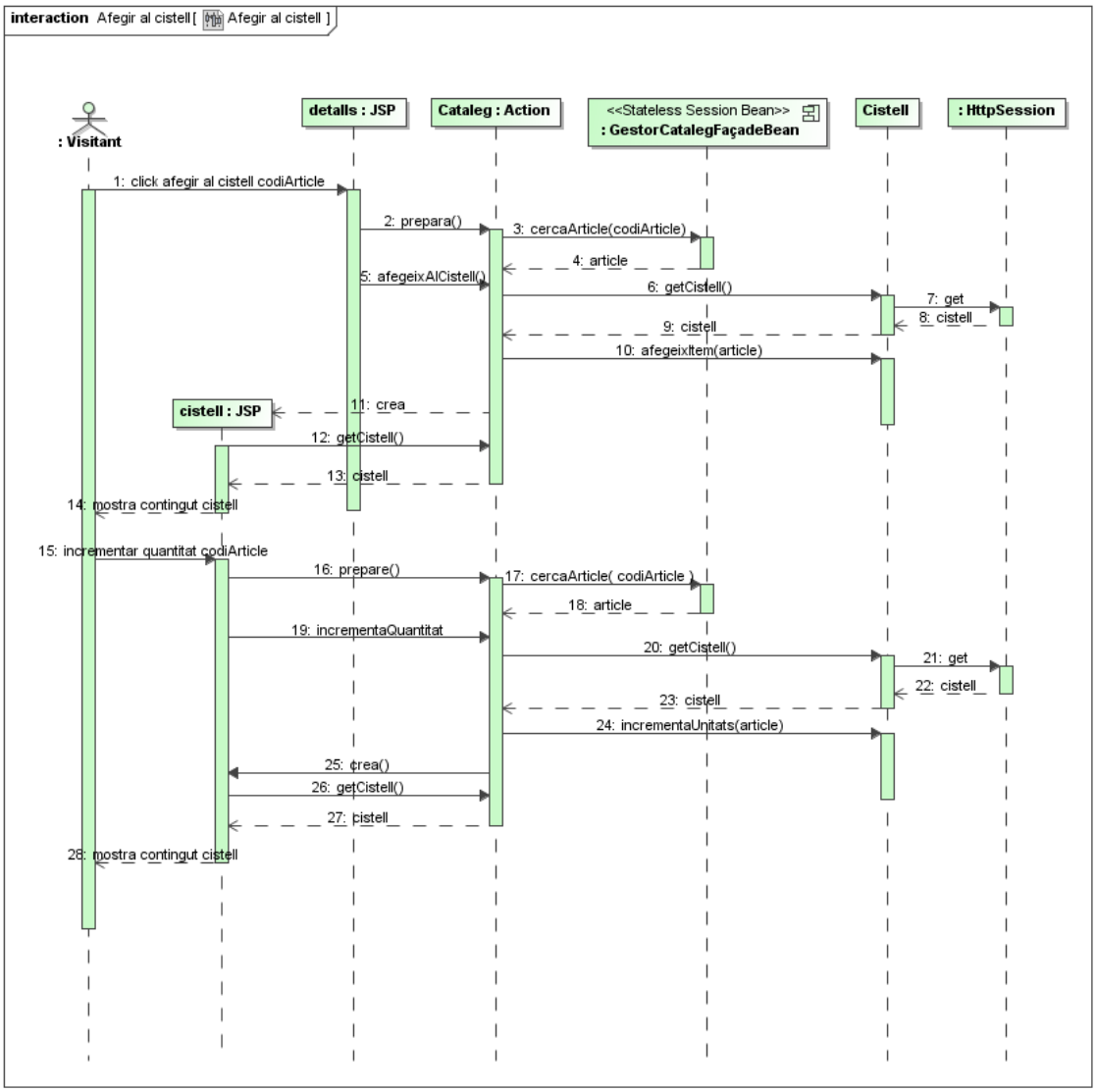


Figura 17. Diagrama de seqüència: afegir un article al cistell

4.4.3. Fer una comanda

El diagrama següent podria ser la continuació de l'anterior, obviant el pas intermedi corresponent a l'autenticació del visitant com a client.

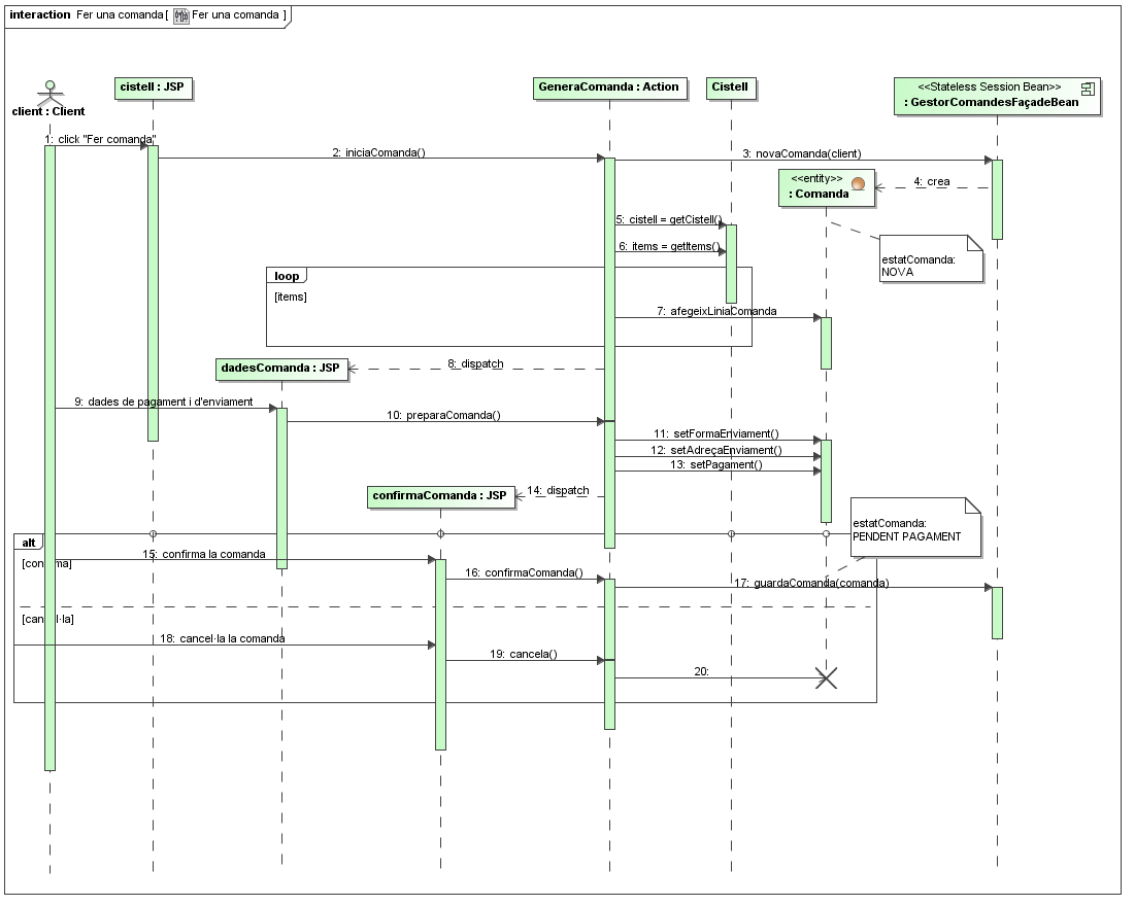


Figura 18. Diagrama de seqüència: fer una comanda

L'estat d'una comanda evoluciona segons el següent diagrama:

