Estudis d'Informàtica i Multimèdia

Disseny i implementació d'un framework de persistència

PROJECTE FINAL DE CARRERA

MEMÒRIA

Cognoms: *Mena Fernàndez*
Nom: *Carlos*

Juny 2013

UOC
www.uoc.edu

LLICÈNCIA CREATIVE COMMONS

Aquest treball està subjecte - excepte que s'indiqui el contrari- en una llicència de Reconeixement-NoComercial-SenseObraDerivada 2.5 Espanya de Creative Commons. Podeu copiar-lo, distribuir-los i transmetre'ls públicament sempre que citeu l'autor i l'obra, no es faci un ús comercial i no es faci còpia derivada. La llicència completa es pot consultar en http://creativecommons.org/licenses/by-nc-nd/2.5/es/deed.es.

ÍNDEX

1. INTRODUCCIÓ....5
2. DESCRIPCIÓ DEL PFC....6
2.1. Propòsit....6
2.2. Objectius generals i específics....7
2.3. Pla de treball....7
2.3.1. Calendari de desenvolupament....7
2.3.2. Planificació temporal....8

3. SISTEMES GESTORS DE BASES DE DADES....9
3.1. Primera generació....9
3.1.1. Avantatges....9
3.1.2. Inconvenients....9
3.2. Segona generació....10
3.2.1. Avantatges....10
3.2.2. Inconvenients....10
3.2.3. Comparativa....10
3.3. Tercera generació....11
3.3.1. OODBMS: Bases de dades orientades a objectes....11
3.3.1.1 Avantatges....11
3.3.2. ORDBMS: Bases de dades objecte-relacional....11
3.3.2.1 Avantatges....12
3.3.2.2 Comparativa....12
3.3.3. NoSQL....12
3.3.3.1 Avantatges....13
3.3.3.2 Inconvenients....13

4. PERSISTÈNCIA DE DADES....13
4.1. Objectius dels SGBD's....13
4.2. Mecanismes de persistència....14
4.3. Mapping d'objectes relacional (ORM)....15
4.3.1 Patró CRUD....15
4.3.2 Càrrega de les relacions....15
4.3.3 Concurrència....16
4.3.4. Memòria cache....16
4.3.4.1 Coherència....16
4.3.4.2 Implementacions....17
4.3.4.2.1 EhCache....18

5. FRAMEWORKS DE PERSISTÈNCIA DE DADES....19
5.1. Requisits i característiques principals....20
5.2. JDBC....21
5.3. Frameworks....21
5.3.1. Java Persistence API (JPA)....22
5.3.2. Top Link Essentials....22
5.3.3. Hibernate....23
5.3.4. Cayenne....23
5.3.5. myBatis....24
5.4. Comparatives....24
5.4.1. Hibernate vs. TopLink....24
5.4.2. myBatis vs. ORM....24

6. DISSENY FRAMEWORK....25
6.1. Introducció....25
6.2. Diagrama de casos d'ús....25
6.2.1. config....27
6.2.2. core....28
6.2.3. util....30
6.3. Diagrama de classes....30
6.4. Diagrama de components....31
6.4.1. core....31
6.4.2. util....32

7. DISSENY DE L'APLICACIÓ D'EXEMPLE D'ÚS DEL FRAMEWORK....33
7.1. Introducció....34
7.2. Diagrama de casos d'ús....35
7.2.1. Autenticació 35
7.2.2. Administració 36
7.2.3. Ofertes 36
7.3. Diagrama de classes....37
7.4. Diagrama de components....38
7.4.1. Vista 39
7.4.2. Controlador 41
7.4.3. Model 42

8. CONCLUSIONS....43

9. BIBLIOGRAFIA....44
9.1. Enllaços i referències a Internet....44

A. FRAMEWORK DE BASE DE DADES – INSTAL·LACIÓ I DISTRIBUCIÓ....46
Prerequisits software....46
Estructura del projecte....46
Configuració de l'entorn de desenvolupament....46
Estructura de les fonts....47
Compilació de l'aplicació....47
Recursos de configuració de l'aplicació....47
uoc-bd-config.properties....47
log4j.xml....48

B. APLICACIÓ WEB – INSTAL·LACIÓ....48
Prerequisits software....48
Estructura de les fonts....49
Compilació de l'aplicació....49
Recursos de configuració de l'aplicació....49
Generació dels beans d'accés a la base de dades....49

C. APLICACIÓ WEB – MANUAL USUARI....50
Pàgina d'inici....50
Login com a administrador del portal....50
Login....50
Error - Login sense dades....50
Error – Usuari desconegut....51
Error - Login amb paraula de pas errònia....51
Llistat d'ofertes 51
Vista general 51
Vista administrador 52
Llistat d'ofertes filtrades....52
Visualitzar una oferta 53
Reserva d'una oferta 53
Alta d'una oferta 55
Modificació d'una oferta 56
Eliminació d'una oferta 57

D. LLIBRERIES UTILITZADES....58

1. INTRODUCCIÓ

El present projecte final de carrera (PFC, en endavant) - *Disseny i implementació d'un marc de treball de persistència* - requereix l'anàlisi i desenvolupament d'un conjunt de components que simplifiquin i agilitzin el desenvolupament de la capa de persistència en aplicacions multicapa en entorn JEE. La memòria del qual estarà formada per tres grans blocs: especificació del PFC, anàlisi de les tecnologies i disseny i implementació de la solució.

El primer dels bloc, especificació del PFC -que constituirà el segon capítol de la memòria-, està enfocat a la descripció del PFC i que inclourà el propòsit, els objectius generals i específics del projecte i el pla de treball on s'especifica el calendari i la planificació temporal.

En el bloc de l'anàlisi de les diferents tecnologies es realitzarà la definició del marc de treball que s'utilitzarà com a base del desenvolupament de la solució i està organitzat en tres capítols diferents: sistemes gestors de bases de dades, persistència i *frameworks* de persistència, donat que es considera la millor manera de relacionar els conceptes dels PFC. S'inicia amb la capa de dades i la seva evolució; seguidament s'incorpora el concepte de persistència amb els diferents mecanismes i amb una explicació detallada del model ORM; i, finalment, es realitza la comparativa dels diferents *frameworks* de persistència de dades del moment.

Així doncs, el tercer capítol de la memòria està centrat en els sistemes gestors de bases de dades (SGBDs) i s'ha dividit en tres "generacions". Inicialment es mostren els primers sistemes - anys 60s amb aplicacions totalment per lots : sistemes de fitxers - amb les avantatges que van introduir en el seu moment. La segona "generació" - anys 70s – tracta l'evolució dels sistemes de fitxers cap a un model de base de dades (primers SGBDs) en el que es prioritzava la coherència de les dades i la no repetició de les mateixes. El capítol es conclou amb el que s'ha anomenat tercera generació i en el que es parla de l'evolució dels SGBDs per a adaptar-se a tres tecnologies d'èxit recent fortament interrelacionades: la multimèdia, l'orientació a l'objecte (OO), i Internet i el web. Així doncs, en aquesta "generació" es parla d' SGBDs orientats a objectes, d'SGBD objecte-relacio i de SGBD's NoSQL.

El quart capítol s'ha dedicat a la persistència i s'inicia amb una definició del concepte de persistència i amb els objectius que "persegueixen" els SGBD's. A continuació, s'expliquen els diferents mecanismes de persistència de dades. En aquest capítol, es fa un ampli anàlisi dels models relacionals – *ORM* - donat que seran la base a partir de la qual es construirà el framework a desenvolupar en el PFC. En aquest sentit, es parla de les diferents característiques que cal gestionar en aquest tipus de sistemes: patró CRUD, càrrega de les relacions, concurrència i memòria caché. En l'apartat de memòria caché es tracten temes fonamentals com per exemple, la coherència de les dades i diferents implementacions disponibles, detallant l'EhCache.

El darrer capítol del bloc, i el cinquè de la memòria, està enfocat als frameworks i, en particular, als frameworks de persistència. Així doncs, s'inici amb una descripció dels requisits i característiques generals dels frameworks. En el següent apartat, s'explica l'API JDBC, donat que es tracta de la base amb la que treballen els frameworks de persistència disponibles avui en dia. A continuació, es realitza l'anàlisi de diferents frameworks: *TopLink*, *Hibernate*, *Apache Cayenne* i *myBatis*; tot i que abans es dedica un apartat a JPA donat que es tracta de l'estàndard de persistència de la plataforma Java. El capítol es finalitza amb un anàlisi comparatiu de les diferents solucions estudiades.

El bloc de la definició del disseny i arquitectura del *framework* de persistència i de l'aplicació web estarà formada per dos capítols més i que es corresponen al sisè i setè capítols de la memòria, respectivament. En aquest sentit, l'estructura de cadascun d'aquests capítols serà la mateixa: un primer apartat d'introducció en el que es comentaran els diferents patrons i principis utilitzats en el disseny, especificant els motius pels quals s'utilitzen i les avantatges que incorporen. A continuació, es mostrarà l'anàlisi funcional mitjançant un diagrama de casos d'ús en el que es detallaran cadascuna de les decisions preses. Un cop es disposa de l'anàlisi funcional, es realitzarà el el punt de vista de la informació mitjançant un diagrama de classes en el que es farà una breu descripció de les entitats i les relacions entre elles. Per a concloure el capítol, es mostrarà el diagrama de components refinat amb el disseny de la implementació.

Finalment, la memòria inclourà les conclusions i bibliografia, un així com els annexos necessaris per a la posada en marxa del *framework*, de l'aplicació web (manuals de desenvolupament i d'instal·lació) i el manual d'usuari de la web desenvolupada. Aquest darrer manual detalla la navegació de les vistes disponibles a l'aplicació web i l'explicació de la funcionalitat de cadascuna de les pantalles desenvolupades.

2. DESCRIPCIÓ DEL PFC

2.1. Propòsit

L'objectiu principal del projecte correspon a dissenyar un *framework* de persistència per a poder abstraure als programadors de la complexitat de la capa de dades.

El treball disposarà de dues parts clarament diferenciades. En primer lloc, es realitzarà una part teòrica que inclourà un estudi sobre les diferents implementacions de la persistència disponibles per a la plataforma Java -capítols 3, 4 i 5 de la memòria-. Serà en aquesta part del projecte, en la que s'estudiarà l'arquitectura de diferents *frameworks*, entre els que es trobaran *JPA*, *Hibernate*, *iBatis o* TopLink així com les característiques que incorporen cadascun d'ells. Aquesta part esdevindrà fonamental, donat que el *framework* que es desenvoluparà haurà d'incorporar les característiques bàsiques i comuns a tots.

En aquest sentit, es considerarà necessari que el *framework* incorpori una funcionalitat per a poder generar els objectes a partir de la cadena de connexió a la base de dades. D'aquesta manera, no serà necessari que el programador realitzi els "*mappings*" entre els objectes Java (POJO's[1]) i la base de dades corresponent. D'altra banda, cal destacar que s'utilitzarà JPA com a referència en el desenvolupament del *framework*, donat que es troba inclosa en l'estàndard EJB 3.0. Aquesta API disposa de diferents implementacions entre les que es troba *Hibernate* o *TopLink* .

Un dels punts febles de la persistència de dades correspon al rendiment, en aquest sentit, es realitzarà un estudi de les alternatives disponibles per a la millora del rendiment. Una pràctica molt bona correspon a desenvolupar una capa intermèdia a memòria - entre el controlador i el model – per a poder reduir el temps de resposta de les operacions i posteriorment, fer el bolcat al SGBC. Així doncs, es farà especial atenció a les caches distribuïdes - que abstrauran de la problemàtica de la replicació de la informació en els diferents nodes d'un clúster- per aconseguir millorar el cost d'aquest tipus d'operacions que penalitzen tant el rendiment. D'altra banda, i tenint en compte el reso que estan obtenint en els darrers temps, s'estudiaran les avantatges que aporten les bases de dades *NoSQL* a les aplicacions, en casos de volums elevats de dades així com els principals inconvenients i dificultats que incorporen en la persistència de dades.

En segon lloc, i un cop estudiades i comparades les característiques de les solucions disponibles al mercat, es realitzarà la part més pràctica del projecte, que correspondrà al disseny i la implementació del *framework* de persistència. La finalitat del *framework* no serà "competir" amb la resta de *frameworks* estudiats, perquè, no es disposa ni del temps ni dels recursos que s'utilitzen en els desenvolupaments d'aquests "*macro-frameworks*"; sinó, que es tractarà de crear un ***framework lleuger*** – qualsevol dels *frameworks* estudiats incorpora una sèrie de llibreries addicionals que augmenta considerablement el "pes" final de l'aplicació – ***i senzill*** - la majoria de *frameworks* estudiats requereix una corba d'aprenentatge molt important, en especial, *Hibernate* donat que disposa d'un llenguatge propi (HSQL) – que permeti al programador "oblidar-se" de la problemàtica que incorpora la capa d'accés a les dades.

El *framework* desenvolupat i l'aplicació web d'exemple hauran de ser extensibles i escalables. És per això, que s'utilitzarà la orientació a objectes que incorpora Java, els patrons de disseny estudiats a les assignatures d'enginyeria del software, així com els principis SOLID de la programació orientada a objecte. En aquest sentit, i tenint en compte el poc temps del que es disposa per a realitzar el desenvolupament, haurà de ser fàcilment mantenible, donat que en aquesta versió només es podran

[1]Acrònim de **Plain Old Java Object**, que es refereix als objectes Java que no es veuen carregats per especificacions que incrementen la seva estructura d'atributs o mètodes.

realitzar les operacions bàsiques. D'aquesta manera, s'obtindria una implementació "oberta" que permetrà d'una manera senzilla incorporar funcionalitats més avançades amb la intenció de millorar i aconseguir un *framework* més robust i complert.

2.2. Objectius generals i específics

Un dels motius pels qual he seleccionat aquest projecte ha estat la importància que suposa la persistència de dades en la implementació de qualsevol aplicació.

Així doncs, els objectius generals correspondran a conèixer l'arquitectura de la capa de persistència de JAVA, fent un anàlisi de les diferents solucions disponibles: començant per JDBC (Java DataBase Connectivity), fins arribar al darrer estàndard, JPA (*Java Persistence API*) - que serà, tal i com s'ha comentat en l'apartat anterior, l'estàndard sobre la que es basarà el *framework* a desenvolupar -.

En aquest sentit, es pretén estudiar les parts més significatives de la JPA: el Llenguatge de Query de Persistència Java (Java Query Persistence Language, JQPL); la relació entre Java Persistence API i Enterprise JavaBeans i, finalment, la relació entre Java Persistence API i l'API Service Data Object (SDO).

D'altra banda, s'analitzaran les diferents implementacions de *frameworks* open-source existents al mercat (Hibernate, iBatis,...) així com algunes de les implementacions de les bases de dades NoSQL per aconseguir un coneixement més profund de les "noves" bases de dades (Cassandra, MongoDB, BigTable, ...).

Pel que fa a les millores en el rendiment de la solució, es valorà la incorporació de les *caches* de memòria amb les avantatges i inconvenients que produeixen. En aquest sentit, es realitzarà un estudi d'una de les *caches* distribuïda més utilitzada, *EHCache*.

Indubtablement, el fet d'haver de dissenyar un *framework* de persistència, em servirà per a reforçar tots els patrons de disseny estudiats a les assignatures d'enginyeria del software (Enginyeria del Programari Orientada a Objectes i Procés d'Enginyeria del Programari) i en els principis SOLID de la programació orientada a objectes.

D'altra banda, aquest PFC provocarà una millora en el disseny i la implementació d'una solució de programari per a la versió de JAVA utilitzada en el desenvolupament (1.5 o superior) – i per tant, permetrà fer un ús de les principals novetats que incorpora respecte a la versió predecessora: anotacions i genèrics, entre d'altres - i del *framework Spring*, que s'utilitzarà com a *framework* d'inversió de control per a poder gestionar les necessitats d'inversió de control de manera senzilla (sense haver de recórrer a l'ús de patrons de disseny clàssics: *Factory* i *AbstractFactory*).

Finalment, amb el desenvolupament de l'aplicació es pretén veure un cas pràctic del *framework* desenvolupat i poder comprovar la utilitat d'aquesta eina.

2.3. Pla de treball

2.3.1. Calendari de desenvolupament

De cara a la realització del pla de treball i la planificació s'han pres com a base les dates proposades per la coordinació de l'assignatura i que són les següents:

Fites de control		
Data	**Fita de control**	**Observacions**
11/03/13	PAC1	Realització del pla de treball.
15/04/13	PAC2	Entrega de la part més "teòrica" del projecte: comparativa dels diferents frameworks de persistència. També s'inclourà una valoració dels mecanismes de millora del rendiment (caches) i un estudi sobre les bases de dades NoSQL.
03/06/13	PAC3	Entrega del disseny del *framework* desenvolupat així com la implementació d'alguna de les funcionalitats més bàsiques.
17/06/13	Memòria i presentació	Entrega de la memòria, la presentació i el producte (codi font).

La data inicial del projecte s'ha fixat en el dia de la sessió no presencial d'inici de curs, el 05/03/2013, i s'hauria de tenir el producte finalitzat - amb la memòria, fonts i presentació- el 17/06/2013. Cal destacar també, que durant el període lectiu existiran els següents dies festius:

- 29 de març: Divendres sant
- 1 d'abril: Dilluns de pasqua
- 1 de maig: Dia del treballador
- 20 maig: Segona pasqua

Així doncs, es disposen de 75 dies laborables per a poder realitzar el projecte, donat que els dies festius especificats s'utilitzaran com a reforç dels dies laborables. Val a dir, que la planificació només compta dies laborables, i és per això, que es disposarà de suficients dies de marge – els caps de setmana – per a poder pal·liar els riscos que es puguin produir durant el desenvolupament del projecte - dificultats tècniques, imprevistos, ... -.

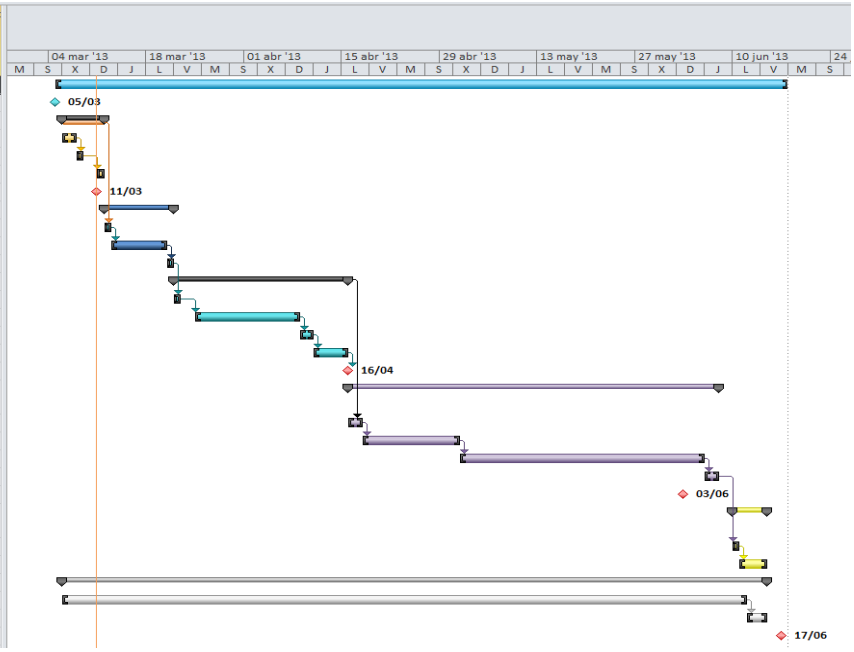
2.3.2. Planificació temporal

Per a la planificació temporal del projecte es mostrarà la precedència de les activitats que el formen:

Codi de l'activitat	Activitat	Descripció de l'activitat	Estimació (en dies)	Precedències
1	**Sessió inici**	Sessió no presencial inici curs	0	
2	**Pla treball**		4	
2.1	Introducció i propòsits	Definició del marc en el que es desenvolupa el framework i propòsits	2	1
2.2	Objectius	Definició dels objectius generals i específics	1	2.1
2.3	Planificació	Calendari del desenvolupament del projecte i planificació temporal	1	
3	**Persistència de dades**		8	
3.1	Introducció persistència	Evolució de la persistència de les dades durant la història	1	2
3.2	Anàlisi dels SGBD	Estudi dels diferents SGBD disponibles: models relacionals, models noSQL, i altres.	6	3.1
3.3	Comparativa dels models	Comparativa amb les avantatges i inconvenients de cadascun dels models.	1	3.2
4	**Frameworks**		17	
4.1	Introducció	Definició de frameworks i exemples d'us	1	3.3
4.2	Anàlisi de frameworks	Estudi dels frameworks actuals: JPA, Hibernate, TopLink, ...	11	4.1
4.3	Comparativa	Comparativa amb les avantatges i inconvenients de cadascun dels framworks.	2	4.2
4.4	Millores	Estudi de les millores de rendiment que es poden incorporar	3	4.3
5	**Disseny i implementació del framework**		39	
5.1	Anàlisi funcional	Anàlisi funcional amb els objectius i requeriments que ha d'implementar el framework	2	4
5.2	Disseny de l'arquitectura	Disseny de l'arquitectura del framework a desenvolupar	10	5.1
5.3	Implementació i proves unitàries	Implementació de la solució amb proves unitàries per a la validació del funcionament.	25	5.2
5.4	Proves funcionals	Validació funcional del framework desenvolupat	2	5.3
6	**Disseny i implementació d'una aplicació d'exemple de l'ús del framework**		5	
6.1	Objectius i requeriments	Objectius i requeriments de l'aplicació per a mostrar el correcte funcionament del framework de persistència	1	5
6.2	Implementació i proves	Implementació de l'aplicació web amb proves unitàries per a la validació del funcionament tant de la web com del framework.	4	6.1
7	**Documentació**		73	
7.1	Memòria	Memòria del projecte	70	
7.2	Presentació	Presentació del projecte	3	7.1

A continuació, es mostra la planificació proposada, tenint en compte que la data d'inici del projecte correspondrà al 5 de març de 2013:

	Nombre de tarea	Duración	Comienzo	Fin	Predec
1	PFC	75 días	mar 05/03/13	lun 17/06/13	
2	Sessió d'inici	0 días	mar 05/03/13	mar 05/03/13	
3	**Pla de treball**	**4 días**	**mié 06/03/13**	**lun 11/03/13**	
4	Introducció i propòsits	2 días	mié 06/03/13	jue 07/03/13	
5	Objectius	1 día	vie 08/03/13	vie 08/03/13	4
6	Planificació	1 día	lun 11/03/13	lun 11/03/13	5
7	PAC1	0 días	lun 11/03/13	lun 11/03/13	
8	**Persistència de dades**	**8 días**	**mar 12/03/13**	**jue 21/03/13**	
9	Introducció	1 día	mar 12/03/13	mar 12/03/13	3
10	Anàlisis dels SGBD	6 días	mié 13/03/13	mié 20/03/13	9
11	Comparativa	1 día	jue 21/03/13	jue 21/03/13	10
12	**Frameworks**	**17 días**	**vie 22/03/13**	**mar 16/04/13**	
13	Introducció	1 día	vie 22/03/13	vie 22/03/13	11
14	Anàlisi	11 días	lun 25/03/13	lun 08/04/13	13
15	Comparativa	2 días	mar 09/04/13	mié 10/04/13	14
16	Millores	3 días	jue 11/04/13	lun 15/04/13	15
17	PAC2	0 días	mar 16/04/13	mar 16/04/13	16
18	**Disseny i implementació del framework**	**39 días**	**mar 16/04/13**	**vie 07/06/13**	
19	Anàlisi funcional	2 días	mar 16/04/13	mié 17/04/13	12
20	Disseny de l'arquitectura	10 días	jue 18/04/13	mié 01/05/13	19
21	Implementació i proves unitàries	25 días	jue 02/05/13	mié 05/06/13	20
22	Proves funcionals	2 días	jue 06/06/13	vie 07/06/13	21
23	PAC3	0 días	lun 03/06/13	lun 03/06/13	
24	**Disseny i implementació d'una aplicació d'exemple**	**5 días?**	**lun 10/06/13**	**vie 14/06/13**	
25	Objectius i requeriments	1 día?	lun 10/06/13	lun 10/06/13	22
26	Implementació i proves unitàries	4 días	mar 11/06/13	vie 14/06/13	25
27	**Documentació**	**73 días**	**mié 06/03/13**	**vie 14/06/13**	
28	Memòria	70 días	mié 06/03/13	mar 11/06/13	
29	Presentació	3 días	mié 12/06/13	vie 14/06/13	28
30	Entrega final	0 días	lun 17/06/13	lun 17/06/13	



3. SISTEMES GESTORS DE BASES DE DADES

En aquest capítol es realitza un estudi de l'evolució dels sistemes gestors de bases de dades -SGBD en endavant -, incorporant l'anàlisi de les característiques dels diferents models. Per realitzar aquest anàlisis s'han diferenciat tres "etapes" o "generacions" de SGBD's: sistemes de fitxers; primers sistemes gestors de bases de dades i model relacional; i, finalment, les tendències actuals en els SGBD's.