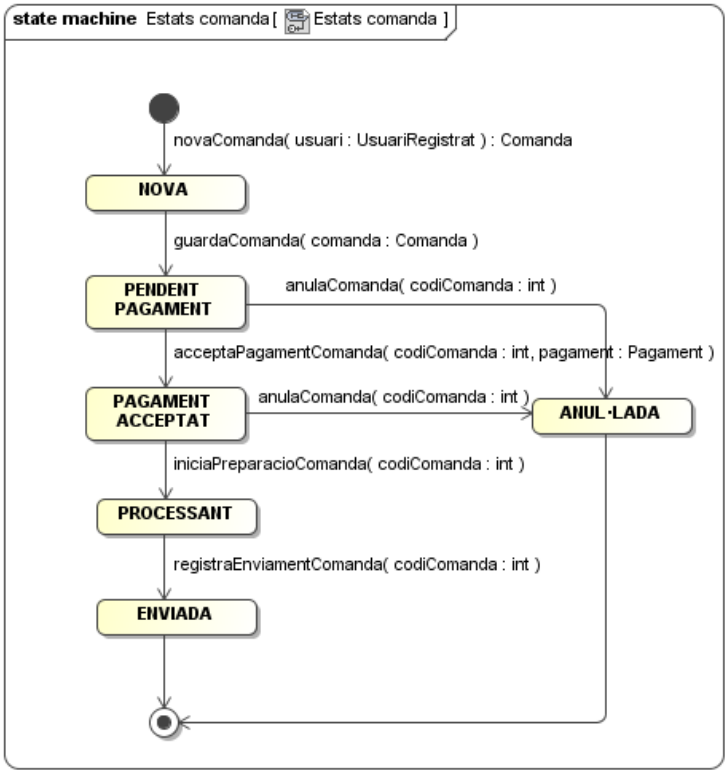


Figura 19. Diagrama d'estats d'una comanda

Cal remarcar que els disparadors que provoquen el canvi d'estat no són crides al propi objecte Comanda, sinó que són crides al GestorComandesFaçadeBean. A més, el pas als estats PAGAMENT ACCEPTAT, PROCESSANT i ENVIADA depenen de la crida d'aquests mètodes per part d'un subsistema intern que no forma part d'aquest TFC.

4.4.4. Gestió del catàleg

Aquest diagrama il·lustra el cas d'ús "Gestionar el catàleg", en el supòsit que l'usuari decideixi modificar un article. Si decideix crear un article nou, el procés és molt similar, però sense la cerca inicial. Si decideix esborrar l'article, també és similar, però cridant directament descatalogaArticle en comptes de passar per la pantalla d'edició i després cridar guardaArticle.

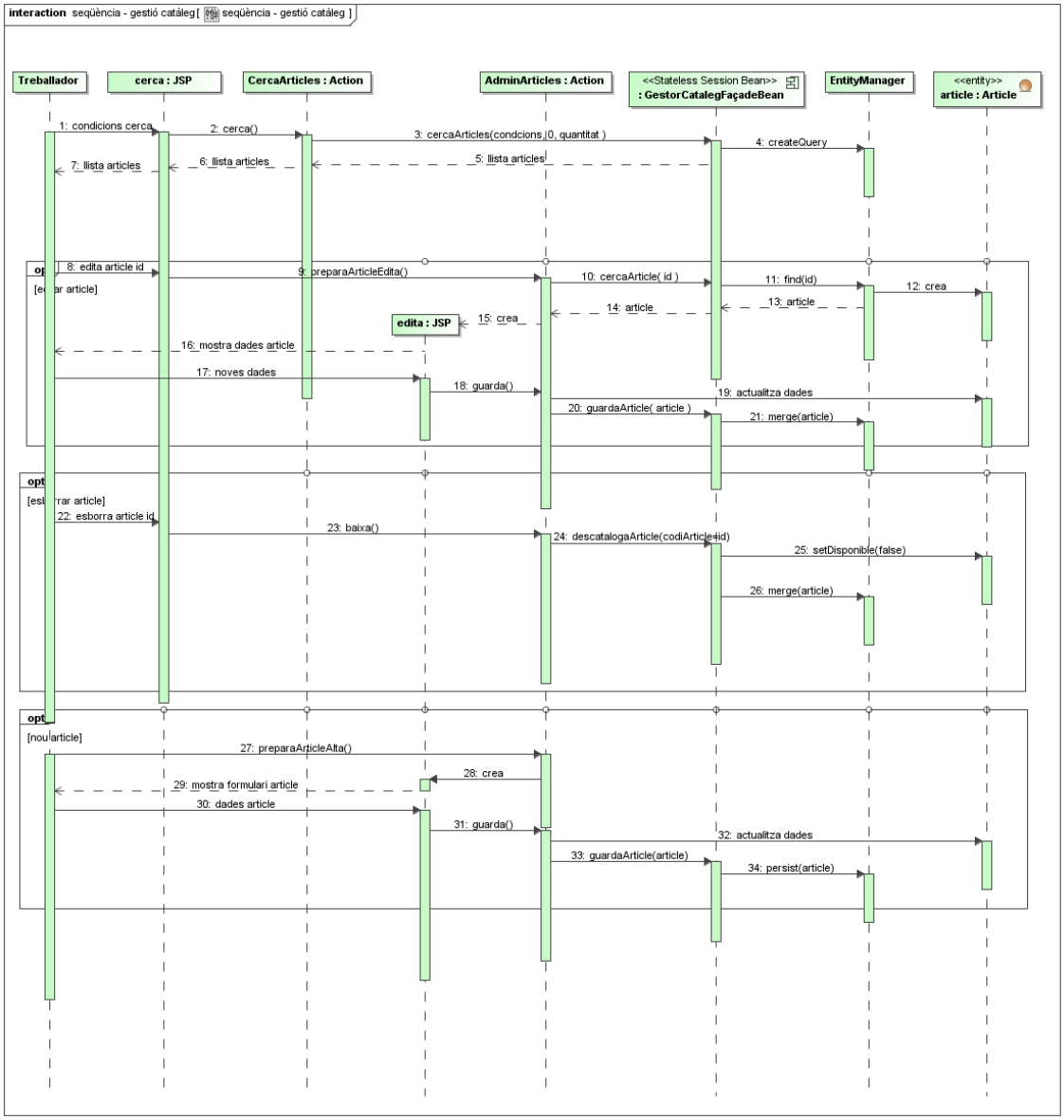


Figura 20. Diagrama de seqüència: gestió del catàleg

4.4.5. Actualitzar la valoració dels productes

Com ja s'ha dit, aquest cas d'ús no és iniciat per cap dels actors identificats anteriorment, sinó que es tracta d'un procés que s'executa periòdicament, cada cop que expira un temporitzador. Segons aquest disseny, el temporitzador es configura quan el contenidor de *servlets* arrenca l'aplicació, mitjançant una instància de *ServletContextListener*. Un cop configurat, cada cop que expira s'executa el mètode actualitzaValoracioArticles del *bean* TfcEjbTimerServicesBean, que calcula el valor mitjà de les puntuacions proporcionades pels clients que han valorat cada producte, i amb aquesta mitjana actualitza l'atribut corresponent del producte en qüestió.