Cal remarcar que per motius de temps, el *framework* desenvolupat només donarà suport als SGBD's relacionals, malgrat tot, es considera important fer un estudi de l'evolució dels SGBD's i tenir en compte les línies futures de cara a possibles ampliacions que es pugin donar en el framework.

3.1. Primera generació

Les aplicacions informàtiques dels anys seixanta acostumaven a ser totalment per lots – *batch* - i estaven pensades per a una feina molt específica i relacionada amb molt poques entitats tipus. Així doncs, cada programa tractava de manera seqüencial amb un fitxer mestre i cada vegada que s'hi volia afegir una aplicació que utilitzés alguna de les dades del fitxer, es creava un nou amb tota la informació i, per tant, s'introduïa redundància de la informació (diferents fitxers amb la mateixa informació).

3.1.1. Avantatges

Les avantatges més rellevants que van incorporar aquest tipus de sistema van ser:

- Continguts dinàmics
- Facilitat d'implantació

3.1.2. Inconvenients

A continuació es detallen els principals inconvenients que incorporen aquest tipus de sistema:

- Redundància
- Inconsistència
- Limitacions d'accés: només permeten un únic usuari / aplicació.
- Concurrència

3.2. Segona generació

Per a poder pal·liar els inconvenients de la primera generació es modifica la orientació dels sistemes per passar de la orientació a processos a la orientació a dades: les dades s'organitzen i es mantenen en un conjunt estructurat que satisfà les necessitats d'informació de tota la organització i no les d'una aplicació en concret.

Així doncs, es pot definir una base de dades (BD) com un conjunt estructurat de dades únic (la representació serà única i integrada) que permet utilitzacions diverses i simultànies, i que representa entitats i les seves interrelacions. El programari de gestió de fitxers de la primera generació era massa elemental per a donar satisfacció a totes aquestes noves necessitats, fet que va provocar que durant la segona meitat dels anys setanta, apareguessin programaris més sofisticats: el que es va anomenar sistemes de gestió de bases de dades (SGBD, RDBMS en anglès).

Els primers SGBD estaven orientats a facilitar la utilització de grans conjunts de dades interrelacionats de manera complexa. Els programes, com que estaven lligats al nivell físic, s'havien de modificar contínuament quan es feien canvis en el disseny i en l'organització de la BD donat que en aquell moment la preocupació bàsica era maximitzar el rendiment: el temps de resposta i les transaccions per segon.

Durant la dècada dels 80, els ordinadors minis, en primer lloc i, després els ordinadors micros, van estendre la informàtica a pràcticament totes les empreses i institucions; fet que va exigir que el desenvolupament d'aplicacions fos més senzill. L'aparició dels SGBD relacionals va suposar un avenç important per a facilitar la programació d'aplicacions amb bases de dades i per aconseguir que els programes fossin independents dels aspectes físics de la base de dades.

Una característica fonamental d'aquests sistemes SGBD, està en la incorporació d'un llenguatge estàndard – *SQL, Structured Query Language* - implementat pels principals motors o sistemes de gestió de bases de dades relacionals per a poder construir les consultes.

3.2.1. Avantatges

Les principals avantatges que incorporen aquest tipus de sistema - en relació als de primera generació- són les següents:

- Coherència: no existeixen dades redundants.
- Disponibilitat
- Concurrència
- Integritat
- Seguretat
- Flexibilitat
- Reducció de l'espai d'emmagatzemament
- Millora de la eficiència i l'accés a les dades

3.2.2. Inconvenients

En relació als inconvenients, disposen dels següents:

- Instal·lació costosa
- Implantació complexa

3.2.3. Comparativa

A continuació es mostrarà una taula amb una comparativa entre els sistemes de primera i segona generació:

Comparativa amb la primera generació		
Característica	Sistema de fitxers	Base de dades - SGBD
Entitats tipus	1	Varies
Interrelacions	No	Si
Redundància	Si. Cada aplicació disposa d'un fitxer propi i en cas de varies aplicacions amb les mateixes dades es troben repetides.	No. Totes les aplicacions disposen de la mateixa base de dades
Usuaris	1	Múltiples. La base de dades és compartida per molts usuaris de diferents tipus

3.3. Tercera generació

Avui dia, els SGBD relacionals estan en plena transformació per a adaptar-se a tres tecnologies d'èxit recent fortament interrelacionades: la multimèdia, l'orientació a l'objecte (OO), i Internet i el web. Aquestes tres tecnologies disposen d'una creixent complexitat de les dades i de les aplicacions que les utilitzen, fet que ha provocat l'aparició de "nous" SGBD:

- OODBMS: bases de dades orientades a objectes.
- ORDBMS: bases de dades objecte-relacionals.
- NoSQL DBMS: bases de dades que no utilitzen el SQL com a llenguatge per a realitzar consultes.

3.3.1. OODBMS: Bases de dades orientades a objectes

Las bases de dades orientades a objecte sorgeixen per a solucionar les principals limitacions que presenta el model relacional:

- Necessitat d'ús de moltes taules - taules intermèdies- per a donar suport a estructures de dades complexes.
- L'ús excessiu de taules intermèdies penalitza el rendiment i complica la recuperació de dades.
- La creació de tipus està altament limitada.
- Manca de mecanismes de reutilització.

Així doncs, aquest nou model incorpora tots els conceptes importants del paradigma d'orientació a objectes:

- Encapsulació: Ocultar informació a la resta d'objectes denegant accessos incorrectes o conflictius.
- Herència: Els objectes hereten comportament en funció d'una jerarquia de classes.
- Polimorfisme: Permetre mètodes amb el mateix nombre i diferent quantitat d'arguments.

3.3.1.1 Avantatges

Les principals avantatges que incorporen aquest tipus de sistema són els següents:

- Permeten emmagatzemar objectes més complexos.
- Poden definir mètodes i procediments complexos "a l'estil Java".
- Inclouen polimorfisme.
- Integració transparent amb el programa: s'emmagatzema exactament el mateix model d'objecte.
- Reducció dels costos de desenvolupament i millora del rendiment.

3.3.2. ORDBMS: Bases de dades objecte-relacional

Les bases de dades objecte-relacional són una extensió de les bases de dades relacionals tradicionals a les que se'ls hi proporcionen característiques de la programació orientada a objectes, és a dir, es tracta d'un model de transició entre el model relacional i el model orientat a objectes. Permeten relacions niades, herència – de tipus i de taula – i tipus de dades complexos: col·leccions ("*arrays*" o

"multisets"); tipus estructurats (equivalent als atributs compostos del model relacional); i, objectes de gran grandària (varis MB o inclús GB).

3.3.2.1 Avantatges

Les principals avantatges que incorporen aquest tipus de sistema són les següents:

- Permeten emmagatzemar objectes més complexos.
- Poden definir mètodes i procediments "complexos" (a l'estil de Java).
- Inclouen polimorfisme.

3.3.2.2 Comparativa

A continuació es mostra una taula comparativa de les característiques més importants en relació amb el model de SGBD orientat a objecte :

Característica	OODBMS	ORDBMS
Tipus de base de dades	Extensibles mitjançant tipus de dades complexes	Extensibles mitjançant l'ús de nous tipus de dades (referència, col·lecció i "row") i definits per l'usuari
Tipus de dades complexes	Objectes complexes	Tipus estructurats complexes
Relacions	1:M, M:N, "is a" i "extends"	1:M, M:N i "extends"
Reutilització	Mitjançant herència d'objectes	Mitjançant herència de tipus estructurats i taules "tipades"
Encapsulat	Com a la programació a objectes	Com a la programació a objectes
Portabilitat	Baixa (pocs estàndards acceptats)	SQL 2003
Corba aprenentatge	Elevada	Moderada
Integritat referencial	Si. Referències creuades	Només claus foranes
Accés	Orientat a conjunts	Semblant al model relacional
Rendiment	Consultes eficients gracies a l'ús de referències entre objectes.	Consultes eficients gracies a l'ús de referències entre tipus de dades estructurades.
Eficiència	Optimitza accés a disc	Semblant al model relacional

3.3.3. NoSQL

Amb el creixement de la web en temps real apareix una necessitat per a poder gestionar grans volums de dades prioritzant el rendiment i la informació "a l'instant" per damunt de la coherència de les dades.

Les bases de dades NoSQL són sistemes d'emmagatzemament d'informació que no compleixen amb l'esquema entitat-relació. En aquest sentit, no imposen una estructura de dades en forma de taules i relacions entre elles -tampoc disposaran de *joins* ni transaccions ACID[2] -, sinó que permeten emmagatzemar la informació en altres formats com clau/valor, *mapping* de columnes, documents o grafs.

Aquest tipus d'SGBD estan pensats per a manipular volums enormes de dades de manera molt ràpida, i és per això que acostumen a emmagatzemar tota la informació que es pot a memòria (utilitzant el disc per a realitzar la persistència). Una altra característica important és que es poden escalar horitzontalment (es permet afegir noves màquines "en calent") sense penalitzar el rendiment.

Existeixen diferents tipus de bases de dades NoSQL: sistemes-documentals, en graf, clau/valor, multi valor, orientades a objecte i tabulars, tot i que les més conegudes són les clau/valor. Les bases de dades més populars són MongoDB; Cassandra d'Apache; BigTable de Google; Dynamo d'Amazon; i, Project Voldemort de LinkedIn.

[2]Acrònim d' **A**tomicity, **C**onsistency, **I**solation and **D**urability: Atomicitat, Consistència, Aïllament i Durabilitat. http://es.wikipedia.org/wiki/ACID

3.3.3.1 Avantatges

Les principals avantatges que incorporen aquest tipus de sistema són les següents:

- Poden tractar enormes quantitats de dades.
- Millora del rendiment: no generen “colls d'ampolla”.
- Alta escalabilitat i molt senzilla d'utilitzar.
- S'executen en clústers de màquines econòmiques.

3.3.3.2 Inconvenients

En relació als inconvenients, disposen dels següents:

- Pèrdua de flexibilitat en temps d'execució.
- Poques operacions: bàsicament de recuperació i agregació de registres.

4. PERSISTÈNCIA DE DADES

Persistència és una qualitat de persistir i significa "durar llarg temps". Així doncs, persistència de dades en un context tecnològic es podria definir com “emmagatzemament de la informació de manera permanent”. D'aquesta definició es pot deduir que la informació disposarà d'un període de vigència superior a la execució del programa i, per tant, serà accessible tot i que aquest hagi finalitzat. Així doncs, *a posteriori*, la informació pot ser explotada o, simplement, utilitzada per a termes d'auditoria.

Aquesta informació, constitueix, cada vegada més, un actiu fonamental de les empreses i està esdevenint una nova revolució de la societat; la qual no depèn tan sols de la millora que els computadors han produït en els processos, sinó també, en la capacitat de generar valor per a obtenir avantatges competitius. És per això, que l'emmagatzemament de les dades representa una de les tasques més importants a l'hora d'implementar qualsevol sistema de programari.

D'altra banda, cal remarcar que la persistència de les dades representa una de les parts més crítiques del disseny de qualsevol sistema programari. És, generalment, el "coll d'ampolla" de la majoria de projectes - cal recordar que les operacions d'*entrada-sortida* són les que més penalitzen el rendiment d'una aplicació - i, per tant, representa un pes molt important a l'hora d'avaluar-lo per a poder determinar definitivament l'èxit o el fracàs del sistema desenvolupat.

En aquest capítol es descriuran els objectius dels SGBD's, els principals mecanismes de persistència i es farà èmfasi especial en el *mapping objecte-relacionat* (ORM) donat que es tracta del sistema que s'utilitzarà per a desenvolupar el framework de persistència del present PFC.

4.1. Objectius dels SGBD's

A continuació, es mostren els principals objectius que disposen la majoria dels SGBD's que es troben al mercat:

- **Consultes no predefinides i complexes:** L’usuari ha de poder formular consultes de qualsevol tipus i complexitat (*adhoc*) amb un llenguatge senzill**,** i el sistema l’ha d’interpretar directament. La solució estàndard per a assolir aquest doble objectiu és el llenguatge SQL.

- **Flexibilitat i independència:** Interessa obtenir la màxima independència possible entre les dades i els processos usuaris perquè es puguin fer tot tipus de canvis tecnològics i variacions en la descripció de la BD, sense que s’hagin de modificar els programes d’aplicació ja escrits ni canviar la manera d’escriure les consultes.

- **Problemes de la redundància:** Ha de tenir en compte el greu risc d’inconsistència o incoherència de les dades, és a dir, la pèrdua d’integritat que les actualitzacions poden provocar quan hi ha redundància. Els SGBD han de permetre que el dissenyador defineixi

dades redundants, però llavors hauria de ser el mateix SGBD el que fes automàticament l'actualització de les dades a tots els llocs on estiguessin repetides.

- **Integritat de les dades:** L'SGBD incorporarà unes regles d'integritat inherents al sistema i facilitarà d'altres perquè l'usuari pugui definir-ne de noves en funció del negoci (delega la restricció d'aquestes a l'SGBD). Així doncs, quan l'SGBD detecti que un programa vol fer una operació que va contra les regles establertes -inherents o "de negoci"- , no ho haurà de permetre, i haurà de retornar un estat d'error.

- **Concurrència d'usuaris:** Els SGBD's han de permetre la concurrència d'usuaris i s'ha de tenir especial atenció en les escriptures dels registres per a no provocar dades inconsistents. Per aquest motiu, els SGBD han d'aconseguir que el conjunt d'operacions d'una transacció mai no s'executi parcialment. O s'executen totes o no se n'executa cap.

- **Seguretat:** El terme seguretat que es sol utilitzar per a fer referència als temes relatius a la confidencialitat, les autoritzacions, els drets d'accés, etc. Addicionalment, ens pot interessar emmagatzemar la informació amb una codificació secreta, és a dir, amb tècniques d'encriptació que molts SGBD's ja tenen previstes.

4.2. Mecanismes de persistència

A continuació s'enumeren els principals mecanismes de persistència:

- **Accés directe a base de dades**: Aquest mecanisme utilitza directament la base de dades per implementar la persistència sense cap tipus de *middleware* entre l'aplicació i el SGBD. En funció del tipus d'SGBD disposarem de diferents opcions: accés directe a base de dades relacional, accés directe a base de dades objecte-relaciona, accés directe a base de dades orientada a objectes i accés directe a base de dades XML.

- ***"Mappers"***: Mecanismes basats en la traducció bidireccional entre les dades encapsulades en els objectes de la lògica d'un sistema orientat a objectes i una font de dades que utilitza un paradigma diferent. Serà l'encarregat de "barallar-se" amb els problemes que sorgeixin en el mapping entre els diferents paradigmes. El conjunt de problemes que es produeixen fruit d'aquestes diferencies s'anomena *impedance mismatch*. Existeixen dos tipus de mappers: Mapping Objecte-Relacional (els que s'utilitzaran per a realitzar el framework), i Mapping Objecte-XML.

- **Generadors de codi:** Aquesta solució resol la persistència utilitzant eines que generen codi correcte, robust i que aplica els patrons de disseny oportuns a partir de la metadata d'un projecte.

- **Orientació a aspectes**: La programació orientada a aspectes (POA o AOP en anglès) és un paradigma que permet definir abstraccions que encapsulen característiques transversals al sistema. En aquest paradigma les classes son dissenyades i implementades de forma separada als aspectes i, posteriorment es fusionen. Aquest mecanisme tracta la persistència com un aspecte ortogonal a les funcionalitats, desacoblant el codi corresponent a la resta del sistema i, d'aquesta manera, es modularitza la persistència per a poder reutilitzar el codi generat.

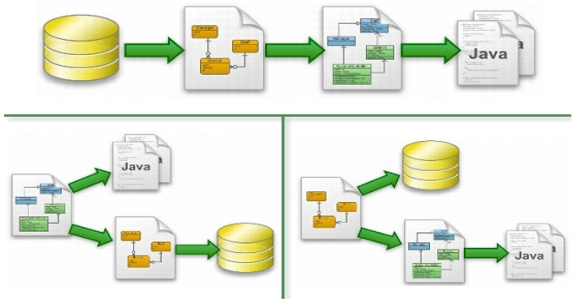
- **Llenguatges orientats a objectes:** Aquesta solució preveu resoldre la persistència utilitzant les funcionalitats del propi llenguatge de programació:
  - Llibreries estàndard: La idea fonamental del mecanisme correspon a l'ús d'un llenguatge de programació orientat a objectes. Amb aquesta estratègia s'evita incloure frameworks aliens al llenguatge.
  - Llenguatges persistents: Aquest mecanisme es basa en els tres principis de persistència ortogonal:
    - Cada objecte, independent del seu tipus, disposa del mateix dret a ser persistit.
    - Només es persistiran els objectes accessibles des de l'arrel de persistència.

    - La introducció de persistència no pot introduir canvis semàntics en el codi font.
  - Llenguatge de consulta integrat: El mecanisme proposa l'ús d'un llenguatge de programació que integri com a part del mateix un llenguatge de consulta i administració de dades persistents.

- **Esquemes de prevalència:** Són frameworks que mantenen tots els objectes en memòria volàtil i utilitzen, periòdicament, mecanismes de serialització d'objectes per guardar una imatge (*snapshot*) dels objectes de l'aplicació. És un mecanisme molt simple i ultra ràpid tot i que presenta algun problema d'escalabilitat.

4.3. *Mapping* d'objectes relacional (ORM)

Com s'ha comentat anteriorment, aquest serà el mecanisme que s'utilitzarà per a desenvolupar el framework del PFC i, per això, s'explicarà més detalladament. El *mapping* d'objectes relacional (ORM, sigles en angles d'*Object-relational mapping*) és una tècnica de programació per convertir dades de llenguatges de programació orientats a objectes en la seva representació en bases de dades relacionals, a través de la definició de les correspondències entre els diferents sistemes. Gràficament:



La relació entre els conceptes del model objecte i els del model relacional és la següent:

Model OO	Model relacional
Classe	Taula
Propietat	Camp
Objecte	Fila
Identificador	Clau primària
Punter a altre objecte	Clau forana

Aquesta correspondència és molt "natural" donat que, en realitat, el model orientat a objectes i el relacional no són tant diferents. A continuació, s'explicaran les característiques més rellevants que incorporen les diferents implementacions.

4.3.1 Patró CRUD

Acrònim de *Create-Read-Update-Delete*. Es considera el pare de tots els patrons de la capa d'accés. Cada objecte ha de ser creat en la base de dades per a que sigui persistent. Un cop creat, la capa d'accés ha de disposar d'un mode per a que es llegeixi i permetre actualitzar-lo o esborrar-lo. La tasca d'esborrat, tot i que teòricament l'hauria de realitzar la pròpia base de dades, queda delegada al programador per motius d'eficiència en bases de dades amb grans volums de dades.

4.3.2 Càrrega de les relacions

La càrrega de les relacions és una altra aspecte que caldrà abordar un cop s'hagi realitzat el mapping entre la relació i l'objecte. Generalment, no interessa carregar tots els objectes associats a un determinat objecte donat que en casos de bases de dades amb grans volums de dades aquesta càrrega

serà molt ineficient. Per a solucionar aquest problema existeix la "càrrega retardada" (o lazy) que consisteix en carregar els objectes només quan s'accedeix (patró Proxy). En determinades ocasions, pot interessar realitzar la "càrrega directa" per poder disposar de les relacions carregades sense necessitat de realitzar un segon accés a la base de dades. Cal destacar que les càrregues directes són operacions que s'han de realitzar d'una manera molt controlada per evitar problemes de latència.

4.3.3 Concurrència

La capa de persistència ha de permetre accedir a múltiples usuaris a la mateixa taula i, per tant, és fonamental la integritat de les dades. Per evitar aquest problema existeixen dues tècniques:

- **Bloqueig pessimista**: No es permet l'accés fins que l'usuari que modifica les dades fa el *commit* o el *rollback*. Aquesta tècnica incorpora dos inconvenients molt importants:
  - Bloqueig: Es pot donar quan un usuari selecciona un registre per a modificar i seguidament abandona la operació. La resta d'usuaris que han d'actualitzar el registre romandran bloquejats fins que es completi la transacció.
  - "Deadlock": Si dos usuaris A i B estan actualitzant la base de dades a la vegada i A bloqueja un registre i intenta obtenir un bloqueig mantingut per B, que a la seva vegada està esperant obtenir un bloqueig mantingut per A; ambdues transaccions quedaran en espera indefinidament i, per tant, es produirà un bloqueig permanent o *deadlock*.

- **Bloqueig optimista**: No bloqueja els registres que s'actualitzen i assumeix que les dades que estan essent actualitzades no canviaran des de que s'han llegit.

4.3.4. Memòria cache

En la majoria de les aplicacions s'aplica la regla del 80-20 en termes d'accés a les dades, és a dir, el 80% dels accessos de lectura corresponen al 20% de les dades de l'aplicació. Així doncs, "pocs" objectes són accedits freqüentment, i per tant, interessarà algun mecanisme per optimitzar aquestes consultes tant freqüents.

La memòria cache és una memòria temporal, generalment d'existència transparent per l'usuari, que proporciona un accés ràpid a les dades d'ús freqüent. La manera de treballar de la memòria cache és simple: les dades sol·licitades es cerquen, en primer lloc, a la memòria cache i, en cas que no estiguin disponibles, es realitza la cerca al motor de base de dades.