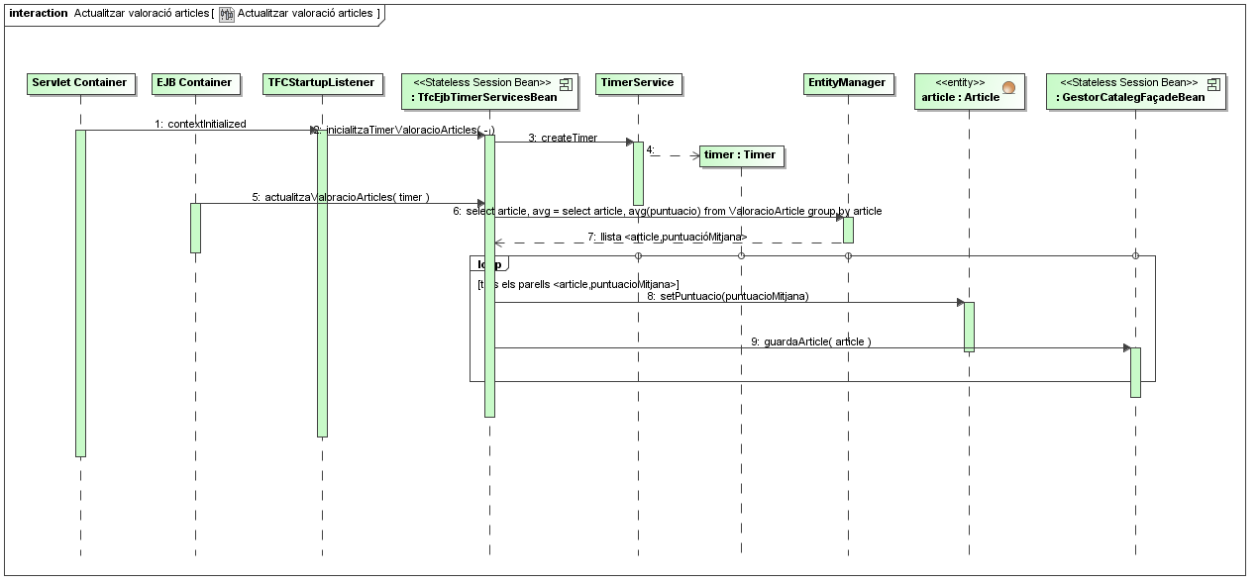


Figura 21. Diagrama de seqüència: actualitzar la valoració dels productes

4.4.6. Altes casos

La resta de casos d'ús no s'il·lustren en aquest apartat perquè o bé seran molt senzills o secundaris, o bé seran molt similars a algun altre cas, i per tant es poden imaginar les seqüències a seguir a partir dels diversos objectes de negoci dissenyats. Per exemple: la gestió d'usuaris, i la consulta i anul·lació de comandes, seran similars a la gestió d'articles del catàleg.

4.5. Seguretat

Els casos d'ús especifiquen quan és necessària l'autenticació dels usuaris per a actuar en el sistema. En aquesta fase de disseny, decidim que no s'utilitzarà el mecanisme de seguretat declarativa de J2EE, sinó que es farà una autenticació mitjançant formulari web i guardant l'usuari en sessió. Si després de la implementació de tota l'aplicació hi hagués temps, es podria intentar d'utilitzar el *DatabaseServerLoginModule* de JBoss, que es basa en l'estàndard JASS i autentica l'usuari contra base de dades per a obtenir el *Principal* i els rols associats, que després també estarien disponibles en els accessos als *Enterprise Java Beans* per a fer les comprovacions de seguretat oportunes.

4.6. Paquets i desplegament

L'estructuració del codi font en paquets respondrà bàsicament a la separació de la capa web de la resta. Així, per una banda tindrem els *beans*, les seves classes auxiliars i les classes del model de dades. Per altra banda, tindrem les accions de Struts i les seves classes auxiliars. A nivell de desplegament, consistirà en un arxiu .ear que empaqueti un .war amb la part web i un .jar que empaqueti la part d'EJB i del model de dades.

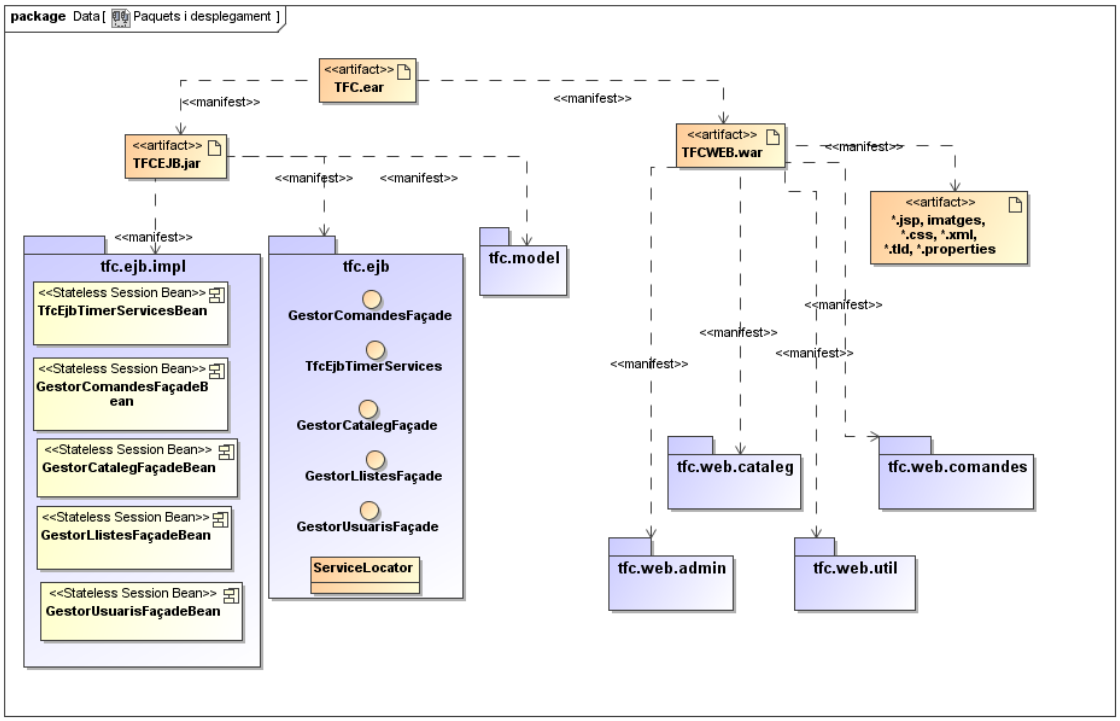


Figura 22. Paquets i desplegament

5. Implementació

A nivell funcional, l'aplicació està acabada, ja que tots els casos d'ús han estat implementats. També s'han pogut incorporar algunes de les funcionalitats secundàries que es proposaven com a opcionals en el capítol de requeriments: les *wish-list*, la llista d'articles visitats recentment, i la valoració dels articles per part dels clients.

En general, no han aparegut divergències remarcables respecte la fase de disseny. A continuació s'expliquen alguns dels detalls d'implementació i es senyalen les possibles variacions respecte el disseny original.

5.1. Capa de presentació

Tal com estava previst, s'ha utilitzat Struts2 com a *framework* per a la construcció dels formularis, recol·lecció de dades i gestió de la navegació.

5.1.1. Codificació en HTML i internacionalització de la interfície gràfica

A l'hora d'implementar les pàgines de l'aplicació amb Struts2, per defecte he deixat que ell s'encarregui de la generació del codi HTML dels formularis (theme="css_xhtml" o theme="xhtml") per a facilitar el desenvolupament inicial, ja que ell mateix escriu el text (etiqueta) associat a cada camp del formulari, text que es pot especificar mitjançant fitxers de *properties*, afavorint així la internacionalització de l'aplicació i facilitant el manteniment dels literals de text. Pel mateix motiu, també he procurat que no quedin textos literals en les pàgines JSP, utilitzant el tag <s:text>, que permet obtenir els textos dels fitxers de properties.

D'altra banda, no sempre resulta fàcil integrar codi HTML fet a mà amb el codi generat automàticament, de manera que en diverses pàgines he acabat utilitzant theme="simple" per a que Struts2 es limiti a generar els camps del formulari, sense intentar estructurar la pàgina. Per al disseny de la pàgina he intentat separar-lo el màxim possible en la fulla d'estils CSS. Per a dinamitzar una mica algunes pàgines, he utilitzat la llibreria de javascript jQuery.

Finalment, comentar l'ús de Tiles com a plug-in de Struts2 per a estructurar de manera uniforme totes les pàgines de l'aplicació. Cada pàgina s'estructura segons una de les següents tres plantilles:

- ✓ Plantilla amb capçalera, subcapçalera, menú esquerra, contingut principal i peu de pàgina
- ✓ Plantilla amb capçalera, subcapçalera, contingut principal i peu de pàgina
- ✓ Plantilla amb capçalera, contingut principal i peu de pàgina

Per això ha calgut definir Tiles com a *result-type* per defecte en el fitxer de configuració de struts, struts.xml, de manera que quan una acció redirigeixi cap a un resultat, aquest sigui processat per Tiles:

```
<package name="tfc-default" namespace="/" extends="tiles-default">
      <result-types>
            <result-type name="tiles"
```

```
            class="org.apache.struts2.views.tiles.TilesResult"
            default="true"/>
    </result-types>
    …
</package>
```

Les pàgines compostes per Tiles a partir de les plantilles es defineixen en el fitxer tiles.xml.

5.1.2. Validació dels formularis

Per validar els camps dels formularis, he utilitzat el *framework* de validació de Struts2. Les validacions d'obligatorietat, mida màxima i format de les dades dels camps dels formularis s'ha especificat mitjançant fitxers XML ubicats en l'arbre de directoris al mateix nivell que les classes de les accions de Struts2 associades als formularis. Aquests fitxers s'anomenen seguint la convenció ClasseOInterficieDeLAccio[-aliasDeLAccio]-validation.xml (la part entre claudàtors és opcional). Així, tenim els següents fitxers de validació:

- ✓ AdminUsuaris-validation.xml: valida els formularis d'alta, modificació i auto-registre d'usuaris.
- ✓ AdminArticles-validation.xml: valida els formularis d'alta i modificació d'articles.
- ✓ GeneraComanda-PreparaComanda-validation.xml: valida el formulari que demana les dades de pagament i d'enviament de la comanda.
- ✓ Login-validation.xml: valida que s'introdueixin usuari i contrasenya en el formulari de login.

Per posar l'exemple més senzill, mostro a continuació el Login-validation.xml:

```
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator
1.0.2//EN"
       "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
    <field name="nomUsuari">
        <field-validator type="requiredstring">
            <message>Aquest camp és obligatori</message>
        </field-validator>
        <field-validator type="stringlength">
            <param name="maxLength">50</param>
            <message>Valor no vàlid</message>
        </field-validator>
    </field>
    <field name="contrasenya">
        <field-validator type="requiredstring">
            <message>Aquest camp és obligatori</message>
        </field-validator>
        <field-validator type="stringlength">
            <param name="maxLength">50</param>
            <message>Valor no vàlid</message>
        </field-validator>
    </field>
</validators>
```

Quan ha calgut alguna comprovació addicional que els validadors disponibles no permetien de fer, en comptes de crear un nou tipus de validador i afegir-lo en el fitxer XML corresponent, he utilitzat l'altre mecanisme de validació de Struts2, corresponent en afegir un mètodes validate[AliasDeLAccio] (la part entre claudàtors és opcional). Un exemple, el tenim a AdminUsuaris, on s'utilitza abans de guardar un usuari per a comprovar que el seu nom d'usuari proporcionat no col·lideixi amb el d'un altre usuari existent a la base de dades:

```
public void validateGuarda() {
	// només hi pot haver un usuari amb un determinat nom d'usuari
	UsuariRegistrat registrat = guf.cercaUsuari(usuari.getNomUsuari());
	if(registrat!=null && registrat.getCodi()!=usuari.getCodi()) {
		addFieldError("nomUsuari",
			getText("administraUsuaris_nomUsuari.validacio")
		);
	}
}
```

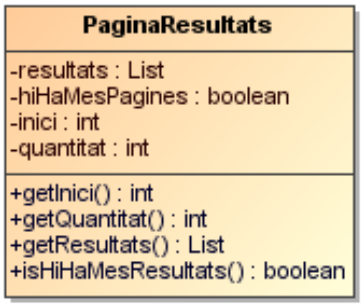
5.1.3. Parametrització mitjançant un fitxer de *properties*

Per a poder fer fàcilment alguns canvis sense haver de tocar el codi, he extret en un fitxers de *properties* (tfcshop.properties) alguns paràmetres, bàsicament el nombre de resultats per pàgina de les diferents taules de resultats (catàleg, usuaris, wish-list, articles visitats recentment, valoracions d'un article per part d'altres usuaris). Aquests paràmetres de configuració són accessibles mitjançant la classe d'utilitat tfc.web.util.ConfigUtil.

5.2. Capa EJB

Tal com es preveia en el disseny, l'accés a la lògica de negoci des de la capa web es fa obtenint el bean EJB que ofereix la funcionalitat necessària mitjançant un ServiceLocator.

Tot s els accessos a la base de dades es troben encapsulats en els EJB. Com que algunes consultes poden retornar un nombre elevat de resultats, i com que en l'aplicació és freqüent la necessitat de paginar aquests resultats, moltes de les operacions de consulta retornen un objecte PaginaResultats, que no estava previst en el disseny original. Allà es preveia només la possibilitat d'especificar l'interval de resultats a retornar, però es feia complicat determinar el total de resultats disponibles, cosa que s'ha simplificat amb el disseny utilitzat finalment. Així, quan es demana un interval amb la finalitat de paginar els resultats, es cerca un element de més que no es retorna a qui fa la crida, i això permet determinar si hi ha més pàgines disponibles. El tipus PaginaResultats que encapsula tota la informació resultant és el següent:



5.3. Model de dades

La implementació del model de dades és bastant directa. A l'hora d'implementar les relacions d'herència a la base de dades, he optat per utilitzar una sola taula per les entitats Persona i UsuariRegistrat, ja que pràcticament no tenen atributs diferents i, a més, en l'aplicació actualment totes les persones són usuaris (en realitat no caldria definir Persona com a entitat). Això es fa mitjançant les següents anotacions de la classe Persona:

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="tipusPersona")
@DiscriminatorValue("Persona")
```

I les següents per a la classe UsuariRegistrat:

```
@Entity
@DiscriminatorValue("UsuariRegistrat")
```

En les anotacions dels atributs que comporten una relació amb altres entitats, en general he especificat fetch=FetchType.EAGER per a que es facin directament les combinacions de taules necessàries per a crear els objectes Java (en comptes d'esperar a que el codi client ho requereixi), i així garantir l'accessibilitat de les dades un cop aquestes hagin estat retornades pels mètodes dels EJB i, per tant, es trobin fora del context del gestor de persistència. Pel que fa al tipus SQL de les columnes, els he deixat tots per defecte excepte el camp "descripció" del tipus Article i el camp "comentaris" del tipus ValoracioArticle , en què he especificat la mida per a que Hibernate (JPA) creï la columna corresponent com a mediumtext (recordem que la base de dades és MySQL) en comptes de varchar.

En l'*Annex II. Sentències DDL per a crear les taules de la base de dades*, es pot trobar la definició de les taules generades per Hibernate.

5.4. Seguretat

En la implementació final, s'autentiquen els usuaris via base de dades, guardant l'usuari en la sessió web, en comptes d'utilitzar el mecanisme estàndard de seguretat declarativa de JavaEE. Com a conseqüència, tenim tota la seguretat implementada via programa i lligada a la capa web.

Amb el temps disponible, no he pogut dedicar més esforços a investigar l'ús de JAAS per associar els rols amb els usuaris definits en la base de dades. Per tant, tampoc té sentit entrar a definir les qüestions de seguretat dels *beans* EJB.

La comprovació que l'usuari hagi estat autenticat abans d'accedir a un recurs protegit s'ha implementat mitjançant un interceptor de Struts2. A continuació revisem breument en què consisteixen els interceptors.

Per a cada acció de Struts2 definida a struts.xml, es pot especificar una llista d'interceptors (*interceptor-stack*) que seran executats seqüencialment abans que l'acció. Cada interceptor implementa la interfície import com.opensymphony.xwork2.interceptor.Interceptor i té accés al context de la invocació. En el moment d'executar-se, pot fer les tasques que consideri oportunes i després permetre que es continuï executant la resta d'interceptors fins arribar a executar l'acció de Struts, o bé trencar la cadena i redirigir cap a alguna altra pàgina. Struts2 defineix algunes llistes d'interceptors i per defecte utilitza la que s'anomena *defaultStack*. Els interceptors que conté fan diverses funcions bàsiques del framework, com recollir els paràmetres de la petició HTTP i passar-los al bean que representa l'acció, validar els paràmetres, etc.

Per a comprovar l'autenticació de l'usuari i que disposa de l'autorització necessària, he creat un interceptor anomenat AuthenticationInterceptor i he creat una nova llista d'interceptors anomenada *secureStack*, que afegeix el nou interceptor a la llista per defecte (*defaultStack*).:

```
<package name="tfc-default" namespace="/" extends="tiles-default">
…
	<interceptors>
		<interceptor name="authenticationInterceptor"
			class="tfc.web.login.AuthenticationInterceptor"/>

		<interceptor-stack name="secureStack">
			<interceptor-ref name="authenticationInterceptor"/>
			<interceptor-ref name="defaultStack"/>
		</interceptor-stack>

		<interceptor-stack name="secureParamsPrepareParamsStack">
			<interceptor-ref name="authenticationInterceptor"/>
			<interceptor-ref name="paramsPrepareParamsStack"/>
		</interceptor-stack>
	</interceptors>
…
</package>
```

A les accions que necessiten aquesta comprovació, els he associat la nova llista d'interceptors, per exemple:

```
<action name="TaulerAdmin">
	<interceptor-ref name="secureStack">
		<param name="authenticationInterceptor.tipusUsuari">
			ADMINISTRADOR
		</param>
	</interceptor-ref>
	<result>TaulerAdmin</result>
</action>
```

El nou interceptor es configura amb un paràmetre que permet especificar el tipus d'usuari que està autoritzat a executar l'acció. Quan s'executa, comprova si existeix un usuari en la sessió i, en cas que existeixi, si és del tipus especificat. Si es compleixen les dues condicions, es continua amb l'execució de l'acció de Struts2 invocada. En cas contrari, es redirigeix a la pàgina de login. Com a funcionalitat addicional, si l'usuari està autenticat i autoritzat i l'acció implementa la interfície UserAware, es passa l'usuari a l'acció cridant el mètode setUser de la interfície. D'aquesta manera fem molt senzill obtenir l'usuari actual a les accions que ho requereixen. De fet, aquesta darrera funcionalitat es podria implementar en un interceptor independent per tal que també estigués disponible en les seccions que no requereixen autorització. No obstant, quasi totes les accions que requereixen accedir a l'usuari actual es troben en seccions que requereixen autorització, de manera que deixarem la implementació així.

5.5. Càrrega d'imatges

L'aplicació permet associar fins a dues imatges a cada producte, una de gran i una de petita, per tal de poder mostrar la més adequada segons el disseny de la pàgina (tot i que en les dades de prova, les dues imatges són iguals). Aquestes imatges es graben a la carpeta que conté la resta d'imatges de l'aplicació, i el seu nom inclou el codi de l'article per permetre relacionar cada imatge amb l'article al qual pertany. Alternativament, s'hauria pogut guardar a la base de dades en una columna binària (LOB) . Tal com ho he implementat, hi ha l'inconvenient que quan es desplega una nova versió de l'aplicació s'esborra la carpeta temporal on JBoss havia desplegat la versió anterior, perdent-se d'aquesta manera les imatges pujades pels usuaris. Evidentment, és senzill canviar la carpeta per una de permanent en qualsevol altre lloc del disc, fins i tot es podria fer parametritzable mitjançant una propietat del fitxer tfcshop.properties, però per no complicar la instal·lació als avaluadors d'aquest TFC he optat per la solució més senzilla.

5.6. Fitxers de diari (*logs*)

JBoss utilitza la llibreria log4j per a gestionar els registres de logs. En el fitxer de configuració conf/jboss-log4j.xml del servidor es troben els diferents gestors de logs que utilitza JBoss per defecte. He afegit un nou gestor que reculli en un fitxer (tfcshop.log) només aquells registres de logs provinents de les classes pròpies d'aquesta aplicació, de manera que no es perdin entre la resta de missatges generats pel contenidor (tot i que els missatges propis també seran processats pels altres gestors definits per JBoss). A continuació indico el que s'ha afegit a jboss-log4j.xml (ometo el contingut original):

```
<log4j:configuration
      xmlns:log4j="http://jakarta.apache.org/log4j/" debug="false">
      …
   <appender name="TFC_FILE"
      class="org.jboss.logging.appender.DailyRollingFileAppender">
      <errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
      <param name="File" value="${jboss.server.log.dir}/tfc.log"/>
      <param name="Append" value="false"/>

      <!-- Rollover at midnight each day -->
```

```
      <param name="DatePattern" value="'.'yyyy-MM-dd"/>

      <layout class="org.apache.log4j.PatternLayout">
         <!-- The default pattern: Date Priority [Category] Message\n -->
         <param name="ConversionPattern" value="%d %-5p [%c] %m%n"/>