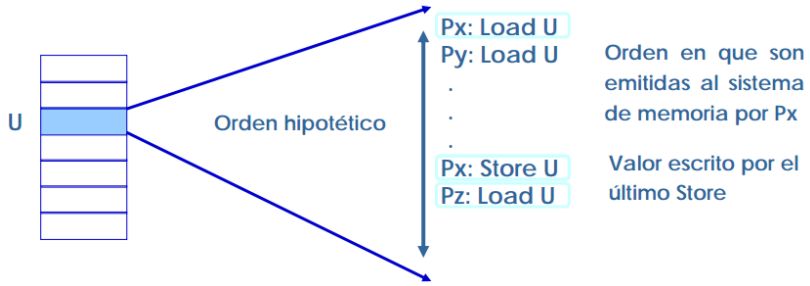
Existeixen una sèrie de mecanismes per a l'administració i gestió de la memòria cache: Write Through (o Write Update) on la dada s'escriu "alhora" en la memòria cache i en la base de dades; o Write Behind, per a escriure en mode "asíncron".

Per a preservar un objecte en la memòria cache, aquest ha de ser utilitzat freqüentment, donat que l'avantatge bàsica que proporcionen aquests mecanismes consisteix en eliminar els temps d'accés a un recurs extern per a recuperar dades (quan es troben emmagatzemats a la cache l'accés a memòria és "immediat").

Finalment, la implementació d'un mecanisme de memòria cache no és "gratuïta", donat que pot incorporar molts problemes. El problema per excel·lència d'aquests sistemes és la incoherència de dades en sistemes distribuïts (entre els diferents nodes del clúster) i per tant, s'haurà de tenir molt de compte quan es decideixi implementar una solució d'aquest tipus. A continuació, es detallaran les diferents característiques d'aquests mecanismes.

4.3.4.1 Coherència

Coherència fa referència a que el resultat de qualsevol lectura sempre dóna el valor de la escriptura més recent, és a dir, la coherència s'aconsegueix quan qualsevol escriptura es vista per una lectura i totes les escriptures son vistes en l'ordre correcte (serialització):



Pel manteniment coherent de les cachés s'utilitzen diferents mecanismes :

- **Protocol *Snoopy***: El seu mecanisme consisteix a distribuir la informació permanentment utilitzant alguna de les següents polítiques:
  - *Invalidar-si-escriptura (write-invalidate):* poden haver múltiples nodes que llegeixen però, en un moment donat, només un escriu.
  - *Actualitzar-si-escriptura (write-update o writebroadcast):* poden haver-hi diferents nodes que escriuen mentre diversos nodes llegeixen. Quan algú vol escriure en un registre compartit es fa un *broadcast* als diferents nodes i les caches que contenen aquest registre, el poden actualitzar.

- **Protocol basat en directori**: Implementa una estructura -anomenada taula o directori – que registra quin node ha emmagatzemat en la seva cache els diferents blocs de la base de dades. Aquesta solució redueix el tràfic en la xarxa donat que només s'envien ordres a les caches que disposen d'una còpia vàlida del bloc implicat en la operació.

4.3.4.2 Implementacions

Existeixen diverses implementacions *OpenSource* en el mercat Java per a realitzar aquestes tasques de "caching" tant positives pel rendiment del framework de persistència: OSCache, EHCache, SwarmCache, JBoss Tree Cache. També, es podria realitzar una implementació "*custom*", tot i que aquesta darrera opció es poc recomanable donada la difícil gestió de les dades en entorns distribuïts (amb clúster). És per això, que s'aconsella l'ús de solucions existents de caches distribuïdes donat que abstrauran al programador de la complexitat inherent a la gestió de repliques entre els diferents nodes d'un clúster.

A continuació es mostren les principals característiques de les diferents implementacions OpenSource disponibles al mercat Java:

- **OSCache**
  - Poderosa
  - Flexible
  - Suporta estratègies de cache: read-only i read-write
  - Suporta cache en memòria i disc
- **EhCache**
  - Lleugera
  - Ràpida
  - Fàcil d'utilitzar
  - Suporta estratègies de cache: read-only i read-write
  - Suporta cache en memòria i disc

- **SwarmCache**
  - Basada en clustering (JGroups)
  - Suporta estratègies de cache: read-only i read-write
  - Recomanable per a sistemes amb més operacions de lectura que d'escriptura
- **JBoss TreeCache**
  - Transaccional i en clúster (basada en JGroups)
  - Ideal quan es requereix una arquitectura de cache transaccional

En el nostre cas, s'estudiarà amb més detall la solució desenvolupada per Terracota i que utilitza Hibernate com a caché de segon nivell, EhCache.

4.3.4.2.1 EhCache

EhCache és una de las llibreries més populars en la gestió de la cache d'objectes en Java. És OpenSource i està basada en la divisió de la cache en diferents "regions" (que són configurables) on s'emmagatzemen els objectes. Disposa d'un XML de configuració per a poder especificar els comportaments de la cache amb les múltiples opcions que presenta.

Las principals característiques que presenta aquesta llibreria són:

- **Petita** i amb poques dependències.
- **Ràpida i simple d'utilitzar**: La corba d'aprenentatge és molt petita gràcies a la simplicitat del seu disseny i el seu rendiment és extraordinari (incrementa la velocitat d'accés als objectes tant des de memòria com des de disc).
- **Escalable**: Suporta múltiples caches i és persistent (permet emmagatzemar les dades de la memòria cache a disc).
- **Flexible**: Permet treballar amb Objects i *Serialitzable*, i disposa d'una configuració senzilla mitjançant XML o programàticament. En cas de no configurar-se res, arrenca amb les configuracions per defecte.
- **Extensible**: Permet afegir *plug-ins*, loaders, *listeners*, *JMX* i *handlers* d'excepcions. Pot treballar conjuntament amb altres APIs – JPA - o frameworks -Hibernate-.
- **Distribuïda**: Disposa de múltiples opcions per a la replicació de les dades.
- Incorpora **diferents suports per a emmagatzemar les dades.**
- Es disposa de **diferents polítiques d'expulsió** dels elements de la caché:
  - LRU: S'elimina l'objecte que porta menys temps sense utilitzar-se.
  - LFU: S'elimina l'objecte que s'utilitza menys.
  - FIFO: S'elimina el primer objecte que va entrar, és a dir, l'objectes més antic.

A continuació es mostra un petit exemple que mostra aquesta facilitat d'ús. Aquestes llibreries "pengen" del *package net.sf.ehcahe.** i, per tant, caldrà importar a l'inici de la classe de test:

```
// test.java
import net.sf.ehcache.CacheManager;
import net.sf.ehcache.Element;
```

Crear una cache programàticament és molt simple:

```
CacheManager.getInstance().addCache("test");
```

Per afegir un nou objecte a la cache, només caldrà recuperar-la i afegir-hi un objecte Element (una tupla clau-valor) mitjançant el mètode *put*:

```
//  mètode que crea una nova Persona amb el ID, Nom i Cognoms
Person carlos= new Person (111111, Carlos, Mena);
Element me= new Element(carlos.getID(), carlos)
// s'afegeix la persona a la cache
CacheManager.getInstance().getCache("test").put(me);
```

Per obtenir un determinat valor en funció de la clau, es tant simple com fer:

```
Element elt=CacheManager.getInstance().getCache("test").get(111111);
// Retornaria la instància de persona amb les dades corresponent a  111111
return (elt == null ? null : elt.getObjectValue());
```

Finalment, cal comentar que es pot substituir la línia de creació de la caché, per un fitxer - *EhCache.xml* - que permetrà crear-la amb la configuració més adient per cada cas:

```
<cache name="test"
          maxElementsInMemory="10000"
          maxElementsOnDisk="1000"
          eternal="false"
          overflowToDisk="true"
          timeToIdleSeconds="300"
          timeToLiveSeconds="1"
          memoryStoreEvictionPolicy="LFU"
/>
```

Així doncs, treballar amb aquesta cache distribuïda resulta molt senzill.

5. FRAMEWORKS DE PERSISTÈNCIA DE DADES

En termes generals, *framework* – marc de treball - defineix un conjunt estandarditzat de conceptes, pràctiques i criteris per enfocar una determinada problemàtica que serveix com a referència per a resoldre problemes similars.

En desenvolupament de programari, un *framework* és una estructura conceptual i tecnològica definida amb artefactes o mòduls de programari concrets, que esdevenen la base per a que un altre projecte de programari pugui ser més fàcilment organitzat i desenvolupat.

Així doncs, les principals avantatges de l'ús d'un framework són les següents:

- Reutilitza components de programari.
- Agilitza el desenvolupament d'aplicacions. Els components inclosos construeixen una capa que abstrau al programador la complexitat inherent d'aquelles funcionalitats que implementa el *framework* que permet que es centri en el disseny i desenvolupament del "negoci" de l'aplicació.

D'altre banda, els inconvenients principals que introdueixen són:

- Demanda de grans quantitats de recursos computacionals donat que tendeixen a generalitzar la funcionalitat dels components. Generalment, els propis *frameworks* disposen d'un gran nombre de llibreries addicionals que cal incorporar al projecte.
- Reducció de la flexibilitat. Els programadors s'han d'adaptar a les funcionalitats que ofereix en funció de l'arquitectura del mateix.

Actualment existeixen nombrosos *frameworks* de persistència de dades que permeten implementar la capa de model[3] de JEE d'una manera més simple. Aquest ampli ventall d'opcions implica un estudi previ per a poder determinar quina de les opcions s'adapta millor a les necessitats concretes de la aplicació en qüestió.

[3] Model Vista Controlador (MVC) és un patró de desenvolupament de programari que separa les dades de la aplicació (model), la interfície d'usuari (vista) i la lògica de negoci (controlador) en tres components. El model correspon al Sistema de gestió de la base de dades (SGBD). http://es.wikipedia.org/wiki/Modelo_Vista_Controlador

Algunes de les característiques que cal avaluar abans de decidir-se per una opció o una altre són les següents: corba d'aprenentatge, rendiment, facilitat d'ús (complexitat), escalabilitat, seguretat o eines de suport.

Aquest capítol es centrarà en mostrar els *frameworks* més importants del moment. Com s'ha comentat anteriorment, només es tindran en compte els SGBD's relacionals i per tant, els *frameworks* estudiats formaran part d'aquest model de SGBD's. En primer lloc, es descriuran els requisits i característiques principals dels frameworks de persistència, incorporant una introducció a l'estàndard que utilitza Java per a la connexió amb les bases de dades – JDBC -. A continuació, es mostrarà l'estudi dels *frameworks* – especificació de les característiques principals de cadascun- i es finalitzarà amb una comparativa entre ells.

5.1. Requisits i característiques principals

En primer lloc, s'enumeren els requisits dels *frameworks* de persistència:

- Emmagatzemar i recuperar els objectes en un mecanisme d'emmagatzemament persistent.
- Confirmar i desfer les transaccions
- Presentar un disseny extensible per donar suport a diferents mecanismes d'emmagatzemament (bases de dades relacionals, arxius o documents XML)

Les principals característiques dels *frameworks* són les següents:

- Correspondència d'esquemes.
- Identitat d'objectes i registres (representació d'una fila).
- Materialització (transformació d'un registre en un objecte) i des-materialització (transformació d'un objecte a un registre).
- Conversor de base de dades: responsable de la materialització i la des-materialització.
- Suport a les transaccions: operacions de *commit* i *rollback.*

Altres característiques que poden suportar - i que provoquen millores en el temps de resposta - són les següents:

- Ús de *caches* per a emmagatzemar els objectes materialitzats.
- Materialització "lazy": Una instància es materialitza sota demanda.
- *Proxies* virtuals per a implementar la materialització "lazy".

En relació a les avantatges que ens aporten, són les següents:

- Rapidesa en el desenvolupament: Permeten crear automàticament el model mitjançant la connexió amb la base de dades i, per tant, s'eviten errors del programador.
- Disposen d'un llenguatge propi per a les consultes.
- Abstracció de la base de dades: Es separa totalment la capa de dades del codi, amb el que es facilita la migració en cas d'un canvi de motor de base de dades.
- Reutilització: Es permet utilitzar els mètodes d'un objecte des de diferents parts del codi.
- Seguretat: Acostumen a implementar sistemes de seguretat per evitar atacs com per exemple l'SQL injection[4].
- Manteniment: Disposar de la capa de dades ordenada simplifica el manteniment del codi.

Finalment, els inconvenients que incorporen són:

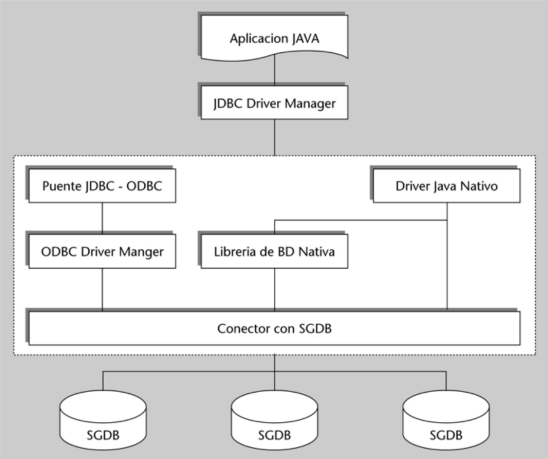
- Corba d'aprenentatge: Generalment, aquestes eines son complexes i, per tant, es requereix d'un temps d'aprenentatge abans de poder utilitzar-les. En el cas d'utilitzar un llenguatge propi encara resulta més complexe.
- Aplicacions més lentes: Transformar les consultes al llenguatge propi afegeix un cost que penalitza el rendiment.
- Increment del pes de "l'aplicació de negoci": Generalment, requereixen llibreries extra – a banda de la pròpia del *framework* – i, per tant, augmenta considerablement la grandària de "l'aplicació de negoci".

[4] Injecció SQL o (*SQL Injection* en anglès) és un mètode d'infiltració de codi intrús que utilitza una vulnerabilitat informàtica present en una aplicació que permet accedir a les dades de la base de dades.

5.2. JDBC

L'accés a la base de dades des de Java es realitza mitjançant l'estàndard JDBC – Java Data Base Connectivity - que permet l'accés uniforme a les bases de dades independentment del SGBD. D'aquesta manera, les aplicacions escrites en Java no necessiten conèixer les especificacions d'un SGBD en particular i, per tant, l'abstrau d'aquesta complexitat.

Per utilitzar aquesta API amb un SGBD en particular es necessitarà un adaptador concret del motor de base de dades (*driver*) que farà d'intermediari entre la tecnologia SGBD i la base de dades. A continuació, es mostra l'estructura de JDBC de manera gràfica:



JDBC ofereix el paquet *java.sql* en el que destaquen les següents classes:

- ***DriverManager***: Carrega un driver d'un SGBD determinat.
- ***Connection***: Estableix connexions amb la base de dades.
- ***Statement***: Executa sentències SQL i les envia a la base de dades.
- ***ResultSet***: Emmagatzema el resultat d'una consulta.

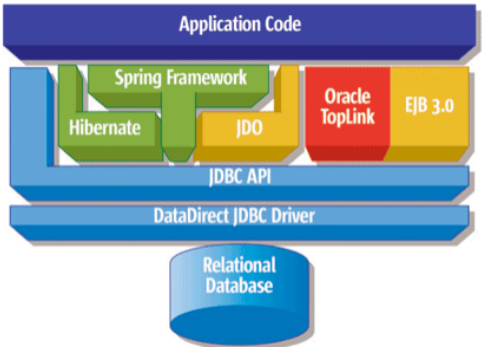
En relació a les sentències, incorpora tres tipus diferents:

- ***Statement***: *S'*envien les sentències a la base de dades cada vegada.
- ***Prepared statement***: Permeten la "precompilació" del codi SQL abans de ser executat amb el que s'obté una millora considerable en el rendiment.
- ***Callable statement***: S'utilitza per a executar store procedures a la base de dades.

Finalment, cal destacar que l'API inclou suport per transaccions, de manera que es pot desfer un conjunt d'operacions relacionades en cas de ser necessari. Per defecte, el comportament serà atòmic, és a dir, s'executarà cada sentència en el moment; i en cas que es vulgui modificar el comportament per a cedir-li el control a l'aplicació, es podrà realitzar mitjançant el mètode *setAutoCommit(false)*.

5.3. *Frameworks*

A continuació, es mostraran els *frameworks* de persistència més populars a dia d'avui, que estan situats entre la capa de codi de l'aplicació Java i la de l'API JDBC:



5.3.1. Java Persistence API (JPA)

Java Persistence API – JPA, en endavant- proporciona un model de persistència basat en POJO's per a fer el mapping amb bases de dades relacionals en Java. La JPA va ser definida com a part de l'especificació de l'estàndard EJB 3.0, que al seu temps també és part de la plataforma Java EE 5.

L'estàndard en qüestió és el resultat del treball d'un grup d'experts que es van basar en les idees dels frameworks lliures líders del mercat com: Hibernate, *Oracle TopLink* i Java Data Objects (JDO) entre d'altres. JPA posa èmfasis en definir la metadada mínima necessària pel mapping d'objectes Java a una base de dades relacionals. En aquest sentit, l'objectiu fonamental és el de convertir-se en l'estàndard per excel·lència per a la utilització objecte-relacional dins de la plataforma i que sigui implementat per diferents proveïdors aconseguint definir una interfície comú.

El mapping objecte-relacional, és a dir, la relació entre les entitats Java i les taules de la base de dades, es realitza mitjançant anotacions en les pròpies classes de la entitat, amb el que no es requereixen fitxers descriptius XML. Les transaccions també es poden definir com anotacions JPA.

Les característiques fonamentals que proporciona són les següents:

- Abstrau del codi del proveïdor de persistències, és a dir, de la implementació de l'estàndard.
- Incorpora llenguatge de consultes orientat a objectes (JPQL[5]): Elimina la necessitat d'escriure codi SQL, les consultes es realitzaran sobre els objectes i no sobre les taules.
- Permet relacions molts-a-un (*ManyToOne*), un-a-un (*OneToOne*), un-a-molts (*OneToMany*) i molts-a-molts (*ManyToMany*) mitjançant les anotacions.
- Inclou control de la transaccionalitat de les consultes (utilitzant JTA[6] o JDBC).
- Incorpora una cache de primer nivell que optimitza l'accés a les dades.
- Permet definir caches de segon nivell per a incrementar la escalabilitat i el rendiment.

Com s'ha comentat anteriorment, JPA es només una especificació per a solucionar la persistència d'un sistema i, per tant, es necessitarà una implementació. A continuació, es mostraran les més importants.

5.3.2. Top Link Essentials

TopLink Essentials es part del servidor d'aplicacions *Glassfish*. Concretament és la versió gratuïta del producte TopLink d'Oracle i correspon a la implementació de referència per la especificació JPA 1.0. En aquest sentit, proporciona un mecanisme altament flexible i productiu per emmagatzemar els objectes i els EJB's de Java en bases de dades relacionals.

Les característiques més destacades són:

- Coherència de dades
  - Configuració senzilla mitjançant anotacions alineades amb JPA.
  - Suport per a l'emmagatzemament d'entitats.
- Tractament XML
  - Inclou suport per a l'assignació d'objectes Java a XML.
- Control de concurrència
  - Ofereix ambdues possibilitats:
    - Bloqueig pessimista.
    - Bloqueig optimista. Es pot implementar de dues maneres:
      - Basat en camp *Versió*: Consisteix en afegir un camp versió (de tipus enter) a totes les taules que s'actualitza cada vegada que es modifica un registre.
      - Basat en camp *Timestamp*: Consisteix en afegir un camp de marca de temps (de tipus date) que s'actualitza cada vegada que es modifica un registre.
- Cache
  - Es pot configurar per a que emmagatzemi el resultat d'una consulta en una caché interna.

[5] Acrònim de **J**ava **P**ersistence **Q**uery **L**anguage
http://en.wikipedia.org/wiki/Java_Persistence_Query_Language
[6] Acrònim de **J**ava **T**ransaction **A**PI (API para transacciones en Java)
http://en.wikipedia.org/wiki/Java_Transaction_API

5.3.3. Hibernate

Hibernate és el framework ORM per excel·lència de la plataforma Java, tot i que també disposa d'una versió per a *.Net* - *NHibernate* –. No requereix cap tipus especial de contenidor Java EE per a funcionar, és a dir, pot ser utilitzat en un servidor d'aplicacions o en una aplicació *standalone*.

L'objectiu del framework, com la resta d'ORM, és abstraure al programador de la capa de persistència de dades i està dissenyat per a ser flexible en relació a l'esquema de taules utilitzat, i per tant, per a poder adaptar-se a una base de dades ja existent.

Hibernate es basa en una filosofia de fer el mapping a POJO's. En aquest sentit, permet utilitzar els objectes Java definits per l'usuari "tal qual", és a dir, no utilitza tècniques de generació de codi a partir de descriptors del model de dades o manipulació de *bytecodes* en temps de compilació, ni obliga a implementar interfícies determinades, ni heretar de cap superclasse, sinó que utilitza el mecanisme de reflexion de Java. Aquest fet, permet obtenir un modelat iteratiu fluït i natural basat en UML, fonamental per obtenir un treball àgil i productiu.

Per una altra banda, disposa d'un conjunt d'utilitats – *Hibernate Tools* – disponibles per ANT i Eclipse que facilita el seu ús. Una de les funcionalitats més útils que incorpora aquesta eina consisteix en generar els POJO's a partir de l'esquema de base de dades, i d'aquesta manera, evitar la farragosa tasca de mapejar les entitats de la base de dades a classes Java.

Les característiques més rellevants són les següents:

- Es distribueix sota llicència LGPL.
- Disposa d'un llenguatge propi de consultes -HQL- i una API per a construir les consultes programàticament - *Criteria* -.
- Permet configuració amb XML o anotacions.
- Implementa l'estàndard JPA.
- Productivitat: Evita codi confús, i permet al programador centrar-se en la capa de negoci.
- Mantenibilitat: Simplifica el codi i permet identificar els errors més fàcilment.
- Rendiment: Permet afegir mecanismes de cachés de segon nivell.
- Independència del proveïdor: Abstrau del SGBD utilitzat.
- Disposa d'una comunitat molt extensa.
- Incorpora suport a transaccions.
- Permet herència i polimorfisme.
- Disposa de mecanismes de càrrega lazy.
- Incorpora estratègies de *fetching* i persistència transitiva.

5.3.4. Cayenne

És un motor de persistència de codi obert publicat sota la Apache License que proveeix mapping objecte-relacional (ORM) i serveis d'accés remot (am*b Remote Object Persistence* es possible fer persistents els objectes Java a nivell de clients mitjançant serveis web). Incorpora un motor generador de classes basat en Velocity i una eina, *CayenneModeler*, que permet realitzar enginyeria inversa.

Les principals característiques són:

- Herència d'objectes.
- Implementa càrrega lazy, per defecte, en les relacions per entre objectes.
- Portabilitat: Sempre que s'utilitzi JDBC com a controlador.
- Permet configurar caches.
- Paginació de resultats.
- Configuració de bloc optimista per garantir la integritat de les dades.

5.3.5. myBatis

myBatis – anteriorment iBATIS – és un framework de persistència lliure i desenvolupat sota la llicència Apache 2.0 que elimina gairebé tot el codi JDBC, l'establiment manual dels paràmetres i la obtenció de resultats. Permet configuracions mitjançant XML o anotacions.

No es un un framework ORM i, per tant, no fa mapping d'objectes Java a taules; sinó de mètodes a sentències SQL. En aquest sentit, permet utilitzar totes les funcionalitats de la base de dades com procediments emmagatzemats, vistes o consultes complexes i permet relacionar "maps" i POJO's amb registres de base de dades. Així doncs, es tracta d'una eina molt aconsellable per a base de dades *legacy*, desnormalitzades o, simplement, quan es vol tenir control total sobre l'SQL que s'executa.

Les característiques més importants són:

- Simplificació de la programació envers JDBC "raw": reducció d'errors.
- Proporciona un motor de resultats SQL a arbres d'objectes.
- Permet composició de sentències SQL dinàmiques mitjançant una llenguatge amb sintaxis XML.
- Suporta ús de caches declaratives: OSCache i EhCache entre d'altres.

5.4. *Comparatives*

5.4.1. Hibernate vs. TopLink

A continuació, es mostra una comparativa entre les dues implementacions més importants de l'estàndard JPA. En concret, es comparen la versió 3.2.1 d'Hibernate amb la v2.0b41 de TopLink:

Criteri	Hibernate	TopLink
Construcció manual d'SQL	✗	✗
Consultes dinàmiques	✓	✓
Creació automàtica d'esquema	✓	✓
DBMS suportats	JDBC compliant	JDBC compliant
Llenguatge de consulta OO	HQL	JPQL
Procés/Generació de codi	✗	✗
Relacions suportades	1 : 1 1 : N N : M	1 : 1 1 : N N : M
Suport Cascading	✓	✓

Com es pot observar, ambdues versions disposen de les mateixes funcionalitats i limitacions.

5.4.2. myBatis vs. ORM

A continuació es mostren les principals avantatges d'aquesta solució en relació als frameworks ORM:

- Més lleugera.
- Major control de les sentències.
- Millor adaptació a bases de dades *legacy* i/o *desnormalitzades.*
- Corba d'aprenentatge menor, ORM disposa dels següents inconvenients:
  - Funcionament no és intuïtiu.
  - Definició de relacions costosa.
  - Generalment requerix aprendre un llenguatge propi (HSQL, JPQL,...).

6. DISSENY *FRAMEWORK*

6.1. Introducció

La solució del model de classes està basada en els patrons i principis SOLID d'enginyeria del programari estudiats en les diferents assignatures d'enginyeria del programari (Enginyeria del programari Orientat a Objectes -EPOO-, Enginyeria del Programari de Components i Sistemes Distribuïts – EPCSD- i Procés d'Enginyeria del Programari -PEP-, entre altres). En aquest sentit, l'aplicació que es construeix estarà basada en:

- Patró Singleton: El patró de disseny singleton (instància única) està dissenya per a restrinfit la creació d'objectes pertanyents a una classe o el valor d'un tipus a un únic objecte. La intenció consisteix en garantir que una classe només disposi d'una única instància i proporcionar un punt d'accés global a la mateixa. Aquest patró s'ha utilitzat en la creació de totes les factories (*SessioFactory*, *ConfiguradorFactory* i *ConnexioFactory*) i en el *SQLHelper* que realitzarà la traducció de POJOs a registres de bases de dades.

- IOC (Inversion of Control): El principi d'inversió de control consisteix en que el control de la construcció dels objectes no recau directament en el desenvolupador mitjançant l'ús de l'operador new, sinó que la construcció dels objectes es delega a d'altres classes. En aquesta solució s'ha utilitzat en la creació dels objectes del component de Connexió (Sessio i Connexio) mitjançant una factoria.

- DRY (Don't Repeat Yourself): Qualsevol funcionalitat existent haurà d'existir de forma única, és a dir, no hauran d'haver-hi blocs repetits. Amb aquest principi s'aconsegueix evitar problemes de manteniment donat que el codi està en una única ubicació amb el que en cas de modificacions no hi haurà problemes de replicacions. En la solució hi ha nombrosos exemples que utilitzen aquest principi (interfícies constants per afavorir la reutilització, classes d'utilitats amb mètodes estàtics, herència entre classes, etc.)

- SRP (Simple Responsability Principle): Cada classe només disposarà d'una única responsabilitat. D'aquesta manera, es facilita la reutilització dels components.

- ISP (Interface Segregation Principle): Una classe A que te una dependència d'una classe B no ha de dependre del mètodes de la classe B que no hagi d'utilitzar.

- OCP (Open Close Principle): tot el codi desenvolupat per una aplicació ha d'estar tancat a les modificacions i obert a la extensibilitat, és a dir, s'ha de poder afegir noves funcionalitats a la aplicació sense impacte en el codi. Aquest principi s'ha utilitzat -mitjançant la API reflection de JAVA - per a la creació dels beans (en funció del driver del motor de base de dades necessari es carregaran unes classes o unes altres). Així doncs, si es volen afegir més motors de base de dades només caldrà modificar el fitxer de configuració i desenvolupar les noves implementacions (que hauran d'implementar les interfícies corresponents) i pel codi aquest canvi serà transparent.

6.2. Diagrama de casos d'ús

Abans de mostrar l'esquema amb els casos d'us del sistema, es considera oportú fer una breu introducció amb les decisions preses per a la realització del disseny.

Per una banda, es disposarà dels següents actors al sistema:

- **Usuari*:*** Correspon a l'usuari que utilitzarà el *framework* per abstraure's de la complexitat inherent a la capa de persistència de les dades. Generalment correspondrà al rol d'un desenvolupador de programari (programador, analista-programador, arquitecte, enginyer software, etc.)

- **InitAplicacio**: Aquest rol es correspon a l'encarregat d'iniciar la configuració del framework que utilitza l'API per a la gestió de la persistència de dades. En el cas d'aplicacions web, correspondrà al Listener que arrencarà la configuració de l'aplicació web. En el cas dels tests, aquest rol l'adoptarà el mètode *main* que els executi.

- **DBA**: En alguns entorns (bàsicament de producció) els desenvolupadors no tenen accés al model de dades, i per tant, l'encarregat de la gestió de la base de dades la realitzarà el rol específic d'administrador de la base de dades (DBA). En els entorns de desenvolupament, aquest rol el pot realitzar el mateix rol d'usuari.

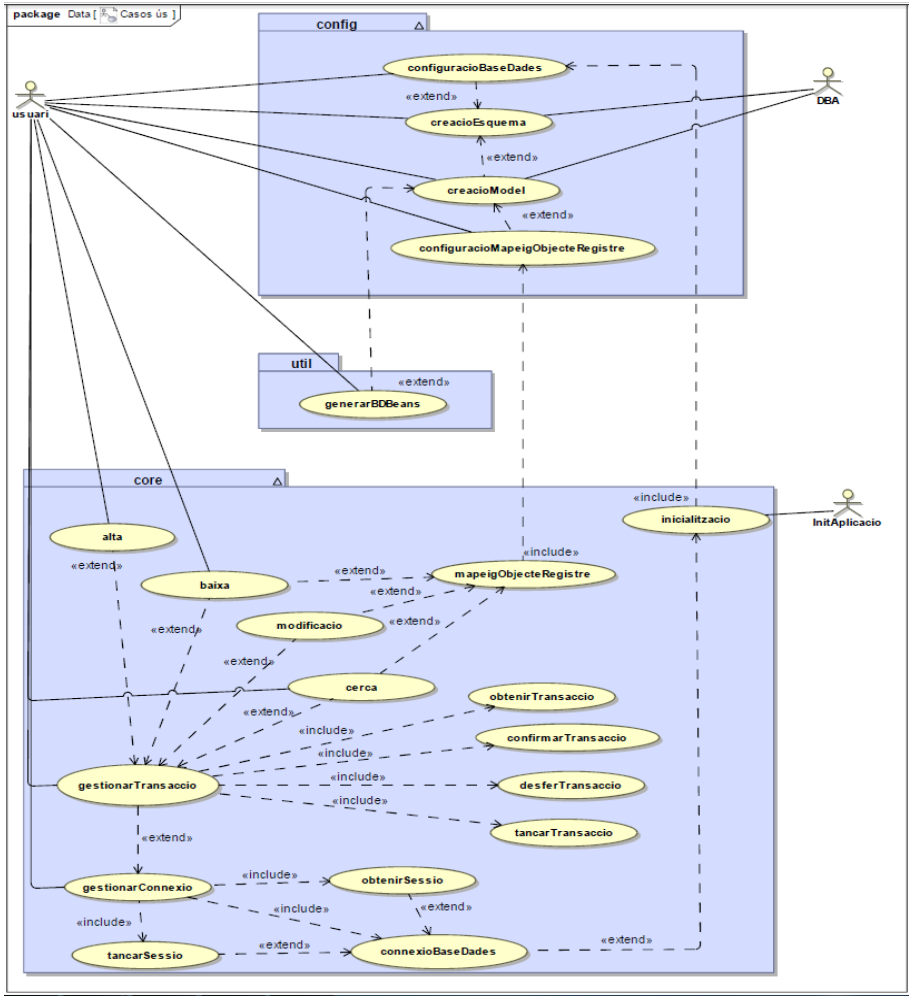
D'altra banda, s'ha decidit agrupar els diferents casos d'ús en tres paquets en relació a la base de la funcionalitat que descriuen:

- **config**: Correspon a la funcionalitat de configuració del framework. Aquesta configuració estarà diferenciada en dos nivells diferents, i per tant, dos casos d'ús. En primer lloc, es disposarà de la configuració de la base de dades, és a dir, indicar al sistema quina serà la base de dades on es farà la persistència (url, usuari, contrasenya, etc.). Finalment, es disposarà d'un segon cas d'ús en el que l'usuari especificarà els *mappings* que existiran entre els beans d'accés a les dades (POJO's) i les taules de la base de dades. Aquesta configuració es realitzarà mitjançant el mecanisme d'*Annotations* que ofereix la plataforma Java a partir de la versió 1.5.

- **core:** En aquest paquet s'agrupen els casos d'ús del funcionament del nucli del *framework*. Així doncs, es disposarà dels casos d'ús que serviran a l'usuari per a poder "interactuar" amb la base de dades -API- i que permetrà realitzar les operacions d'alta, baixa, modificació i cerca, així com la gestió d'aquestes operacions (gestió de transaccions i sessió). Els 4 primers casos d'ús es recolzen en el cas d'ús que fa el *mapeig* Objecte-Registre entre la classe JAVA i la taula de la base de dades i que permetrà poder explotar la base de dades. D'altra banda, el cas d'ús de la gestió de les transaccions inclou tres casos més que corresponen a les accions que es poden realitzar amb les transaccions: obtenir una nova (per a que l'usuari pugui realitzar diferents operacions de forma atòmica, per exemple), confirmar els canvis que s'han realitzat en la transaccions i, per tant, fer-los "visibles" a la base de dades, o desfer-los (perquè s'ha produït algun problema durant la transacció). També és important remarcar que aquest cas d'ús també estenen el cas d'ús d'obtenció de la sessió, donat que aquesta és l'encarregada de la gestió de les transaccions. Finalment, cal destacar que el cas d'ús de gestió de la sessió inclourà la configuració de la base de dades donat que necessita d'aquestes dades per a poder connectar-se amb el SGBD.

- **util**: En aquest paquet s'inclouran les eines que faciliten a l'usuari la interconnexió amb la base de dades. Bàsicament es disposarà de la funcionalitat per a poder generar dinàmicament els beans (POJO's) d'accés al model de dades en funció de la base de dades que estigui configurada. D'aquesta manera, l'usuari es podrà centrar en el "negoci" de la seva aplicació "oblidant-se" del *mapping* d'SQL a JAVA.

Així doncs, el diagrama de casos d'ús del sistema és el següent:



A continuació s'especificarà cadascun dels casos d'un agrupats pel paquet al que pertanyen.

6.2.1. config

Nom	***ConfiguracioBaseDades***
Funcionalitat	Configura les dades de connexió que utilitzarà el *framework* per a comunicar-se amb el sistema gestor de base de dades.
Flux principal	1. L'usuari crea el fitxer de configuració *(uoc-bd-config.properties)* a la ubicació corresponent . 2. L'usuari omple les dades de connexió en la propietat corresponent. 2.1. Introdueix la URL de la base de dades 2.2. Introdueix l'usuari de la base de dades on es disposarà del model. 2.3. Introdueix la contrasenya per a poder accedir.

Nom	***ConfiguracioMapeigObjecteRegistre***
Funcionalitat	Configurar el mapping entre els beans de base de dades (POJO's) que s'utilitzaran a l'aplicació i les taules disponibles al model.
Flux principal	1. Usuari crea un POJO que representa una taula del model 1.1. Especifica la taula a la que referencia mitjançant l'annotacio @Taula 1.2. Crea un atribut a l'objecte per a especificar una columna de la taula a la que referencia mitjançant l'annotació @Columna 1.3. Es repeteix el pas 1.2 per a cadascuna de les columnes de la taula 2. Es repeteix el pas 1 per a cadascuna de les taules del model que es vol utilitzar a l'aplicació.

Nom	***CreacioEsquema***
Funcionalitat	Crear l'usuari sobre el que es crearà el model de dades.
Flux principal	1. El DBA crea l'usuari de base de dades al *tablespace* corresponent amb una contrasenya d'accés.

Nom	***CreacioModel***
Funcionalitat	Crear el model al motor gestor de base de dades.
Flux principal	1. El DBA accedeix a l'esquema amb les credencials (usuari+contrasenya) en el que es crearà el model. 2. El DBA executa l'script de creació del model.
Flux alternatiu	1. Les credencials no són correctes, es genera un error d'accés. 2. L'*script* no es consistent (les referències no existeixen, els tipus especificats són incorrectes, etc.), es genera un error.

6.2.2. core

Nom	***Alta***
Funcionalitat	Afegir un registre a una de les taules del model de dades.
Flux principal	1. Crear una nova instancia del POJO que representa la taula a la que es volen afegir dades. 2. Afegir el contingut dels atributs de la instancia que es vol afegir a la taula. 3. Executar el mètode *create* disponible a l'objecte Sessio passant per paràmetre el POJO instanciat. 4. Obtenir el resultat de l'alta (identificador en cas de que la clau primària sigui un auto-numèric, per exemple).
Flux alternatiu	3. En cas de voler gestionar la transaccionalitat, requerirà obtenir de la sessió una nova transacció especificant el *setAutocommit* a fals. Si no s'especifica res, el sistema executarà la operació de manera atòmica obtenint la transacció i tancant-la al finalitzar la operació. 4. En cas d'error, es generarà una excepció indicant el motiu de l'error.

Nom	***Baixa***
Funcionalitat	Esborrar registres existents a una de les taules del model de dades.
Flux principal	1. Crear una nova instancia del POJO que representa la taula de la que es volen eliminar dades. 2. Obtenir la referència del registre que es vol eliminar. 3. Executar el mètode *delete* disponible a l'objecte Sessio passant per paràmetre el POJO instanciat.. 4. Obtenir el resultat de la baixa.
Flux Alternatiu	3. En cas de voler gestionar la transaccionalitat, requerirà obtenir de la sessió una nova transacció especificant el *setAutocommit* a fals. Si no s'especifica res, el sistema executarà la operació de manera atòmica obtenint la transacció i tancant-la al finalitzar la operació. 4. En cas d'error, es generarà una excepció indicant el motiu de l'error.

Nom	***Modificacio***
Funcionalitat	Modificar els valors d'un registre existent a una de les taules del model de dades.
Flux principal	1. Crear una nova instancia del POJO que representa la taula a la que es volen modificar dades. 2. Obtenir la referència del registre que es vol modificar. 3. Afegir el nou contingut dels atributs de la instancia que es vol modificar. 4. Executar el mètode *update* disponible a l'objecte Sessio passant per paràmetre el POJO instanciat. 5. Obtenir el resultat de la modificació.
Flux Alternatiu	4. En cas de voler gestionar la *transaccionalitat*, requerirà obtenir de la sessió una nova transacció especificant el *setAutocommit* a fals. Si no s'especifica, el sistema executarà la operació de manera atòmica obtenint la transacció i tancant-la al finalitzar la operació. 5. En cas d'error, es generarà una excepció indicant el motiu de l'error.

Nom	***Cerca***
Funcionalitat	Obtenir una sèries de registres que compleixin amb unes determinades condicions de cerca.
Flux principal	1. Crear una nova instancia del POJO que representa la taula (o la cerca). 2. Obtenir una nova sessió a través de l'objecte Sessio. 3. Executar el mètode *find* disponible a l'objecte Sessio passant per paràmetre el POJO instanciat. 4. Obtenir els resultats de la cerca.
Flux Alternatiu	3. Si el POJO que es passa per paràmetre no és un bean de base de dades (és a dir, està definit com a una entitat -@Entitat- o com un mapejador -@Mapper- es genera un error. 4. En cas d'error, es generarà una excepció indicant el motiu de l'error.

Nom	***MapeigObjecteRegistre***
Funcionalitat	Realitzar la traducció del codi JAVA introduït pel desenvolupador al codi SQL que interpreta la base de dades.
Flux principal	1. L'usuari realitza una operació (CRUD). 2. El sistema recupera la configuració de la base de dades per a poder realitzar el *mapping* en funció del motor de base de dades que estigui configurat. 3. El sistema accedeix a la taula gràcies als mappings dels objectes implicats en la operació. 4. El sistema tradueix la operació en el codi SQL del motor de base de dades configurat. 5. El sistema executa la *query*. 6. El sistema recupera les dades dels registres i les transforma a POJO's. 7. El sistema retorna les dades.

Flux Alternatiu	2. Error en la recuperació de la configuració: no existeix el fitxer, sistema gestor no es compatible, etc. Es genera una excepció amb l'error específic. 4. Error en la traducció del codi JAVA a SQL . Es genera una excepció detallant l'error. 5. Error en l'execució de la *query:* configuració de la taula, incompatibilitat de tipus, etc. Es detalla l'error en la descripció de la excepció. 6. Error en la traducció del codi SQL a JAVA. Es genera una excepció detallant l'error.

Nom	***GestioTransaccio***
Funcionalitat	Gestionar l'atomicitat de les operacions de base de dades en els casos que es requereixi. Inclou els casos d'ús d'obtenció, confirmació, desfeta i tancament de la transacció.
Flux principal	1. L'usuari sol·licita una nova transacció mitjançant l'objecte *Sessio (cas d'us ObtenirTransaccio)*. 2. Un cop realitza les operacions necessàries, les confirma (cas d'us *ConfirmarTransaccio*). 3. L'usuari tanca la transacció (cas d'us *TancarTransaccio*).
Flux Alternatiu	2. En cas d'error o comportament anormal, es desfan els canvis (cas d'us *DesferTransaccio*).

Nom	***ObtenirTransaccio***
Funcionalitat	Obtenir una nova transacció per a poder realitzar una sèrie d'operacions de manera atòmica.
Flux principal	1. L'usuari sol·licita una nova transacció mitjançant l'objecte *Sessio* especificant el *setAutocommit* a fals. 2. El sistema accedeix a la base de dades i crea una nova sessió. 3. El sistema retorna la nova sessió creada.
Flux Alternatiu	1. En cas de no disposar d'una sessió activa, crea una de nova. 2. En cas d'error es genera una excepció.

Nom	***ConfirmarTransaccio***
Funcionalitat	Fer efectives totes les modificacions realitzades en una transacció.
Flux principal	1. L'usuari recupera una transacció mitjançant l'objecte *Sessio*. 2. L'usuari confirma les modificacions realitzades mitjançant l'objecte SessioFactory .
Flux Alternatiu	2. En cas d'error es genera una excepció.

Nom	***DesferTransaccio***
Funcionalitat	Desfer totes les modificacions realitzades en la transacció i deixar l'estat de la base de dades tal i com estava abans de realitzar les modificacions en la transacció.
Flux principal	1. L'usuari recupera una transacció mitjançant l'objecte *Sessio.* 2. L'usuari desfà els canvis realitzats en la transacció realitzades mitjançant l'objecte SessionManager .
Flux Alternatiu	2. En cas d'error o comportament anormal, es genera una excepció amb la descripció detallada.

Nom	***TancarTransaccio***
Funcionalitat	Tancar la transacció creada.
Flux principal	1. L'usuari recupera una transacció (cas d'us *ObtenirTransaccio*). 2. L'usuari tanca la transacció.
Flux Alternatiu	2. En cas d'error o comportament anormal, es genera una excepció amb la descripció detallada.

Nom	***GestioSessio***
Funcionalitat	Gestió de les connexions "clàssiques" a la base de dades. Inclou els casos d'ús d'*ObtenirSessio*, *TancarSessio* i *connexioBaseDades*.
Flux principal	1. L'usuari obté una nova sessió (cas d'us *ObtenirSessio*). 2. Realitza les operacions necessàries. 3. Finalment, tanca la sessió (cas d'us *TancarSessio*).
Flux Alternatiu	1. En cas d'error, es genera la excepció corresponent. 2. En cas d'error, es traça l'error.

Nom	***ObtenirSessio***
Funcionalitat	Obtenir una nova sessió.
Flux principal	1. L'usuari sol·licita una nova sessió a través del mètode *newSessio*() de la classe SessioFactory. 2. El sistema es connecta a la base de dades. 3. El sistema retorna una nova sessió.
Flux Alternatiu	2. En cas d'error, es genera la excepció corresponent. 3. En cas d'error, es genera una excepció amb la descripció corresponent.

Nom	***TancarSessio***
Funcionalitat	Obtenir una nova sessió.
Flux principal	1. L'usuari recupera una sessió (cas d'us *ObtenirSessio*). 2. L'usuari tanca la sessió.
Flux Alternatiu	1. En cas d'error, es genera la excepció corresponent. 2. En cas d'error, es traça l'error.

Nom	***ConexioBaseDades***
Funcionalitat	Connectar amb la base de dades.
Flux principal	1. El sistema obté les dades del fitxer de configuració. 2. El sistema carrega la configuració (driver) i realitza la comunicació amb el motor mitjançant JDBC.
Flux Alternatiu	1. En cas d'error, es genera l'excepció corresponent (fitxer no existent o dades del fitxer incorrectes). S'atura l'execució. 2. En cas d'error (credencials incorrectes, URL sense accés, etc.) es genera una excepció amb la descripció de l'error.