      </layout>
   </appender>
      …
   <category name="tfc">
     <priority value="DEBUG" />
     <appender-ref ref="TFC_FILE"/>
   </category>
      …
</log4j:configuration>
```

Des de l'aplicació, s'escriuen alguns missatges de log amb les prioritats INFO i DEBUG. Per a extreure la màxima utilitat d'aquest mecanisme de logs, potser seria interessant afegir alguns missatges més en alguns punts del codi, però per al lliurament tampoc és cap requeriment, i ja han fet la seva funció per ajudar-me a depurar el programa. L'important és, doncs, que el mecanisme està a punt per a ser utilitzat i ampliat si cal.

6. Conclusions

Arribat el moment de treure conclusions, voldria començar valorant el gran esforç de dedicació que ha suposat la realització d'aquest treball de fi de carrera. El ventall de coneixements que ha calgut dominar per tal d'integrar-los en aquest projecte és bastant ampli: HTML, CSS, JavaScript, JSP, EJB, JPA, Struts2, Tiles, JBoss... si bé en algunes àrees ja tenia coneixements previs, en altres ho tenia tot per aprendre, i han fet falta moltes hores d'estudi i pràctica abans d'adquirir el *saber fer* necessari. En aquest sentit, fins al darrer moment he estat aprenent nous detalls. Tampoc ha estat gens menyspreable el temps dedicat a documentar formalment les diferents fases del projecte, un temps que només pot restar-se del disponible per a fer una millor implementació, ja que treballant a jornada completa el meu temps lliure és escàs.

Però crec que l'esforç s'ha vist recompensat amb la satisfacció d'haver pogut complir els objectius establerts a l'inici d'aquest treball , i també amb un aprenentatge bastant sòlid d'una arquitectura i unes eines àmpliament utilitzades en el món laboral avui en dia. Pel tipus d'aplicació que he fet, Struts2 ha resultat ben adequat. Un cop adquirit el *saber fer* necessari, el desenvolupament ha estat fluid i m'ha demostrat que està ben dissenyat per facilitar molts dels aspectes de la capa web d'una manera bastant flexible (pas de dades entre vista i model, i viceversa, validació de dades, gestió de la navegació, interacció amb altres eines –per exemple: Tiles -, internacionalització / localització, interceptors, etc). D'altra banda, en l'àmbit EJB he tingut ocasió de posar en pràctica els Stateless Session Beans, la injecció de dependències per obtenir el context de persistència, i també l'API EJB Timer Services. La meva impressió és que en un projecte més complex, es podria treure molt més profit d'aquesta especificació, ja que disposa de més coses de les que he pogut utilitzar aquí, com és ara els Stateful Session Beans, els Message-Driven Beans i la gestió de la seguretat basada en rols. Finalment, JPA ha resultat molt pràctica per no haver de fer el disseny de la base de dades a mà, i per ocultar la complexitat de les operacions amb la base de dades que hi ha darrere la conversió entre el món orientat a objectes i el món relacional.

Glossari

Articles recents	Llista dels darrers articles pels quals l'usuari ha mostrat interès consultant-ne la pàgina de detalls.
Bean	Terme genèric per designar un objecte o component que conté la informació que ens interessa. Pot fer referència a un EJB, o bé a un objecte Java destinat a contenir dades (en cert sentit, les accions de Struts2 també ho són).
Client	Actor del sistema que és conegut per aquest (és un Usuari Registrat) i que bàsicament només hi interactua per localitzar els articles que li interessen i fer comandes. Ocasionalment, també fa referència al client en un context de client-servidor.
EJB	Acrònim de Enterprise JavaBeans.
Enterprise JavaBeans	Especificació de JavaEE relativa als components de la capa de negoci.
Entity Bean	En l'antiga especificació de EJB, s'incloïa un mecanisme de transformació entre objectes Java i la seva representació persistent en base de dades. Un *entity bean* representava una entitat de la base de dades.
Fitxer de properties	Fitxer de text on cada línia conté un parell clau=valor. La finalitat és recuperar els valors des de l'aplicació utilitzant la clau corresponent. D'aquesta manera és poden alterar paràmetres de l'aplicació sense alterar el codi font ni haver de recompilar.
JavaEE	Java Enterprise Edition: versió de la plataforma Java que inclou les APIs de Servlets, JSPs i EJBs, i d'altres com JMS i JTA.
JavaSE	Java Standard Edition: versió de la plataforma Java que inclou les APIs bàsiques.
JPA	Java Persistence Architecture: API de Java per a la persistència d'objectes en base de dades relacionals.
JSP	Java Server Pages. És una especificació de la capa web de JavaEE. Habitualment serveix per referir-se a fitxers de text que poden contenir codi HTML barrejat amb Java, que són compilats en forma de Servlet pel contenidor. Al no ser, en origen, classes Java, són més adequats que els Servlets per a codificar els components que generen vistes amb abundant codi HTML.

Servlet	Element bàsic de la capa web de JavaEE. Processa peticions (normalment HTTP) i envia al client la resposta adequada.
Stateful EJB	Tipus d'EJB que manté l'estat entre diferents invocacions de les seves operacions. Per aquest motiu, cada bean d'aquest tipus queda associat amb un sol client al qual ofereix els serveis.
Stateless EJB	És un tipus d'Enterprise JavaBeans, un component les operacions del qual no depenen d'un estat associat a una instància concreta del component. Les seves operacions poden ser invocades per qualsevol client que n'obtingui una referència.
Struts2	Framework fet en Java que facilita una sèrie de tasques habituals en la programació de la capa web.
Tag	Element fonamental d'un llenguatge de marcatge.
Tiles	Framework fet en Java que permet donar una estructura uniforme a les diferents pàgines d'una aplicació web gràcies a la utilització de plantilles que defineixen les diferents àrees de la pàgina o es pot posar contingut. Per a cada vista possible, s'especifica quin contingut anirà a cada àrea.
Usuari Registrat	Actor del sistema i a la vegada entitat que representa un usuari conegut per l'aplicació (perquè ha estat donat d'alta a la base de dades). Pot ser un client, un treballador o un administrador.
Visitant	Actor del sistema que no és conegut per l'aplicació i que només pot accedir a la part pública d'aquesta (cerca del catàleg).
Wish-list	Llista a la qual un client afegeix els productes que ha trobat en el catàleg i que li pot interessar comprar en el futur. Així, el client pot revisar la llista en qualsevol moment per recordar-se'n i fer la compra.

Bibliografia

[1] Deepak Alur, John Crupi, Dan Malks. Core J2EE patterns. Best practices and design strategies. 2nd Edition. Prentice-Hall PTR, 2003.
http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html

[2] Rima Patel Sriganesh. Mastering Enterprise JavaBeans 3.0. Wiley Publishing Inc, 2006

[3] Donald Brown, Chad Michael Davis, Scott Stanlick. Struts2 in action. Manning Publications Co., 2008

[4] Simon Rowell. Practical J2EE application architecture. McGrawHill-Osborne, 2003

Annex I. Requeriments de programari i instruccions per a la instal·lació de l'aplicació

L'aplicació requereix el següent programari per a funcionar correctament:

- ✓ Java 6 (probablement també funcionaria amb la versió 5)
- ✓ Servidor d'aplicacions JBoss Application Server 4.2.2 GA (http://www.jboss.org/jbossas/downloads/)
- ✓ MySQL Server 5.0 (http://dev.mysql.com/downloads/mysql/5.0.html)
- ✓ MySQL Connector/J 5.1.6 (http://dev.mysql.com/downloads/connector/j/5.1.html)
- ✓ Struts 2.0.11 (http://struts.apache.org/download.cgi#struts210)
- ✓ Apache Commons Digester 1.8 (http://commons.apache.org/digester/)
- ✓ Apache Commons FileUpload 1.2.1 (http://commons.apache.org/fileupload/)
- ✓ Apache Commons IO 1.4 (http://commons.apache.org/io/)
- ✓ Apache Commons BeanUtils 1.6 (http://commons.apache.org/beanutils/)

Llibreries necessàries només en temps d'execució:

El connector de MySQL no és necessari per a la compilació de l'aplicació, però cal fer-lo accessible al JBoss copiant-lo en el classpath del servidor. Jo he fet servir sempre el servidor 'default', o sigui que he copiat el fitxer .jar a la carpeta <JBOSS_HOME>/server/default/lib. Per a que no sigui necessari descarregar-lo altra vegada, he afegit aquest fitxer a l'arxiu comprimit del lliurament. Les llibreries d'Apache Commons tampoc són necessàries en temps de compilació. En aquest cas, seran accessibles perquè s'inclouen en el directori WEB-INF/lib de l'aplicació web.

Llibreries necessàries en temps de compilació:

Per a compilar, cal tenir en el classpath les llibreries de Servlets, JSP, EJB3, Hibernate i Struts2. Excepte Struts2, totes elles venen amb el servidor JBoss, o sigui que n'hi ha prou amb afegir els següents arxius al classpath:

- ✓ <JBOSS_HOME>/client/servlet-api.jar
- ✓ <JBOSS_HOME>/ client /ejb3-persistence.jar
- ✓ <JBOSS_HOME>/ client /jboss-ejb3x.jar
- ✓ <JBOSS_HOME>/client/jboss-common-client.jar
- ✓ <JBOSS_HOME>/client/jboss-j2ee.jar

Les llibreries necessàries per a Struts2 són les que s'inclouen en el directori WEB-INF/lib de l'aplicació web.

Passos necessaris per a la instal·lació:

L'aplicació es lliura en forma d'arxiu comprimit (jjunyentr_producte.zip), que cal descomprimir en un directori qualsevol i que representarem per <TFC> en endavant.

- ✓ Instal·lar el JBoss (a continuació farem referència al directori d'instal·lació com : <JBOSS_HOME>)
- ✓ Instal·lar el MySQL
- ✓ Descarregar d'Internet el connector MySQL o agafar els fitxer .jar del directori <TFC>, i copiar-lo a <JBOSS_HOME>/server/default/lib.
- ✓ Copiar<TFC> /mysql-ds.xml i <TFC> /TFC.ear al directori <JBOSS_HOME>/server/default/deploy/. El fitxer mysql-ds.xml conté els paràmetres de connexió a la base de dades, que són els habituals per defecte (jdbc:mysql://localhost:3306/tfc, amb usuari root i cap contrasenya).
- ✓ Copiar <TFC>/jboss-log4j.xml al directori <JBOSS_HOME>/server/default/conf. Aquest fitxer conté la configuració de log4j a la qual s'ha afegit les indicacions necessàries per a dirigir els missatges de l'aplicació cap a un fitxer separat de la resta (tfcshop.log)
- ✓ Carregar les dades de prova a la base de dades, p.ex: mysql –user=root – p < <TFC>/tfc.sql. Jo, però, normalment ho faig utilitzant la interfície gràfica del MySQL Query Browser (que es pot descarregar del mateix web que el servidor de base de dades).
- ✓ Arrencar el JBoss (<JBOSS_HOME>\bin\run.bat).

Per començar a veure l'aplicació, cal anar a la pàgina principal de l'aplicació: <servidor>:<port>//tfc/TFCWeb/ (p.ex: http://localhost:8080/TFCWeb/). Des d'aquí es pot navegar pel catàleg i entrar com a client registrat (usuari/contrasenya = client/client), com a treballador (usuari/ contrasenya = worker/worker) o com a administrador (usuari/ contrasenya = admin/admin).

Annex II. Sentències DDL per a crear les taules de la base de dades