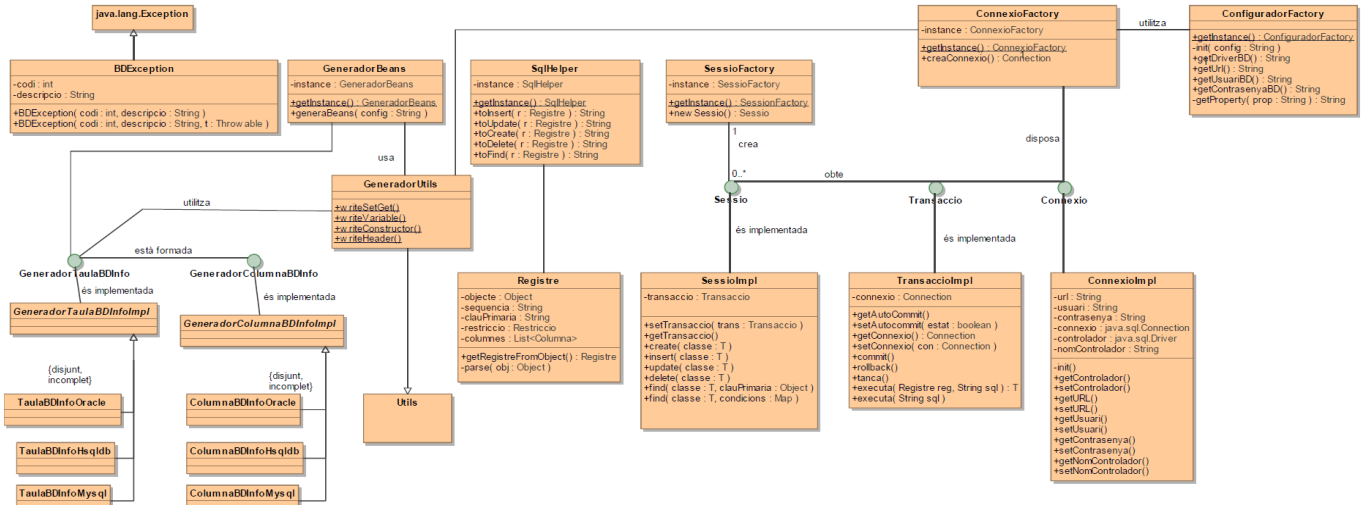
Nom	***Inicialització***
Funcionalitat	Inicialitzar el *framework* amb les dades de configuració per a poder obtenir el *datasource* associat al pool de connexió.
Flux principal	1. L'aplicació client carrega les dades de configuració i executa el mètode d'inicialització del framework. 2. El sistema connecta amb la base de dades (cas d'us *ConnexioBaseDades*). 3. El sistema inicialitza totes les estructures de dades del framework. 4. Retorna el *datasource*.
Flux Alternatiu	2. En cas d'error, es genera l'excepció i s'atura l'execució. 3. En cas d'error, es genera l'excepció i s'atura l'execució.

6.2.3. util

Nom	***GenerarBDBeans***
Funcionalitat	Generar les classes JAVA relacionades amb les taules de l'esquema de base de dades de la configuració. D'aquesta manera, es dona la opció a l'usuari de no realitzar la ferragosa tasca de relacionar el model de dades amb una classe JAVA (taules, tipus de dades de les columnes, restriccions d'integritat, claus foranes, etc.) en funció del sistema motor de base de dades que estigui configurat. Aquest cas generarà els beans que es fa referència al cas d'ús ConfiguracioMapeigObjecteRegistre del paquet *config*.
Flux principal	1. L'usuari especifica la configuració on es generaran les classes JAVA. 2. L'usuari sol·licita la generació dels *beans* (POJOs) que relacionaran les taules i columnes amb la classe JAVA. 3. El sistema connecta amb la base de dades (cas d'us *ConnexioBaseDades*). 4. El sistema genera les classes JAVA relacionades amb cadascuna de les taules del model en la ubicació especificada per l'usuari.
Flux Alternatiu	3. En cas d'error, es genera l'excepció i s'atura l'execució. 4. En cas d'error, es genera una excepció amb la descripció de l'error i s'atura l'execució.

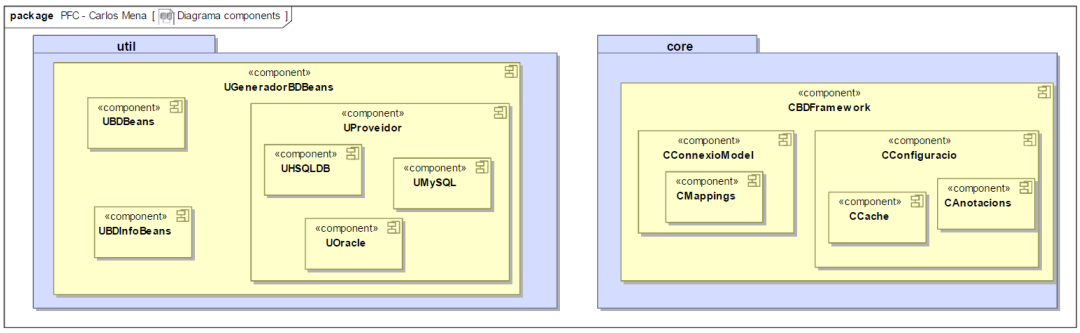
6.3. Diagrama de classes

A continuació, es mostra el diagrama amb les classes i mètodes més rellevants del disseny en funció dels patrons i principis descrits al primer apartat:



6.4. Diagrama de components

En aquest apartat es realitzarà el detall del diagrama de components de la solució implementada. S'ha realitzat un disseny *top-down.* Així doncs, els components del sistema sense refinar són els següents :



Cadascun dels paquets disposa de la seva responsabilitat: *util* per a donar eines d'ajuda a l'usuari; i, *core* que correspon al negoci del propi *framework*.

A continuació, es refinaran els diagrames acabats de presentar.

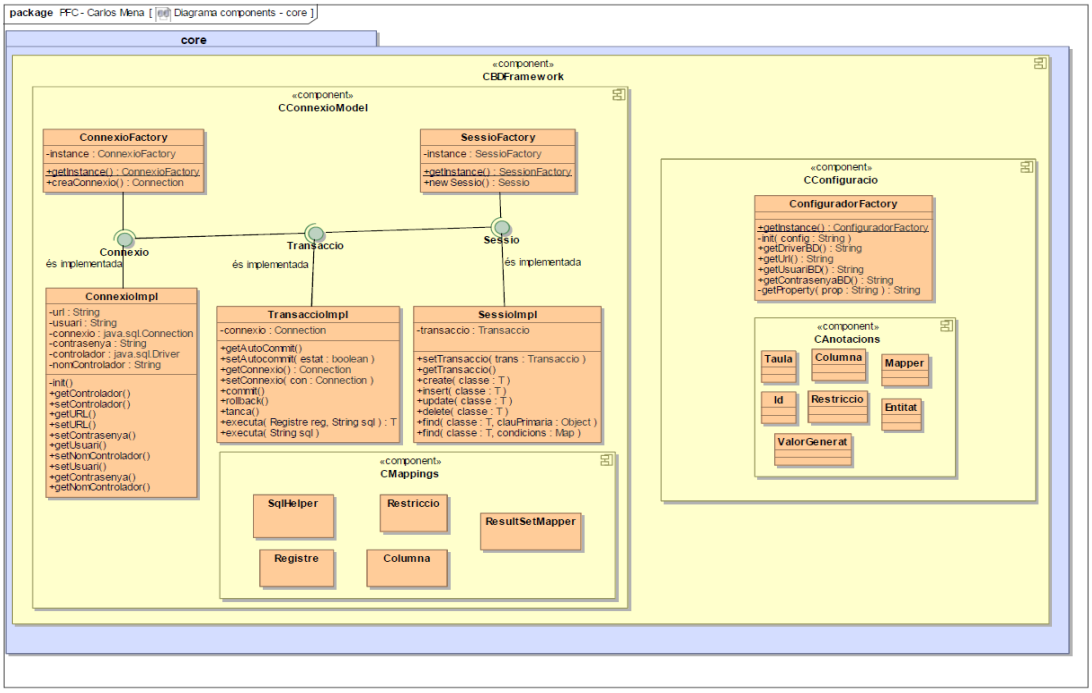
6.4.1. core

Com es pot observar en l'anterior diagrama, el framework està format per dos subcomponents: CConfiguració i CConnexioModel en funció de les responsabilitats de cada component.

CConfiguració és el component encarregat de la configuració del framework, és a dir, disposarà de les estructures de dades necessàries per a la configuració del framework. En aquest sentit, disposarà d'un subcomponent, CAnnotacions, en el que s'han creat Annotations Custom de la versió 1.5 de Java per a poder crear els mappings de beans a SQL. Finalment disposa de la classe estàtica encarregada de la lectura de les dades del fitxer de configuració.

CConnexioModel, és el codi encarregat d'abstraure al programador de la complexitat de la persistència. Així doncs, la seva responsabilitat serà la d'accedir al model de dades i fer les lectures/escriptures corresponents. En aquest component resultaran fonamentals el tractament de la *Sessio* i de la *Connexio* mitjançant el principi IOC. D'altra banda, també es disposarà d'un component, *CMappings*, encarregat del mapping entre la base de dades i el codi Java.

A continuació, es mostra el diagrama refinat:



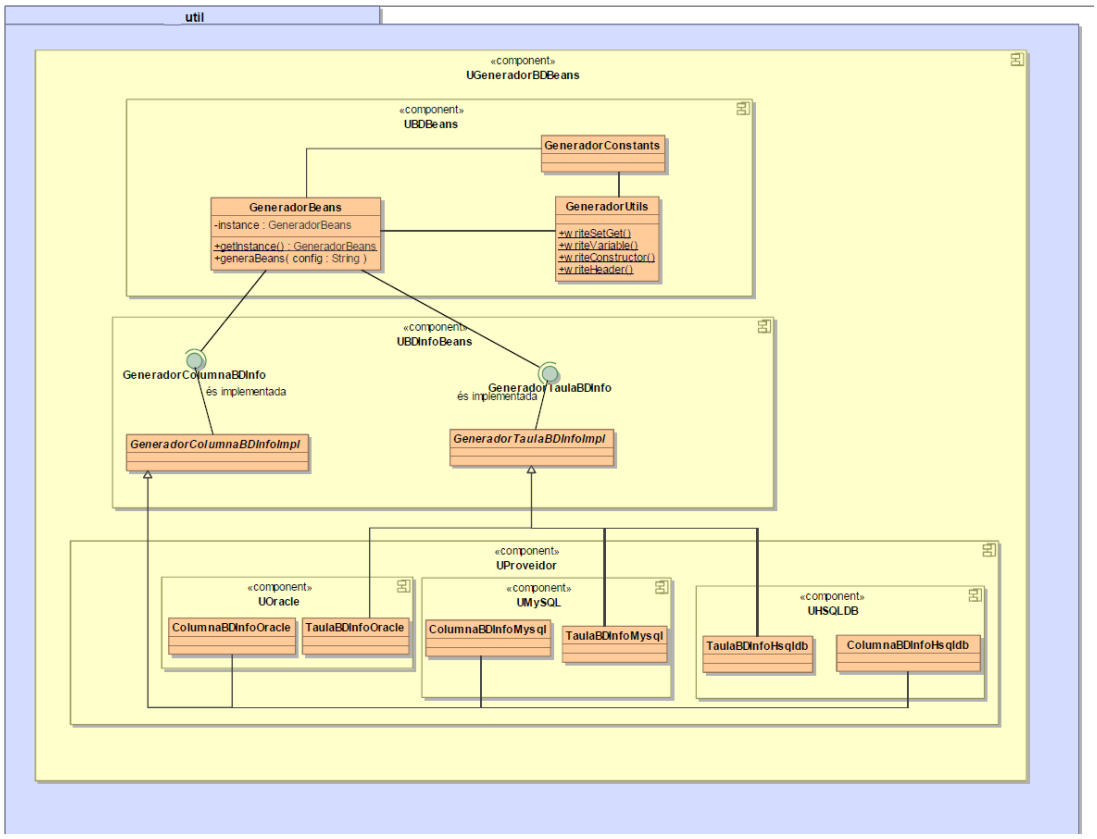
6.4.2. util

El component d'utilitats està format per tres subcomponents : UBDBeans, UBDInfoBeans i UProveidor.

UBDBeans, és el component que proporcionarà a l'usuari el mètode necessari per a poder crear els beans en funció del model de dades. Per a poder realitzar-ho, utilitzarà el mòdul de configuració del core que li permetrà accedir al model i obtenir les metadades de cadascuna de les taules i columnes del motor. Aquest mètode haurà de crear POJO's als directori especificat a la configuració, i és per això, que requerirà mètodes de creació de classes. Aquests mètodes estaran centralitzats en la classe *GeneradorUtils* (principi DRY) que incorporarà totes les utilitats necessàries per a que l'eina pugui generar els beans.

UBDInfoBeans, correspon al component que incorpora la informació de la taula i de les columnes de la base de dades. S'ha seguit el principi OCP per a que al Generador li resulti transparent la implementació del motor de base de dades. Així es disposa d'una interfície comuna per a la informació de la taula -*GeneradorTaulaBDInfo*- i de les columnes -*GeneradorColumnaBDInfo*- i d'una implementació abstracta amb els mètodes comuns per a totes les implementacions del component *UProveidor* que es detalla a continuació.

UProveidor, és el component encarregat de la implementació específica del model. Aquest model està format per N subcomponents, un per a cadascuna de les implementacions amb les que el framework és compatible. Inicialment, ho serà per MySQL, Oracle i HSQLDB. Així doncs, aquest components disposaren de dues classes (una per taula i una altra per columna) que extenen de les classes abstractes definides al component *UBDInfoBeans* (*GeneradorTaulaBDInfoImpl* i *GeneradorColumnaBDInfoImpl,* respectivament). Així doncs, aquestes classes pròpies de cada proveïdor incorporaran les dades específiques del motor en qüestió seguint les especificacions de les interfícies definides al component *UBDInfoBeans*. Aquesta solució segueix el principi OCP i permet afegir més funcionalitat (nous motors de bases de dades) sense impactar en el codi existent.



7. DISSENY DE L'APLICACIÓ D'EXEMPLE D'ÚS DEL *FRAMEWORK*

En aquest capítol es realitza una descripció de l'anàlisi de l'aplicació d'exemple que utilitzarà el *framework* de persistència dissenyat i implementat descrit al capítol anterior. Com aquesta aplicació és "lliure", en primer lloc caldrà especificar els requisits, objectius i la descripció del que es pretén aconseguir mitjançant la seva implementació.

L'aplicació d'exemple correspondrà a una aplicació web clàssica que es desplegarà en un servidor d'aplicacions i que realitzarà les operacions bàsiques d'accés a la base de dades amb l'objectiu de mostrar les avantatge que aporta utilitzar el *framework* de persistència desenvolupat. Així doncs, per a poder instal·lar l'aplicació web es disposarà dels següents requisits:

- Servidor de bases de dades correctament instal·lat i arrencat. Per a les proves que es realitzaran en aquest PFC caldrà disposar de d'un motors per a poder validar el correcte comportament del framework desenvolupat. En el nostre cas, per a les proves s'utilitzarà *MySQL* i *HSQLBD* o Oracle.

- Script SQL de base de dades amb el model.

- Servidor d'aplicacions JAVA (amb versió 1.5 com a mínim) en el que desplegar l'aplicació web. En el nostre cas s'utilitzarà *Tomcat*.

- WAR amb la distribució (o en el seu defecte les fonts necessàries per a poder generar el distribuïble).

En relació a l'aplicació web, suposarem que una agència de viatges del barri i, a conseqüència de l'elevat ús d'Internet entre el jovent (principals clients del seu negoci) i de la forta competència en el sector, ens ha encarregat desenvolupar una aplicació web per a cercar "txollos". En aquest sentit, es tractarà d'una aplicació molt simple que constarà de dues parts:

- **Part pública**: mostrarà un llistat amb les darreres ofertes. Disposarà d'un filtre que permeti afinar (preu, destinació, etc.) la "cerca del txollo". També permetrà accedir al detall de les ofertes disponibles. Aquesta pantalla serà accedida bàsicament pels clients que vulguin contractar els viatges. Amb aquesta part es podrà "testejar" les cerques i les correspondències entre les taules del model (claus foranes, restriccions d'integritat, etc.).

- **Part privada**: l'aplicació, previ login al sistema, disposarà de pantalles d'administració de les ofertes de la web i que permetrà, entre d'altres, donar d'alta noves ofertes, eliminar-ne les caducades i modificar els continguts de les ofertes vigents. Aquesta part, permetrà validar les operacions CRUD.

Per a simplificar la complexitat, es suposarà que l'aplicació només permetrà reservar les places pel "txollo". Així doncs, els clients que estiguin interessats en adquirir la oferta hauran de confirmar la plaça reservada mitjançant el pagament de l'import a través d'una transferència bancària al número de compte de l'agència o bé desplaçant-se a la oficina indicant el codi de la reserva obtingut. Finalment, es suposarà que les reserves es mantenen durant 3 dies i que l'agència no suporta "overbooking", és a dir, un cop es reservi una plaça aquesta desapareixerà de les disponibles.

Un cop coneguda amb més detall l'aplicació d'exemple, i de la mateixa manera que s'ha realitzat en el cas del *framework*, es mostrarà el disseny de la solució amb els diagrames de casos d'ús, de classes i de components.

7.1. Introducció

L'estil arquitectònic que es proposarà per l'aplicació web de l'agencia de viatges correspondrà a una arquitectura heterogènia: arquitectura client-servidor organitzada en capes, i on cadascuna d'aquestes capes, s'estructurarà seguint una arquitectura orientada a objectes distribuïts.

Amb l'arquitectura client-servidor, s'aconseguirà un sistema fàcilment escalable, que a més, i a conseqüència d'estar organitzada en capes – presentació (vista), negoci (controlador) i dades (model) -, serà poc acoblat, i fàcilment mantenible (cadascuna de les capes s'encarrega d'unes tasques especifiques).

En funció de les visites que disposi l'agència es podrien afegir varies bases de dades, i per tant, es crearia una quarta capa (servidor d'integració), que s'encarregaria de recollir les dades distribuïdes i presentar-les a l'aplicació com si estiguessin disponibles en una sola base de dades. Aquest fet, també incrementaria la tolerància a les fallides, donat que en cas de fallida d'una de les base de dades el servidor d'integració podria localitzar les dades en una altre, d'una manera transparent per l'usuari i, d'aquesta manera, es reduiran les caigudes del sistema en "pics" de connexió massiva de clients (sobretot quan acaben de sortir les ofertes o s'acaba el període de "vigència"), fet que podria suposar la pèrdua dels clients.

Finalment, i tenint en compte que cadascuna d'aquestes capes seguirà una arquitectura orientada a objectes distribuïts, el sistema implementat serà obert (permetrà afegir nous recursos, en cas que faci falta), molt flexible, (es podran crear diferents instàncies del mateix servei per fer front a sobrecàrregues del sistema) i on serà possible re-configurar el sistema dinàmicament segons les necessitats.

7.2. Diagrama de casos d'ús

Aquesta aplicació web disposarà dels següents actors:

- **Administrador**: Rol encarregat de la gestió de les ofertes que apareixen a la web.

- **Client**: Usuari d'Internet al que està enfocada la web amb la intenció de revifar el negoci. Serà qui realitzi les cerques d'ofertes i realitzari, en cas que estigui interessat, la reserva de les ofertes disponibles.

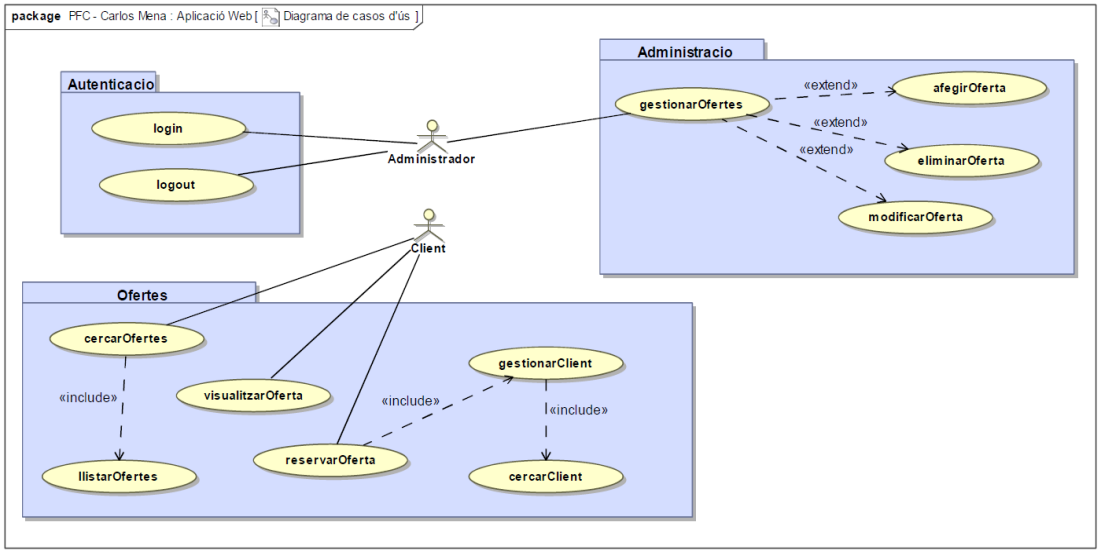
D'altra banda, i de la mateixa manera que s'ha realitzat al framework, s'ha decidit agrupar els casos d'ús en diferents paquets en relació a la base de la funcionalitat que descriuen. Així doncs disposarem dels següents paquets:

- **Autenticació**: Agrupa els casos d'ús relacionats amb la funcionalitat d'autenticació al sistema. Per tant, es disposarà de dos casos d'ús: login i logout.

- **Administració**: Es una funcionalitat exclusiva de l'administrador del sistema i es correspon a les altes, baixes i modificacions d'ofertes. Així doncs, s'han definit aquests 3 casos d'ús *afegirOferta*, *eliminarOferta* i *modificarOferta*, que seran els que facin ús *gestionarOfertes*.

- **Ofertes**: En aquest darrer paquet s'agrupen els casos d'ús que estan relacionats amb les ofertes. Així doncs, es disposarà dels casos d'ús de cerca, visualització d'una oferta i la reserva de la mateixa. Cal destacar que la cerca inclourà també el cas d'ús de llistat d'ofertes.

Així doncs, el diagrama de casos d'ús serà el següent:



7.2.1. Autenticació

Nom	***login***
Funcionalitat	Accedir a la part privada del portal per a gestionar les ofertes.
Flux principal	1. L'administrador accedeix al portal i indica les seves credencials. 2. El sistema mostra la pàgina d'inici de la gestió d'ofertes.
Flux alternatiu	2. En cas d'error es mostrarà un missatge d'error especificant el motiu (usuari desconegut o contrasenya incorrecte).

Nom	***logout***
Funcionalitat	Abandonar la part privada del portal.
Flux principal	1. L'administrador selecciona el botó de sortir de la part privada del portal. 2. El sistema redirigeix a la part pública de la web.

7.2.2. Administració

Nom	***afegirOferta***
Funcionalitat	Incorporar una nova oferta al portal.
Flux principal	1. L'administrador accedeix al portal i indica les seves credencials. 2. L'administrador selecciona la opció d'afegir nova oferta. 3. El sistema mostra un formulari amb les dades que cal omplir. 4. L'administrador omple les dades amb la informació de la nova oferta i selecciona el botó d'Afegir. 5. El sistema mostra un missatge indicant la correcta execució de la operació.
Flux alternatiu	5. En cas d'error es mostrarà un missatge d'error especificant el motiu.

Nom	***eliminarOferta***
Funcionalitat	Esborrar una oferta disponible al portal.
Flux principal	1. L'administrador accedeix al portal i indica les seves credencials. 2. L'administrador selecciona la opció d'esborrar una oferta. 3. El sistema mostra un llistat amb les ofertes disponibles. 4. L'administrador selecciona una oferta i clica el boto d'Esborrar. 5. El sistema mostra un missatge de confirmació de la operació ("Esta segur d'esborrar la oferta?"). 5. El sistema mostra un missatge indicant la correcta execució de la operació.
Flux alternatiu	5. Si l'usuari cancel·la la operació es redirigeix a la pàgina principal amb el llistat de les ofertes. 6. En cas d'error es mostrarà un missatge d'error especificant el motiu.

Nom	***modificarOferta***
Funcionalitat	Modificar les dades d'una de les ofertes disponibles al portal.
Flux principal	1. L'administrador accedeix al portal i indica les seves credencials. 2. L'administrador selecciona la opció de modificar una oferta. 3. El sistema mostra un llistat amb les ofertes disponibles. 4. L'administrador selecciona una oferta i clica el botó Modificar. 5. El sistema mostra un missatge indicant la correcta execució de la operació.
Flux alternatiu	5. En cas d'error es mostrarà un missatge d'error especificant el motiu.

7.2.3. Ofertes

Nom	***cercarOfertes***
Funcionalitat	Cercar alguna de les ofertes disponibles al portal (en funció d'uns determinats criteris de cerca).
Flux principal	1. L'usuari especifica mitjançant el filtre de la pàgina els criteris de cerca que consideri oportuns. 2. El sistema mostra un pàgina amb totes les ofertes que compleixen els criteris especificats.
Flux alternatiu	2. En cas de no haver-hi cap resultat, es mostra un missatge descriptiu.

Nom	***visualitzarOferta***
Funcionalitat	Veure els detalls d'una de les ofertes del llistat.
Flux principal	1. L'usuari selecciona una de les ofertes del llistat 2. El sistema mostra una nova pantalla amb tots els detalls de la oferta.
Flux alternatiu	2. En cas d'error es mostrarà un missatge d'error especificant el motiu.

Nom	***reservarOferta***
Funcionalitat	Guardar les places especificades de la oferta seleccionada. Per a confirmar la oferta caldrà realitzar el pagament en un període màxim de 3 dies des de la data de la reserva.
Flux principal	1. L'usuari reserva una de les ofertes disponibles al portal. 2. El sistema mostra un missatge d'alerta indicant si son correctes les dades incorporades. 3. El sistema emmagatzema del dades de l'usuari a la base de dades. 4. El sistema mostra un "justificant" amb el codi de reserva.
Flux alternatiu	3. En cas que existeixi el client, associa la inscripció al client obtingut. 4. Si l'usuari cancel·la la operació, es torna al llistat d'ofertes.

Nom	***gestionarClient***
Funcionalitat	Donar d'alta un nou client a la base de dades o modificar les dades d'un existent.
Flux principal	1. El client facilita les seves dades per a poder confirmar la reserva. 2. El sistema obté les dades del client de la base de dades. 3. El sistema mostra un missatge amb el resultat de l'operació.
Flux alternatiu	2. a. En cas que l'usuari existeixi, es modifiquen les dades disponibles per les noves. b. En cas que l'usuari no existeixi, el crea amb les dades obtingudes.

Nom	***cercarClient***
Funcionalitat	El sistema fa una cerca a la base de dades per obtenir les dades d'un determinat client en funció d'un filtre especificat (generalment el nif).
Flux principal	1. El sistema obté les dades d'un client que vol reservar un viatge. 2. El sistema realitza la cerca de l'usuari a la taula d'usuari. 3. El sistema mostra les dades de l'usuari obtingudes.
Flux alternatiu	2. En cas que no existeixi el client especificat es generarà un error.

7.3. Diagrama de classes

En aquesta solució els patrons i principis SOLID que s'han utilitzat són els següents:

- Patró DAO (Data Acces Object): Amb aquest patró s'aconsegueix separar les responsabilitats de negoci i persistència, i per tant, aportarà flexibilitat a la solució. En aquest cas s'ha optat per a utilitzat un *GenericDAO* amb els mètodes comuns a tots els DAOs, i d'aquesta manera seguir el principi DRY per evitar la repetició de codi.

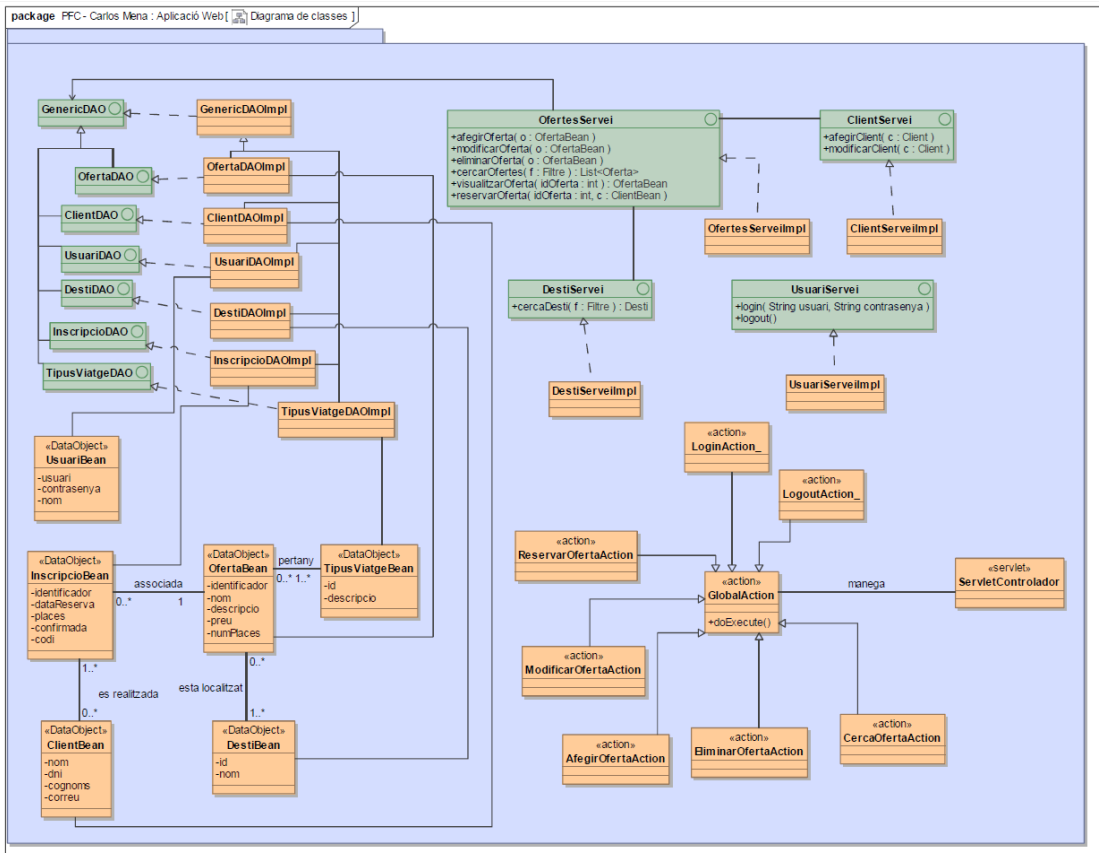
- ISP (Interface Segregation Principle): S'ha aplicat conjuntament amb el patró DAO per a la persistència de les dades.

- Patro Command: Aquest patró permet sol·licitar una operació a un objecte sense conèixer realment el contingut de la operació, ni el receptor real de la mateixa. Per fer-ho, s'encapsula la petició com un objecte amb el que facilita la parametrització dels mètodes.

- OCP (Open Close Principle): En aquest cas, s'ha utilitzat conjuntament amb el patró Command per a la gestió de les actions (*GlobalAction*).

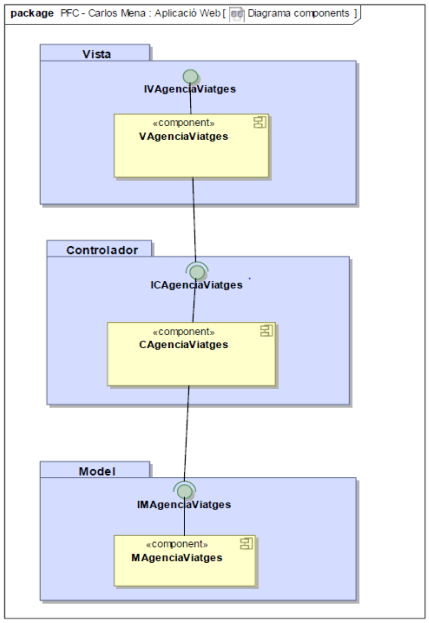
- IOC (Inversion Of Control): Per implementar aquest principi s'ha utilitzat el framework Spring que permet gestionar les necessitats d'inversió de control d'una manera senzilla sense haver de fer ús de patrons de disseny clàssics (Factory o AbstractFactory). En aquesta aplicació s'ha utilitzat per a la construcció de serveis d'accés als DAO's.

A continuació es mostra el diagrama de classes de l'aplicació web:



7.4. Diagrama de components

Per a la implementació de l'aplicació web també s'ha utilitzat un disseny *top-down*. El diagrama sense refinament és el següent:



A continuació es detalla cadascuna de les capes del sistema *AgenciaViatges*.

7.4.1. Vista

En la capa de presentació s'utilitzarà una estructura Model-2 seguint un patró de disseny MVC (Model – Vista – Controlador). Seguint aquest patró de disseny, i tenint en compte que el Model, es propi de la capa de lògica de negoci, podem dividir el component *VAgenciaViatges*, en dos components: el component que farà de controlador, *AgendaViatgesControlador* i el que farà de vista, *AgenciaViatgesVista*.

El controlador s'implementarà seguint el patró de disseny *Command*. Aquest patró enfatitza l'ús d'objectes com accions executables, és a dir, un objecte encapsula una simple acció (mètode) que implementa una interfície, el qual es invocat per un objecte Controlador. Amb aquest patró, s'aconsegueix tenir desacoblada la part de gestió (això ho farà mitjançant el *Controller*), de la de les operacions (es fa mitjançant *Actions*), el que permet poder afegir al sistema noves accions (funcionalitats) de forma més senzilla, i es per això que s'utilitzarà en el disseny del sistema.

S'utilitzarà el *framework Struts*, que implementa el patró *Command* pel model MVC amb el que el component *Controller* s'implementarà amb un servlet, les vistes s'implementaran amb JSP, mentre que les accions estaràn tipades com *actions*.

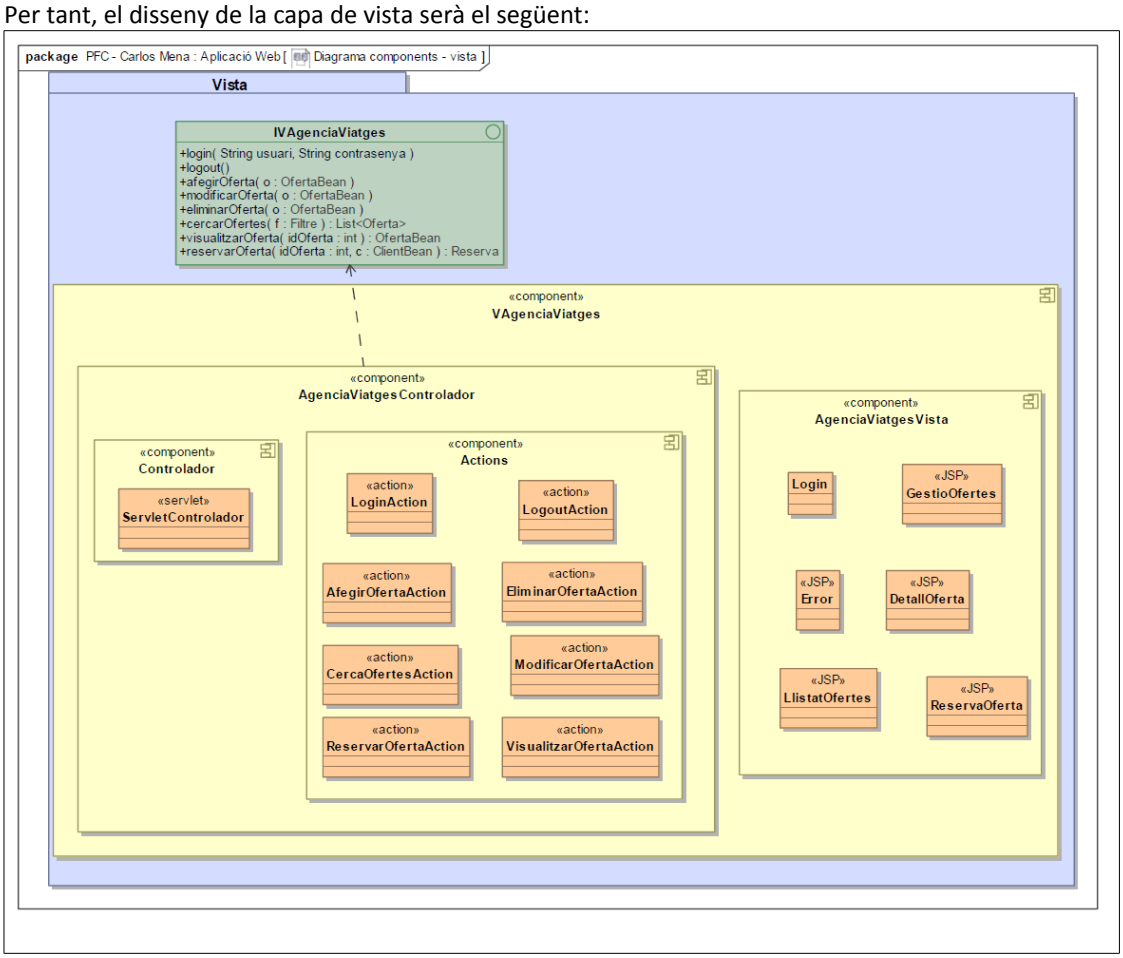
Així doncs, i tenint en compte les decisions anteriors, caldrà definir els següents objectes:

- Vistes
  - **Login**: Correspon a la vista amb el formulari d'accés a la part privada del portal (gestió d'ofertes)
  - **GestioOfertes**: Correspon a la home de la gestió d'ofertes mitjançant la qual es podran realitzar les altes, baixes i modificacions d'ofertes.
  - **DetallOferta**: Vista amb el detall de la oferta seleccionada. En cas de ser usuari administrador es permetrà modificar el contingut.
  - **LlistatOfertes**: Vista amb el llistat de les ofertes que compleixen un determinat criteri. Així doncs, disposarà d'un formulari per a poder filtrar-les.
  - **ReservaOferta**: Vista mitjançant la qual l'usuari podrà realitzar una reserva d'un determinat "txollo".

- **Error**: Vista d'error general en la que es mostraran els errors genèrics que puguin ocórrer durant la realització d'alguna de les operacions disponibles. Aquesta vista només es mostrarà en casos aliens i desconeguts. Altrament, errors controlats de negoci, cadascuna de les vistes disposarà de la seva pròpia gestió d'errors per a poder garantir la usabilitat de l'aplicació i facilitar a l'usuari la interacció amb la mateixa.

- Actions
  - **LoginAction**: Valida que existeixi l'administrador i que la contrasenya introduïda sigui la correcta.
  - **LogoutAction**: Redirigeix a l'administrador a la part pública del portal.
  - **AfegirOfertaAction**: Afegeix una nova oferta.
  - **ModificarOfertaAction**: Modifica els continguts d'una oferta del catàleg.
  - **EliminarOfertaAction**: Esborra una oferta del catàleg d'ofertes.
  - **CercaOfertesAction**: Llista les ofertes en funció d'un filtre.
  - **ReservarOfertaAcion**: Afegeix la reserva d'una oferta d'un determinat client.
  - **VisualitzarOfertaAction**: Recupera la informació d'una determinada oferta.

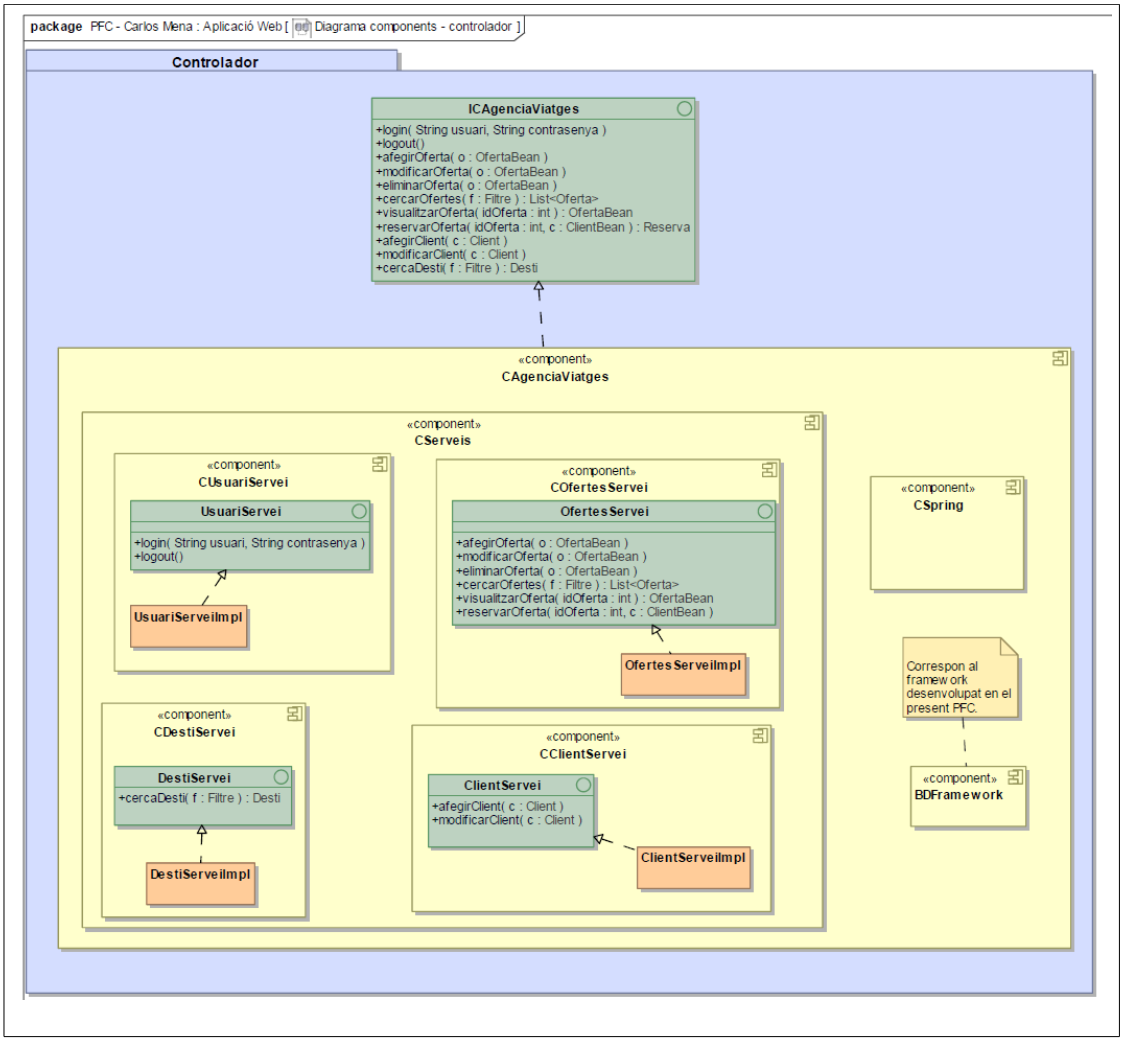
Per tant, el disseny de la capa de vista serà el següent:



7.4.2. Controlador

Per a la implementació d'aquesta capa s'utilitzarà el *framework* de persistència desenvolupat en aquest PFC. D'altra banda, també s'utilitzarà el principi IOC mitjançant Spring per disposar de diferents serveis agrupats per la funcionalitat dels mateixos i d'aquesta manera facilitar la modularitat i el manteniment de l'aplicació.

Així doncs, el diagrama de components per aquesta capa serà el següent:



7.4.3. Model

La capa del model estarà formada per 7 components: un per a cadascuna de les taules disponibles al model (Usuari, Client, Desti, TipusOferta, Ofertes, Inscripcio) i pel component DAO. En aquest cas, s'ha optat per a utilitzar un *GenericDAO* amb els mètodes comuns a tots els DAOs. Aquesta decisió aporta flexibilitat a la solució i evitar la repetició de codi -les operacions comuns estaran centralitzades en un únic punt- tal i com s'especifica al principi DRY.

El component DAO estarà format per una interfície amb la descripció dels mètodes (*GenericDAO*) i la classe que els implementa (*GenericDAOImpl*).

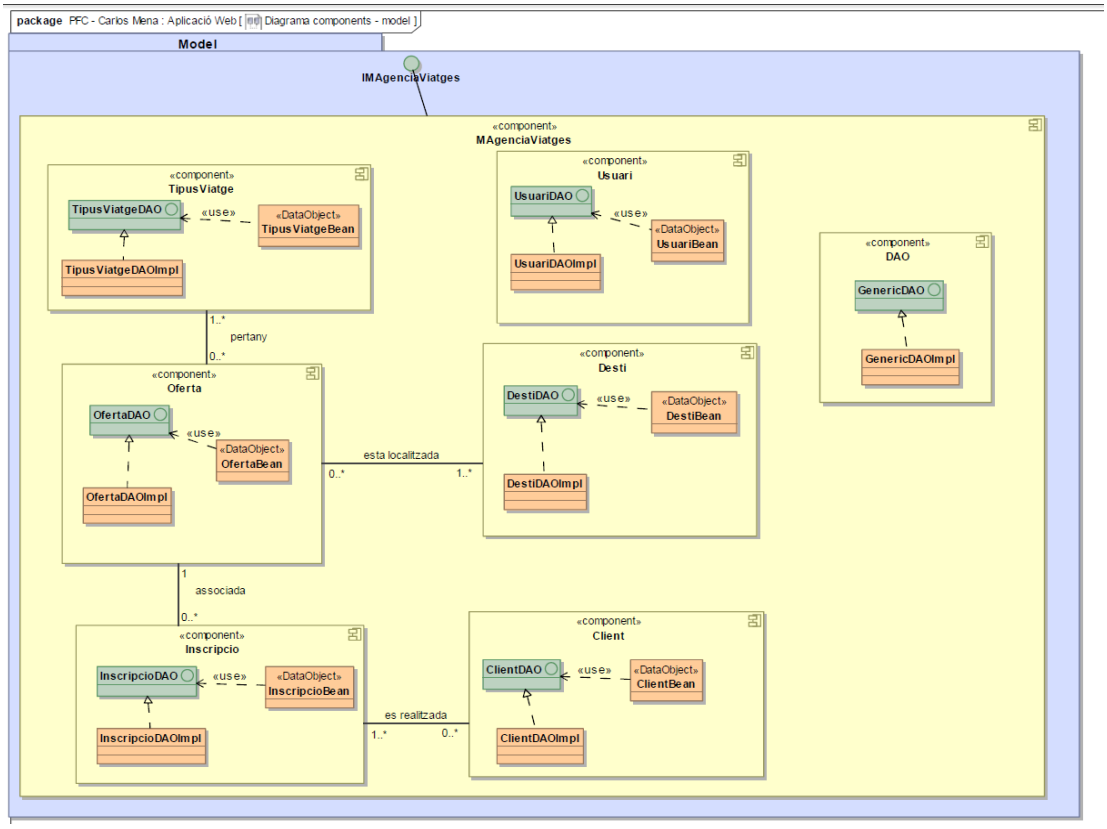
Per la seva banda, cadascun els components propis del model estan formats pels següents objectes:

- <Taula>DAO: Interfície que disposa dels mètodes necessaris per a la persistència de les dades. Aquesta interfície exten de GenericDAO.

- <Taula>DAOImpl: Implementació dels mètodes de la interficie. Només implementa aquells mètodes específics de la taula donat que exten de *GenericDAOImpl* que incorpora els mètodes comuns a tots els DAO's.