```
--
-- Definition of table `adreca`
--

DROP TABLE IF EXISTS `adreca`;
CREATE TABLE `adreca` (
  `codi` int(11) NOT NULL auto_increment,
  `linia1` varchar(255) default NULL,
  `linia2` varchar(255) default NULL,
  `linia3` varchar(255) default NULL,
  `codiPostal` varchar(255) default NULL,
  `pais_codi` int(11) NOT NULL,
  PRIMARY KEY  (`codi`),
  KEY `FK74A188944C8C4152` (`pais_codi`),
  CONSTRAINT `FK74A188944C8C4152` FOREIGN KEY (`pais_codi`) REFERENCES `pais`
(`codi`)
) ENGINE=InnoDB AUTO_INCREMENT=15 DEFAULT CHARSET=latin1;


--
-- Definition of table `article`
--

DROP TABLE IF EXISTS `article`;
CREATE TABLE `article` (
  `codi` int(11) NOT NULL auto_increment,
  `preu` float NOT NULL,
  `disponible` bit(1) NOT NULL,
  `nom` varchar(255) default NULL,
  `descripcioBreu` varchar(255) default NULL,
  `descripcio` mediumtext,
  `valoracio` float NOT NULL,
  `categoria_codi` int(11) default NULL,
  PRIMARY KEY  (`codi`),
  KEY `FK379164D6538B9A4E` (`categoria_codi`),
  CONSTRAINT `FK379164D6538B9A4E` FOREIGN KEY (`categoria_codi`) REFERENCES
`categoria` (`codi`)
) ENGINE=InnoDB AUTO_INCREMENT=90 DEFAULT CHARSET=latin1;


--
-- Definition of table `articlevisitat`
--

DROP TABLE IF EXISTS `articlevisitat`;
CREATE TABLE `articlevisitat` (
  `codi` int(11) NOT NULL auto_increment,
  `ultimaVisita` datetime default NULL,
  `numVisites` int(11) NOT NULL,
  `usuari_codi` int(11) NOT NULL,
  `article_codi` int(11) NOT NULL,
  PRIMARY KEY  (`codi`),
  KEY `FK9E1C69088849CBAE` (`article_codi`),
  KEY `FK9E1C6908D0F957BB` (`usuari_codi`),
  CONSTRAINT `FK9E1C69088849CBAE` FOREIGN KEY (`article_codi`) REFERENCES
`article` (`codi`),
```

```
  CONSTRAINT `FK9E1C6908D0F957BB` FOREIGN KEY (`usuari_codi`) REFERENCES
`persona` (`codi`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


--
-- Definition of table `articlewishlist`
--

DROP TABLE IF EXISTS `articlewishlist`;
CREATE TABLE `articlewishlist` (
  `codi` int(11) NOT NULL auto_increment,
  `dataHoraInsercio` datetime default NULL,
  `usuari_codi` int(11) NOT NULL,
  `article_codi` int(11) NOT NULL,
  PRIMARY KEY  (`codi`),
  KEY `FK8D31BCDB8849CBAE` (`article_codi`),
  KEY `FK8D31BCDBD0F957BB` (`usuari_codi`),
  CONSTRAINT `FK8D31BCDB8849CBAE` FOREIGN KEY (`article_codi`) REFERENCES
`article` (`codi`),
  CONSTRAINT `FK8D31BCDBD0F957BB` FOREIGN KEY (`usuari_codi`) REFERENCES
`persona` (`codi`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=latin1;


--
-- Definition of table `categoria`
--

DROP TABLE IF EXISTS `categoria`;
CREATE TABLE `categoria` (
  `codi` int(11) NOT NULL auto_increment,
  `nom` varchar(255) default NULL,
  `pare_codi` int(11) default NULL,
  PRIMARY KEY  (`codi`),
  KEY `FKD4C70113A8171E3D` (`pare_codi`),
  CONSTRAINT `FKD4C70113A8171E3D` FOREIGN KEY (`pare_codi`) REFERENCES
`categoria` (`codi`)
) ENGINE=InnoDB AUTO_INCREMENT=603 DEFAULT CHARSET=latin1;

--
-- Definition of table `comanda`
--

DROP TABLE IF EXISTS `comanda`;
CREATE TABLE `comanda` (
  `codi` int(11) NOT NULL auto_increment,
  `adrecaEnviament` varchar(255) default NULL,
  `formaEnviament` int(11) default NULL,
  `estat` int(11) default NULL,
  `dataHora` datetime default NULL,
  `dataHoraEnviament` datetime default NULL,
  `client_codi` int(11) NOT NULL,
  `pagament_codi` int(11) default NULL,
  PRIMARY KEY  (`codi`),
  KEY `FK9BD9324B35692912` (`pagament_codi`),
  KEY `FK9BD9324B965E8F91` (`client_codi`),
  CONSTRAINT `FK9BD9324B35692912` FOREIGN KEY (`pagament_codi`) REFERENCES
`pagament` (`codi`),
  CONSTRAINT `FK9BD9324B965E8F91` FOREIGN KEY (`client_codi`) REFERENCES
`persona` (`codi`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=latin1;
```

```
--
-- Definition of table `liniacomanda`
--

DROP TABLE IF EXISTS `liniacomanda`;
CREATE TABLE `liniacomanda` (
  `codi` int(11) NOT NULL auto_increment,
  `quantitat` int(11) NOT NULL,
  `preuUnitari` float NOT NULL,
  `article_codi` int(11) NOT NULL,
  `comanda_codi` int(11) NOT NULL,
  PRIMARY KEY  (`codi`),
  KEY `FK1A55BEE2B5C8ECCE` (`comanda_codi`),
  KEY `FK1A55BEE28849CBAE` (`article_codi`),
  CONSTRAINT `FK1A55BEE28849CBAE` FOREIGN KEY (`article_codi`) REFERENCES
`article` (`codi`),
  CONSTRAINT `FK1A55BEE2B5C8ECCE` FOREIGN KEY (`comanda_codi`) REFERENCES
`comanda` (`codi`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=latin1;


--
-- Definition of table `pagament`
--

DROP TABLE IF EXISTS `pagament`;
CREATE TABLE `pagament` (
  `codi` int(11) NOT NULL auto_increment,
  `dataHora` datetime default NULL,
  `codiTargeta` varchar(255) default NULL,
  `validesaTargeta` varchar(255) default NULL,
  PRIMARY KEY  (`codi`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=latin1;


--
-- Definition of table `pais`
--

DROP TABLE IF EXISTS `pais`;
CREATE TABLE `pais` (
  `codi` int(11) NOT NULL auto_increment,
  `nom` varchar(255) default NULL,
  PRIMARY KEY  (`codi`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=latin1;


--
-- Definition of table `persona`
--

DROP TABLE IF EXISTS `persona`;
CREATE TABLE `persona` (
  `tipusPersona` varchar(31) NOT NULL,
  `codi` int(11) NOT NULL auto_increment,
  `nom` varchar(255) default NULL,
  `cognoms` varchar(255) default NULL,
  `dataNaixement` datetime default NULL,
  `email` varchar(255) default NULL,
  `nomUsuari` varchar(255) default NULL,
  `contrasenya` varchar(255) default NULL,
```

```
  `actiu` bit(1) default NULL,
  `tipusUsuari` int(11) default NULL,
  PRIMARY KEY  (`codi`)
) ENGINE=InnoDB AUTO_INCREMENT=20 DEFAULT CHARSET=latin1;


--
-- Definition of table `persona_adreca`
--

DROP TABLE IF EXISTS `persona_adreca`;
CREATE TABLE `persona_adreca` (
  `Persona_codi` int(11) NOT NULL,
  `adreces_codi` int(11) NOT NULL,
  PRIMARY KEY  (`Persona_codi`,`adreces_codi`),
  UNIQUE KEY `adreces_codi` (`adreces_codi`),
  KEY `FK127BC8A792ED0D6E` (`Persona_codi`),
  KEY `FK127BC8A7F23B1E4B` (`adreces_codi`),
  CONSTRAINT `FK127BC8A792ED0D6E` FOREIGN KEY (`Persona_codi`) REFERENCES
`persona` (`codi`),
  CONSTRAINT `FK127BC8A7F23B1E4B` FOREIGN KEY (`adreces_codi`) REFERENCES
`adreca` (`codi`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


--
-- Definition of table `valoracioarticle`
--

DROP TABLE IF EXISTS `valoracioarticle`;
CREATE TABLE `valoracioarticle` (
  `codi` int(11) NOT NULL auto_increment,
  `puntuacio` int(11) NOT NULL,
  `comentaris` mediumtext,
  `dataHora` datetime default NULL,
  `article_codi` int(11) NOT NULL,
  `usuari_codi` int(11) NOT NULL,
  PRIMARY KEY  (`codi`),
  KEY `FKE9F42BEA8849CBAE` (`article_codi`),
  KEY `FKE9F42BEAD0F957BB` (`usuari_codi`),
  CONSTRAINT `FKE9F42BEAD0F957BB` FOREIGN KEY (`usuari_codi`) REFERENCES
`persona` (`codi`),
  CONSTRAINT `FKE9F42BEA8849CBAE` FOREIGN KEY (`article_codi`) REFERENCES
`article` (`codi`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=latin1;
```

Annex III. Mostra de pantalles del producte final

Figura 23. Pàgina principal – navegació del catàleg (versió final)

Figura 24. Pàgina de detalls de l'article (versió final)

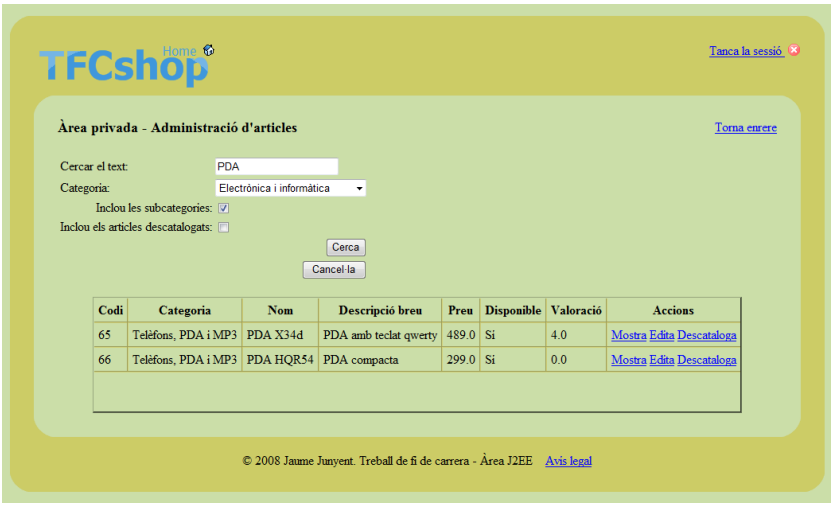


Figura 25. Administració d'articles existents – cerca (versió final)

Figura 26. Administració d'articles existents – edició (versió final)

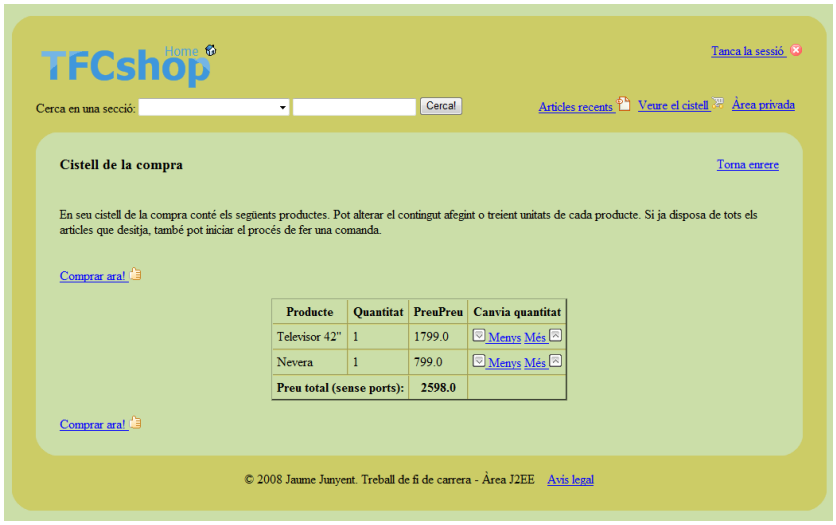


Figura 27. Cistell de la compra (versió final)

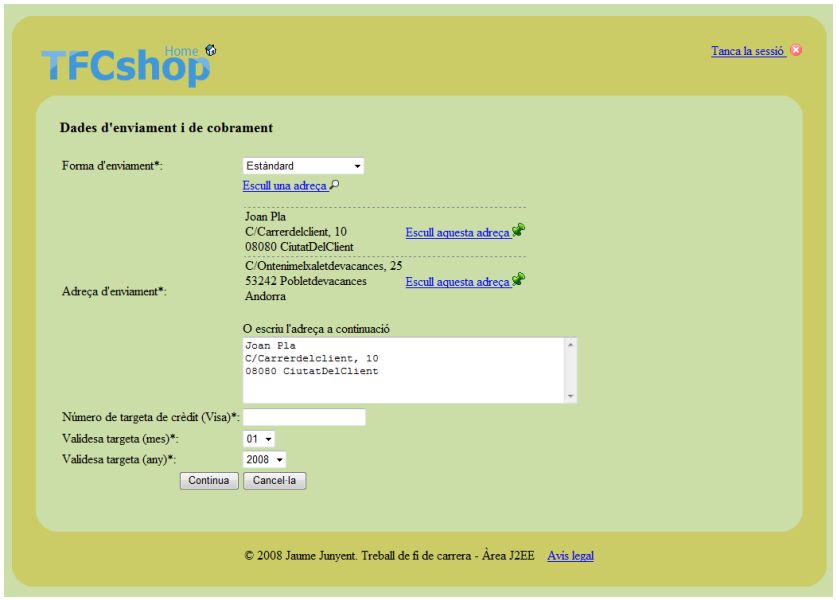


Figura 28. Formulari de comanda (versió final)

Figura 29. Pàgina de confirmació definitiva de comanda (versió final)