- <Taula>Bean: Correspon als beans de base de dades. Es un POJO clàssic amb el *mapping* de la taula a codi Java. En cas d'utilitzar l'eina del modul d'util (*GeneradorBeans*), aquests objectes es crearan de forma dinàmica en funció del model de dades.

Així doncs, el diagrama de components d'aquesta capa serà el següent:



8. CONCLUSIONS

En primer lloc, el fet d'haver dissenyat un *framework* de persistència, m'ha permès endinsar-me en el core d'algunes de les solucions existents al mercat (Apache cayenne, hibernate, etc.) i ampliar el coneixement que disposava abans de la realització del present PFC.

Per una banda, el desenvolupament m'ha permès reforçar els coneixements de les anotacions Java i fer ús més extens de l'API reflection de Java i dels genèrics tant útils per evitar la replicació de codi. El fet de crear el generador de beans, m'ha permès endinsar-me en l'API de JDBC per a poder generar dinàmicament les classes JAVA a partir de les metadades de la taula, fet que ha resultat molt enriquidor. També he pogut reforçar els coneixements d'Spring i del patró DAO, donat que en l'aplicació web s'ha fet un bon ús.

D'altra banda, els dos projectes s'han desenvolupat "desde zero", fet que m'ha servit per a reforçar tots els patrons de disseny estudiats a les assignatures d'enginyeria del software i en els principis SOLID de la programació orientada a objectes. En aquest sentit, i en la mesura de les possibilitats, s'ha desenvolupat un producte -el framework – i una aplicació web basats em aquests principis i que ha permès construir una solució senzilla, lleugera i fàcilment mantenible - que eren els requisits que s'havien plantejat a l'inici del projecte.

En relació a les ampliacions, i tal i com s'ha comentat en els objectius, el framework pretén ser una utilitat per a cobrir les operacions bàsiques per a la connexió amb una base de dades. En aquest sentit, les proves que s'han realitzat han estat amb taules amb poca informació i de baixa complexitat i és per això que no es pot saber la resposta del framework amb taules amb molta informació i complexa. Podria resultar interessant fer un estudi més acurat sobre el comportament del framework en aquest tipus d'escenari. Una altra millora que podria resultar interessant, seria la generació dels beans a partir de l'esquema de base de dades, donat que actualment només genera les dades mínimes. Finalment, es podria incloure algun mecanisme de cache per a millorar el rendiment quan fos necessari.

Finalment, podem concloure afirmant que gràcies a l'arquitectura de la solució proposada qualsevol modificació que es vulgui realitzar -tant al framework com a la web - es podrien realitzar d'una manera senzilla.

9. BIBLIOGRAFIA

[1] Álvarez Caules, Cecilio. "Arquitectura Java Sólida". 2012.
[2] Beck, Kent. "*Extreme Programming Explained".* Addison -Wesley, 1999.
[3] Camps, Rafael. "*Introducció a les bases de dades"*. Editorial: UOC.
[4] Costal, Dolors. "*Disseny de bases de dades".* Editorial: UOC.
[5] Costal, D. M.Ribera Sancho, Ernest Teniente. "*Enginyeria del Software Especificació"*. Edicions UPC, 2000.
[6] Crawford, W.; Kaplan, J. "J2EE Desgin Patters". O'Reilly, 2003.
[7] Freeman, Elisabeth; Freeman, Eric; Bates, Bert. "*Head First Design Patterns*". O'Reilly, 2004.
[8] Gibert, Marc. "*Desenvolupament d'aplicacions en connexió amb bases de dades"*. Editorial: UOC.
[9] Jeffiries, Ron; Andersen, Ann; Hendrickson Chet. "*Extreme Programming Installed"*. Addison-Wesley, 2000.
[10] Korth. Henry F.; Silberschatz, Abraham. "*Fundamentos de Bases de datos*". MgGrawHill.
[11] Pressman, Roger S. "*Software Engineering*". McGrawHill, 1991.

9.1. Enllaços i referències a Internet

http://www.mysql.com

http://ldc.usb.ve/~mgoncalves/IS3/Clase%201c%20IS-Persistencia.pdf

http://dis.um.es/~jmolina/Persistencia%20de%20Objeto%20JDO.pdf

http://www.fing.edu.uy/inco/grupos/coal/uploads/Investigaci%F3n/mps07.pdf

http://ca.wikipedia.org/wiki/Mapatge_d%27objectes_relacional

http://www.agiledata.org/essays/mappingObjects.html

http://ocw.uoc.edu/computer-science-technology-and-multimedia/bases-de-datos/bases-de-datos/P06_M2109_02153.pdf

http://www.programacion.com/articulo/conceptos_basicos_de_orm_object_relational_mapping_349

http://materias.fi.uba.ar/7500/montaldo-tesisdegradoingenieriainformatica.pdf

http://www.db4o.com/espanol/db4o%20Whitepaper%20-%20Bases%20de%20Objetos.pdf

http://es.wikipedia.org/wiki/Java_Database_Connectivity

http://surpatterns.com/sitio/tutoriales-programacion-surpatterns/hibernate/tutorial-hibernate-en-espanol-cache-de-primer-y-segundo-nivel/

http://www.fing.edu.uy/inco/cursos/tsi/TSI2/teorico/Clase11.pdf

http://www.fdi.ucm.es/profesor/llorente/AIC/Leccion-5.pdf

http://aurea.es/wp-content/uploads/sistemasinformacion.pdf

http://es.wikipedia.org/wiki/Coherencia_de_cach%C3%A9

http://www.dte.eis.uva.es/Docencia/ETSII/SMP/BAK/tema5/ANEXO1.pdf

http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/222

http://migranitodejava.blogspot.com.es/2011/06/ehcache.html

http://www.ehcache.org/documentation

http://www.dataprix.com/21-os-sesenta-setenta-sistemas-centralizados

http://ocw.uoc.edu/informatica-tecnologia-i-multimedia/bases-de-dades/materials/

http://sg.com.mx/content/view/966

http://www.enriquedans.com/2011/11/entender-el-futuro-la-evolucion-de-las-bases-de-datos.html

http://es.wikipedia.org/wiki/Hibernate

http://unpocodejava.wordpress.com/2011/01/10/tecnicas-de-bloqueo-sobre-base-de-datos-bloqueo-pesimista-y-bloqueo-optimista/

http://www.unife.edu.pe/ing/desarrollo.doc

http://wiki.typo3.org/index.php/MVC_Framework

http://wiki.typo3.org/index.php/Object_Persistence_Framework

http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/179

http://www.hibernate.org/

http://carlos-henriquez.blogspot.com.es/2010/07/hibernate-framework-para-el-mapeo-de.html

A. FRAMEWORK DE BASE DE DADES – INSTAL·LACIÓ I DISTRIBUCIÓ

El present annex és una guia destinada a facilitar el procés de configuració de l'entorn de desenvolupament i la instal·lació del "core" del framework desenvolupant en el present PFC.

Prerequisits software

Els prerequisits per portar a terme la configuració de l'entorn de desenvolupament són els següents:

- Java JDK 1.5 o superior.
- ANT 1.5.4 o superior.
- Dependències de llibreries correctament instal·lades al classpath. Per motius de grandària no s'han pogut incloure en el projecte i, per tant, prèviament caldrà obtenir-les[7].
- Motor de base de dades instal·lat i funcionant correctament (per a les proves s'ha utilitzat MySQL).
- Scripts de BBDD executats sobre el motor
  - AgenciaViatgesModel.sql
  - AgenciaViatgesDades.sql

Estructura del projecte

A continuació es detalla la estructura amb la que està desenvolupat el projecte:

```
${cmeni_pfc}/
  |
  |-cmeni_fw_bd:           Directori amb el framework de base de dades
  |-cmeni_webapp_agencia: Directori amb el projecte de la webapp d'exemple d'ús.
  |-env:                   Directori per a configurar l'entorn de desenvolupament. És comú pel
                           framework de base de dades i per a la web d'administració.
```

Configuració de l'entorn de desenvolupament

Per a poder compilar el projecte és necessari tenir definit l'entorn en el directori env comentat anteriorment. Per fer-ho caldrà realitzar els següents passos:

- Crear un executable (*setenv.bat*) i definir les següents propietats que s'obtindran des del fitxer *build.xml* i que permetrà generar el distribuïble mitjançant l'eina ANT:

```
@echo off

if "%AOC_ENV%"=="y" goto fin
set AOC_ENV=y

set PRJ_HOME=C:\Users\Jammy\Desktop\cmeni_pfc\cmeni_webapp_agencia
set ANT_HOME=C:\ANT\apache-ant-1.6.5
set JAVA_HOME=C:\java\jdk1.6.0_31
set TOMCAT_HOME=C:\tomcat\apache-tomcat-5.5.36-pfc

set PATH=%ANT_HOME%\bin;%JAVA_HOME\bin%;%PATH%

echo ---------- [PFC - Carlos MENA (WebAPP)] ----------
echo PRJ_HOME=%PRJ_HOME%
echo ANT_HOME=%ANT_HOME%
echo JAVA_HOME=%JAVA_HOME%
echo ---------- [PFC - Carlos MENA (WebAPP)] ----------
goto fin

:fin
```

[7] Les llibreries necessàries es troben especificades al capítol de disseny del framework.

```
cd %PRJ_HOME%
@echo on
```

Estructura de les fonts

L'estructura resumida de les fonts del subprojecte és la que es detalla en el següent esquema:

```
${cmeni_pfc}/cmeni_fw_bd
  |
  |-build: Directori on es compilen totes les classes i es genera el JAR de l'aplicació
  |-dist: Directori on es genera els JAR de la distribució
  |-config: Directori amb els recursos de configuració
  |-src: Directori amb el codi font del projecte
  |-sql: Scripts SQL per la creació de la BD i inserir els valors de configuració
  |-lib: Llibreries necessàries per a la execució de l'aplicació
  |-libExt: Llibreries necessàries per a la compilació de l'aplicació (i disponibles al servidor
            d'aplicacions en el que es desplegarà).
```

Compilació de l'aplicació

Per tal de compilar el codi del sistema executarem la següent comanda des del directori `${cmeni_pfc}/cmeni_fw_bd`:

```
ant clean dist
```

L'*script ANT* s'encarregarà de generar el fitxer JAR del framework desenvolupat. Cal tenir en compte les propietats del fitxer de configuració necessàries per a poder realitzar la comunicació amb la base de dades.

Recursos de configuració de l'aplicació

uoc-bd-config.properties

Es troba disponible a :

```
${cmeni_pfc}/cmeni_fw_bd/config/uoc-bd-config.properties
```

Aquest fitxer ha de contenir les següents propietats:

Nom	Descripció
bd.connection.driver	Classe del driver de connexió amb la base de dades
bd.connection.driver.id	Identificador del driver. S'utilitzarà per a poder crear per reflection els beans.
bd.connection.url	URL d'accés a la base de dades
bd.connection.user	Usuari del catàleg en el que s'ha executat l'script de creació del model
bd.connection.pass	Paraula de pas de l'usuari per a accedir al catàleg.
bd.connection.bdbeans.prefix.package	Configuració del proveïdor mitjançant el qual es realitzaran els mappings corresponents al motor de base de dades que estigui configurat. Aquesta propietat indicarà el package en el qual es trobaran els mappings i que seran carregada mitjançant reflection: *${bd.connection.bdbeans.prefix.package}${bd.connection.bdbeans.prefix.class} ${bd.connection.driver.id}${bd.connection.bdbeans.sufix.class}*
bd.connection.bdbeans.prefix.class	Prefix de la classe amb els mappings (es carregarà per reflection).
bd.connection.bdbeans.sufix.class	Sufix de la classe amb els mappings (es carregarà per reflection).
bd.cache.enabled	Flag per a especificar si es vol habilitar la cache o no.
bd.beans.basedir	Directori base en el que es generaran els beans (POJO's) d'accés a la base de dades.
bd.beans.package	Package java en el que es crearan els beans (a partir del directori base especificat anteriorment).

Un exemple d'aquest fitxer de configuració:

```
# Connexio a la BBDD
bd.connection.driver=com.mysql.jdbc.Driver
bd.connection.driver.id=mysql
bd.connection.url=jdbc:mysql://127.0.0.1/agencia_viatges
bd.connection.user=root
bd.connection.pass=

#Configuració provider
```

```
bd.connection.bdbeans.prefix.package=edu.uoc.pfc.mena.db.core.provider
bd.connection.bdbeans.prefix.class=TaulaBDInfo
bd.connection.bdbeans.sufix.class=Impl

# Configuracio de la cache
bd.cache.enabled=false

# Generacio de BD Beans
bd.beans.basedir=C:/Users/Jammy/Desktop/cmeni_pfc/cmeni_fw_bd/src
bd.beans.package=test.main.exemple.pojo.client
```

log4j.xml

Correspon al fitxer de configuració del log4j per a la generació dels logs de l'aplicació:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

<appender name="Fitxer" class="org.apache.log4j.FileAppender">
        <param name="File" value="/logs/cmeni_fw_bd.log"/>
        <param name="Append" value="true"/>
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d{ISO8601} [FW BD] - %p - %m%n"/>
        </layout>
</appender>

<category name="edu.uoc.pfc.mena.db" additivity="false">
        <level value="debug"/>
        <appender-ref ref="Fitxer"/>
</category>

<root>
    <level value="fatal"/>
     <appender-ref ref="Fitxer"/>
</root>

</log4j:configuration>
```

B. APLICACIÓ WEB – INSTAL·LACIÓ

El present annex és una guia destinada a facilitar el procés de configuració de l'entorn de desenvolupament i la instal·lació de l'aplicació web desenvolupant en el present PFC com a exemple d'ús del framework.

Prerequisits software

Els prerequisits per portar a terme la configuració de l'entorn de desenvolupament són els següents:

- Java JDK 1.5 o superior.
- ANT 1.5.4 o superior.
- Dependències de llibreries correctament instal·lades al classpath. Per motius de grandària no s'han pogut incloure en el projecte i, per tant, prèviament caldrà obtenir-les[8].
- Motor de base de dades instal·lat i funcionant correctament (per a les proves s'ha utilitzat MySQL).
- Servidor d'aplicacions instal·lat i funcionant correctament (per a les proves s'ha utilitzat Apache Tomcat 5.5.36).
- IDE de desenvolupament (Eclipse)
- Scripts de BBDD executats sobre el motor
    - AgenciaViatgesModel.sql
    - AgenciaViatgesDades.sql

[8] Les llibreries necessàries es troben especificades al capítol de disseny del framework.

Estructura de les fonts

L'estructura resumida de les fonts del subprojecte és la que es detalla en el següent esquema:

```
${cmeni_pfc}/cmeni_webapp_agencia
  |
  |-build: Directori on es compilen totes les classes i es genera el WAR de l'aplicació
  |-dist: Directori on es genera el WAR de la distribució
  |-config: Directori amb els recursos de configuració
  |-src: Directori amb el codi font del projecte
  |-lib: Llibreries necessàries per a la execució de l'aplicació
  |-libExt: Llibreries necessàries per a la compilació de l'aplicació (i disponibles al servidor
            d'aplicacions en el que es desplegarà).
  |-webroot
  |  |
  |  |- content
  |  |   |
  |  |   |-css: Directori amb els estils
  |  |   |-img: Directori amb les imatges
  |  |   |-js: Directori amb els Javascripts
  |  |   |-jsp: Directorit amb les JSP
  |  |   |  |
  |  |   |  |-error: Directori amb les JSP d'error
  |  |
  |  |-web-inf: Directori amb els fitxers de configuració de l'aplicació web (web.xml i struts)
  |  |  |-classes: Directori que conté l'arxiu de properties de configuració, l'arxiu amb els tags
                  d'idioma i l'arxiu de configuració de logs i l'arxiu de context d'Spring.
  |  |  |-lib: Directori amb les llibreries necessaries per a l'execució de la web.
  |  |  |-tld: Directori amb els tld de les taglib
```

Compilació de l'aplicació

Per tal de compilar el codi del sistema és necessari configurar l'entorn de desenvolupament comentat en l'annex anterior (*setenv.bat*). Un cop configurat, executarem la següent comanda des del directori `${cmeni_pfc}/cmeni_webapp_agencia`:

```
ant clean war (deploy)
```

L'*script ANT* s'encarregarà de generar el fitxer WAR amb l'aplicació web. Cal tenir en compte les propietats del fitxer de configuració necessàries per a poder realitzar la comunicació amb la base de dades i que s'han comentat en l'annex de distribució del framework.

Incloent l'argument *deploy* es desplegarà automàticament al servidor d'aplicacions configurat a l'entorn de desenvolupament.

Recursos de configuració de l'aplicació

El recursos necessaris pel funcionament de la web corresponen als mateixos del framework, donat que es tracta d'una aplicació web d'exemple de l'ús del framework desenvolupat.

Addicionalment, es disposen dels fitxers de configuració dels *framewoks* utilitzats en la implementació de la web -Struts i Spring- disponibles a *${cmeni_pfc}/cmeni_webapp_agencia/webroot/content/WEB-INF* i *${cmeni_pfc}/cmeni_webapp_agencia/webroot/content/WEB-INF/classes*, respectivament.

Generació dels beans d'accés a la base de dades

Per a generar els beans d'accés a la base de dades només serà necessari crear un classe amb un mètode main que permeti executar la següent comanda[9]:

```
edu.uoc.pfc.mena.db.utilities.GeneradorBeans.getInstance().generaBDBeans();
```

En el projecte es disposa d'una classe -`generador.bd.beans.GeneradorBDBeans`- que implementa aquesta crida.

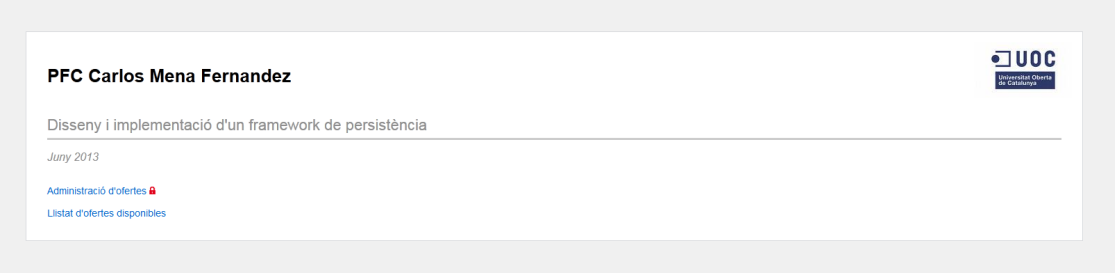
[9]Per a poder-ho executar correctament caldrà carregar els recursos de configuració del framework al classpath.

C. APLICACIÓ WEB – MANUAL USUARI

El present annex correspon a una mostra del funcionament de l'aplicació web amb el framework desenvolupat amb les funcionalitats descrites al capítol del disseny de l'aplicació.

Pàgina d'inici

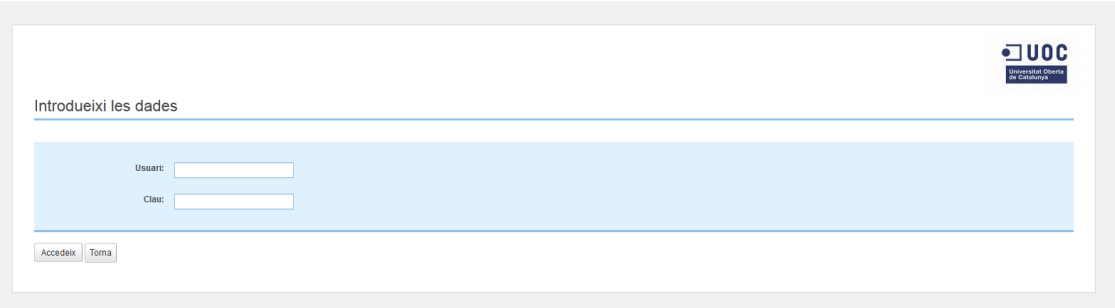
La pàgina d'inici de l'aplicació ofereix dues opcions: administrar les ofertes disponibles a l'agència (requereix fer login al sistema) o mostrar el llistat amb les ofertes disponibles.



Login com a administrador del portal

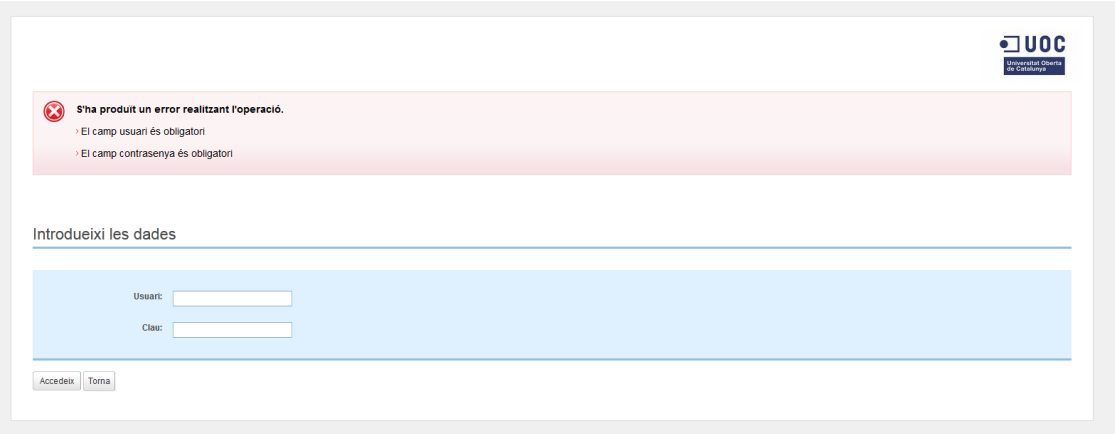
Login

Es mostra un formulari per a que un administrador pugui introduir les seves credencials d'accés i poder realitzar la gestió de les ofertes:

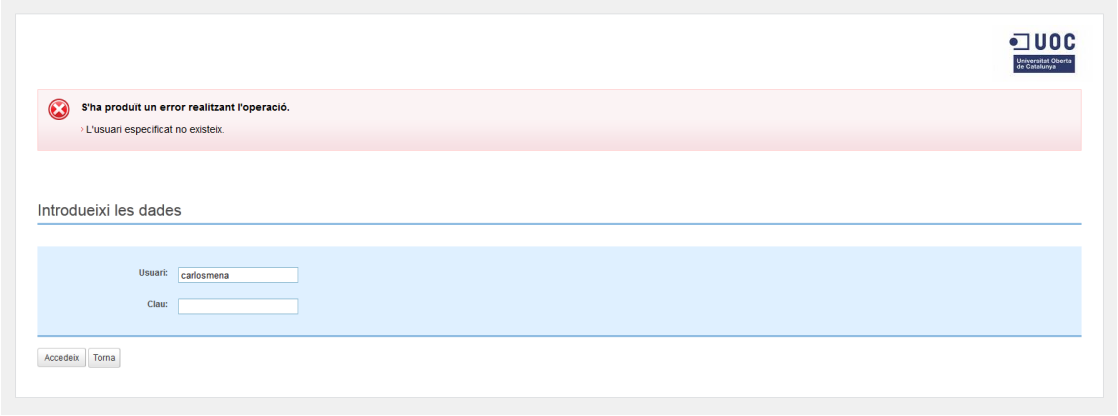


En cas de realitzar el *login* correctament es mostra el llistat de les ofertes amb les funcionalitats disponibles pels administradors (alta, modificació i eliminació) i que es mostrarà més endavant.

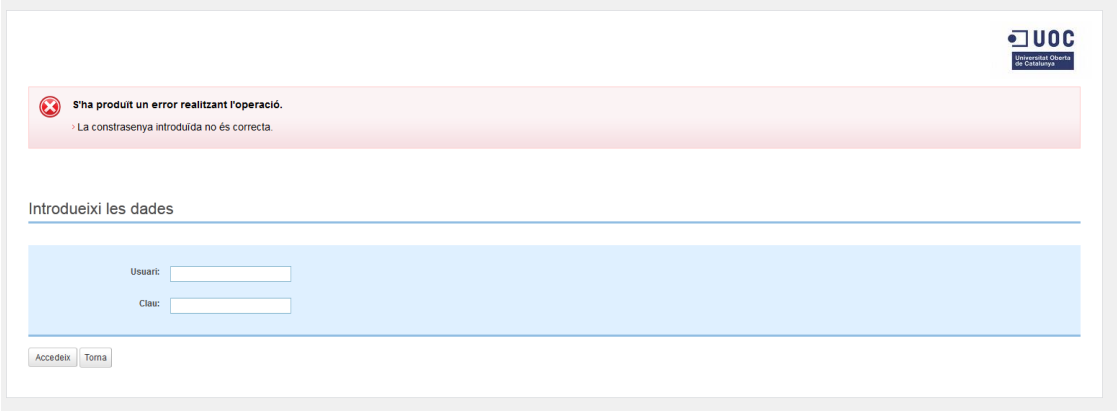
Error - Login sense dades



Error – Usuari desconegut



Error - Login amb paraula de pas errònia



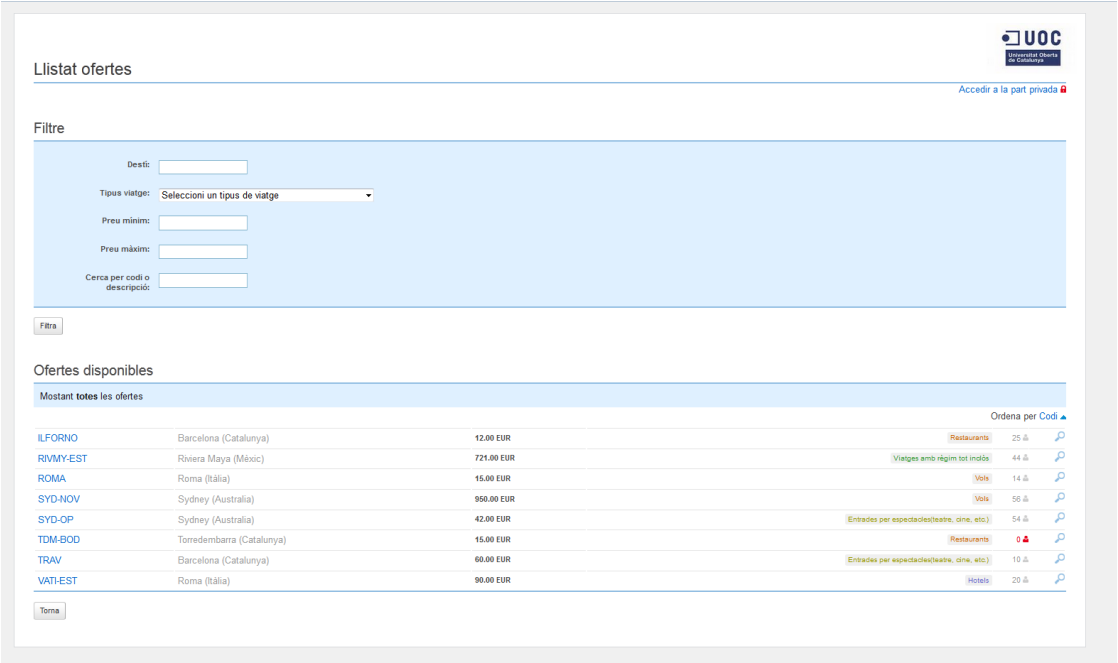
Llistat d'ofertes

Correspon a la mateixa pàgina que s'ha mostrat en el cas dels administradors, però en la que només es permet el filtratge i la visualització de les ofertes. A continuació es mostren les diferents vistes disponibles.

Vista general

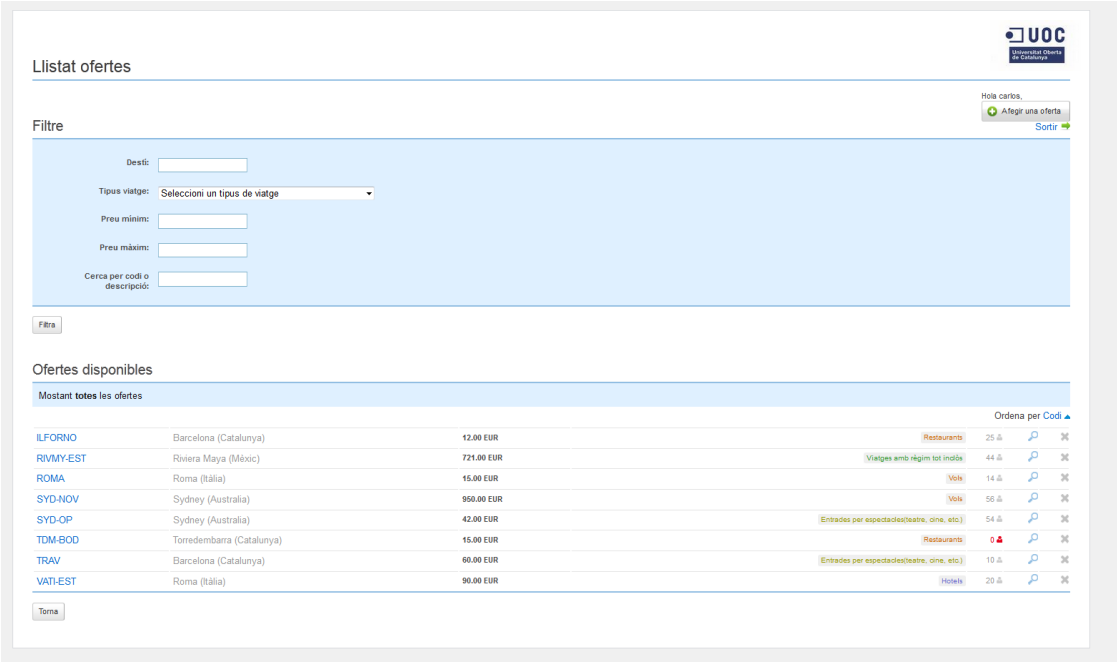
Serà la vista que inclourà el llistat amb les ofertes disponibles a l'agència mostrant part de la informació de la mateixa. Els camps que es mostraran al llistat, són els següents:

- Codi
- Localitat (Pais)
- Nom del tipus de viatge
- Preu
- Places disponibles



Vista administrador

Correspon a la vista a la que accediran els administradors un cop logats al sistema. Correspon a la mateixa vista general però incloent les funcionalitats bàsiques dels administradors (altes, modificacions i eliminacions) així com el nom de l'usuari i un enllaç per a sortir de la vista d'administració:

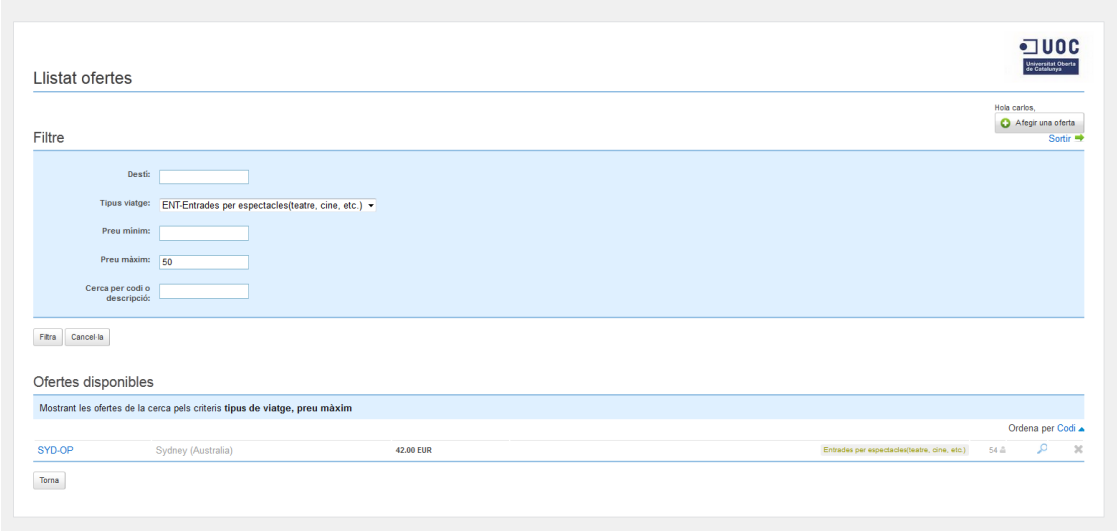


Llistat d'ofertes filtrades

Els criteris disponibles al filtre són els següents:

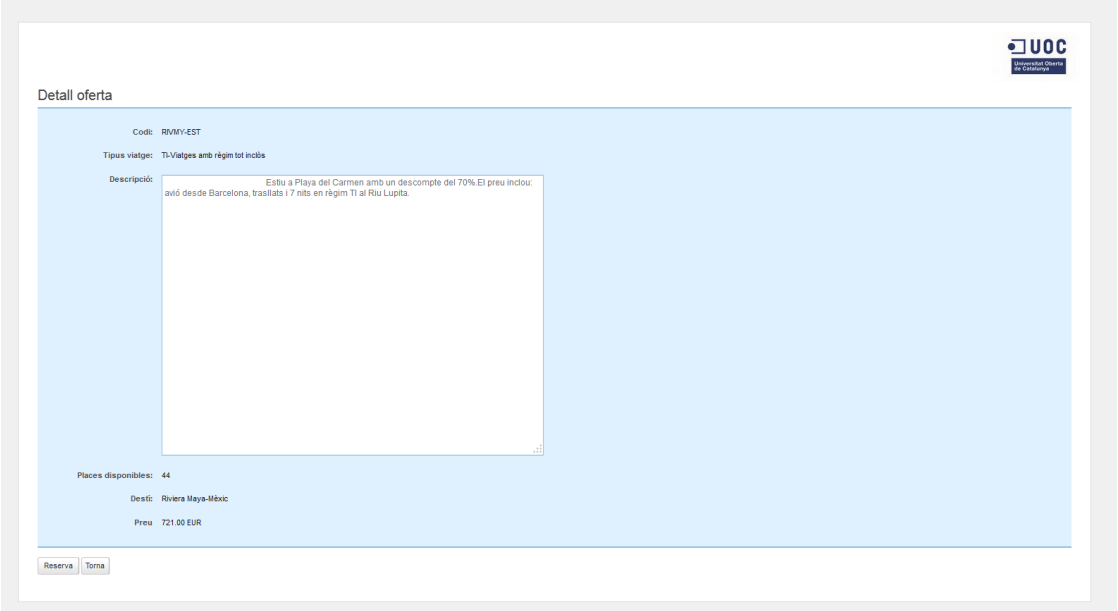
- Cerca: Permet filtrar pel codi i la descripció de la oferta.
- Destí: Permet especificar un filtre pel destí tant a nivell de població com de país.
- Tipus viatge: Filtre pel tipus de viatge. Es una combo amb els valors existents a la taula TIPUS_VIATGE.
- Preu: Permet especificar al filtre un rang de preus (preu mínim i preu màxim).

Un cop aplicat el filtre es mostrarà un boto per a poder eliminar els criteris aplicats a la cerca. A continuació es mostra un exemple d'ús del filtre:



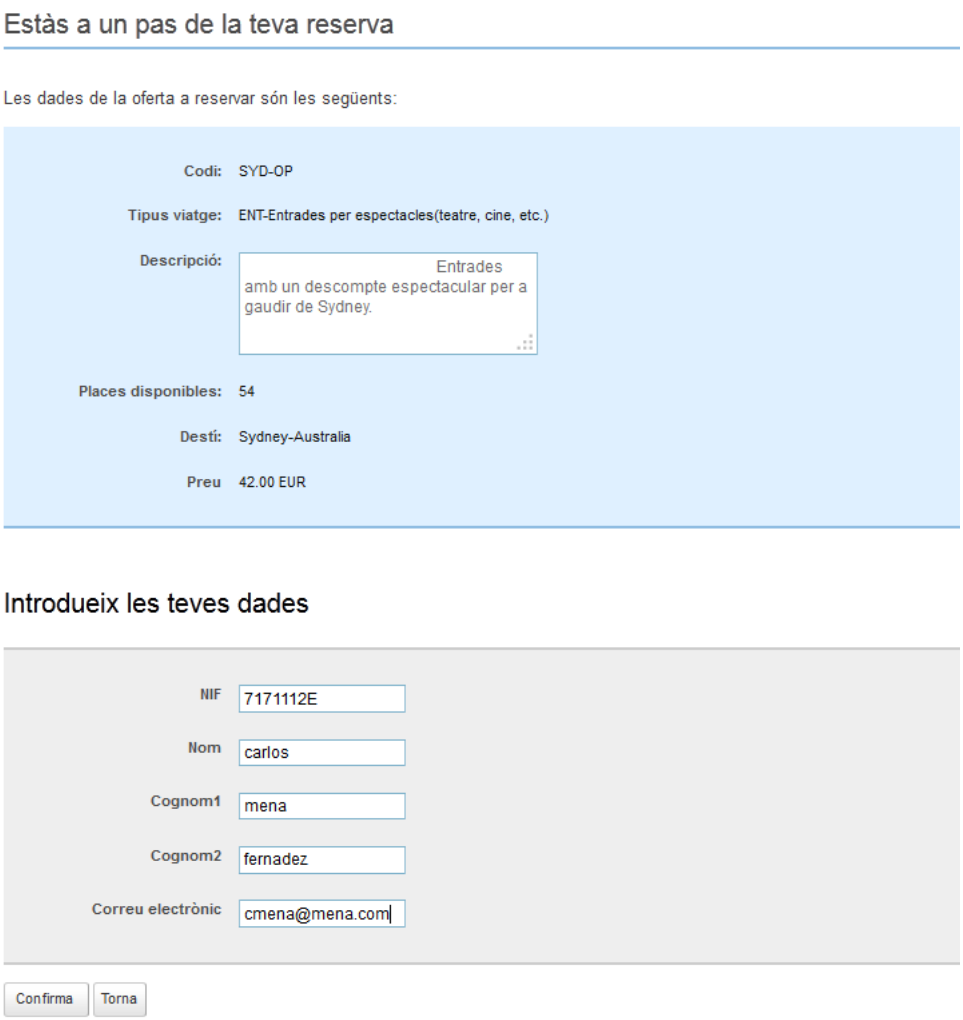
Visualitzar una oferta

La pantalla de visualització, específica pels clients del sistema, mostrarà totes les dades disponibles de la oferta. un botó per a poder realitzar la reserva i un altre per a poder tornar enrera.

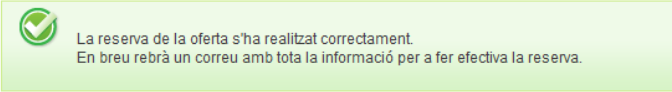


Reserva d'una oferta

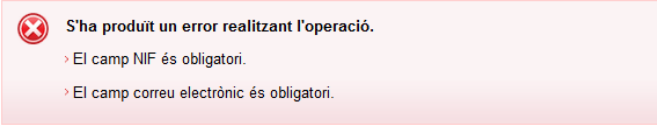
Per a poder reservar una oferta, client haurà de visualitzar-la prèviament (mitjançant la opció de vista del llistat) i seleccionat la opció de Reserva. A continuació, el sistema mostrarà les dades de la oferta i facilitarà un formulari al client per a que introdueixi les seves dades:



Un cop omplertes les confirmi mitjançant el botó de confirmar, es mostrarà una pantalla de confirmació de la operació:

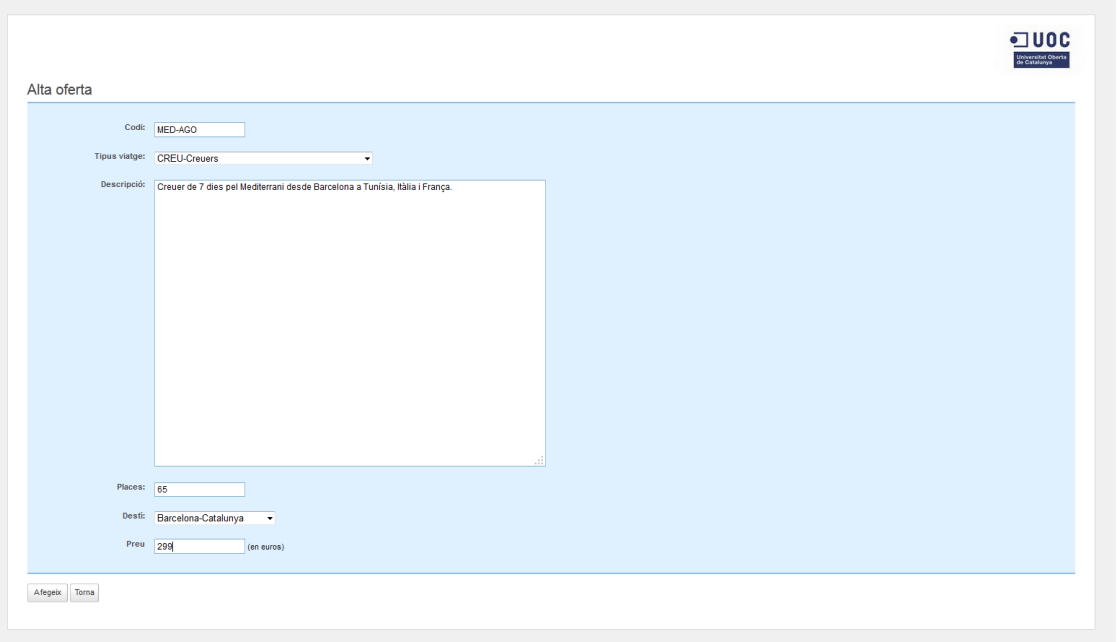


En cas de no omplir els camps mínims necessaris per a confirmar la reserva, es mostrarà l'error corresponent:

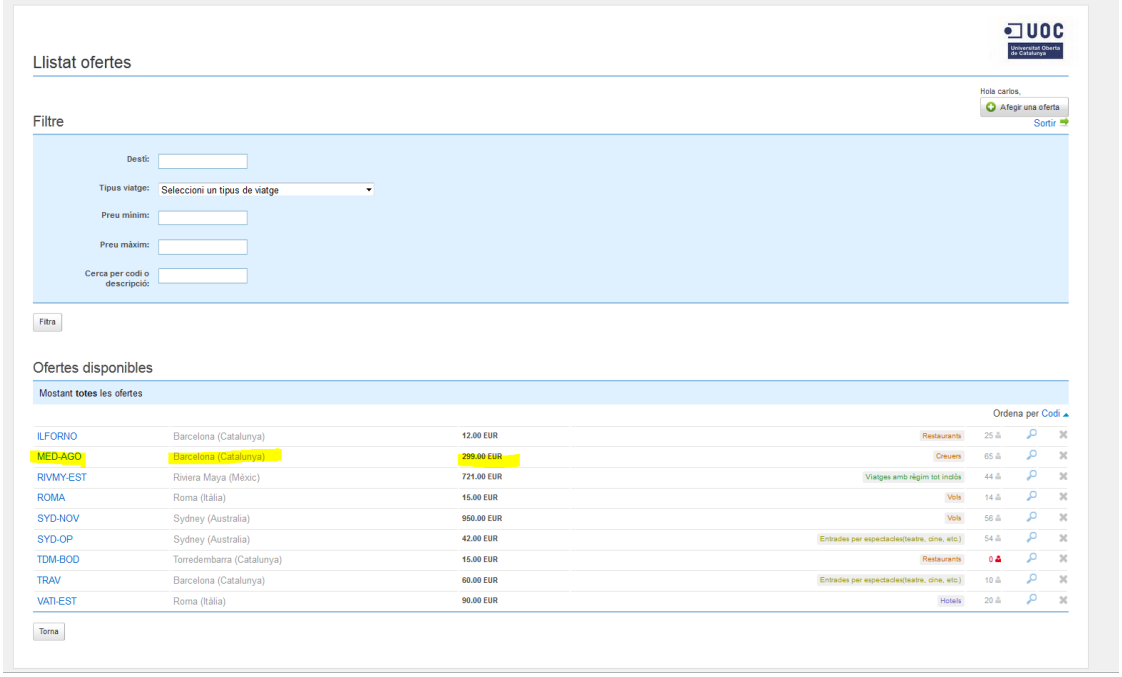


Alta d'una oferta

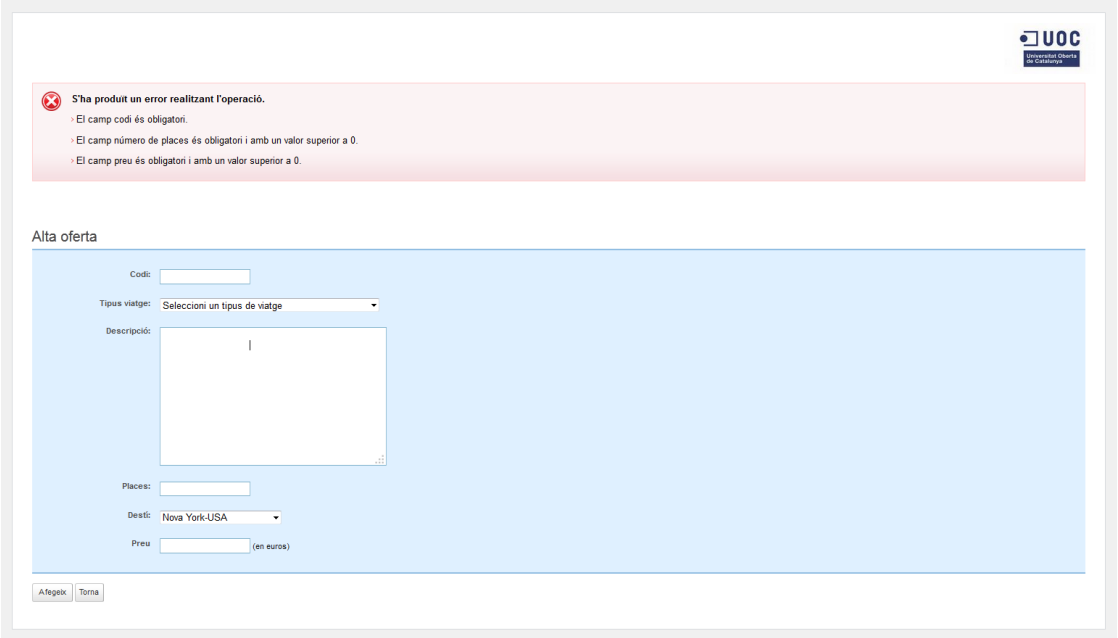
Només estarà disponible pels administradors del sistema, i s'accedirà mitjançant l'enllaç disponible a la pàgina d'inici de l'administrador:



Un cop introduïdes les dades de la nova oferta i, si l'alta s'ha pogut realitzar al sistema, es redirigirà a la vista amb el llistat d'ofertes que inclourà la nova oferta afegida:

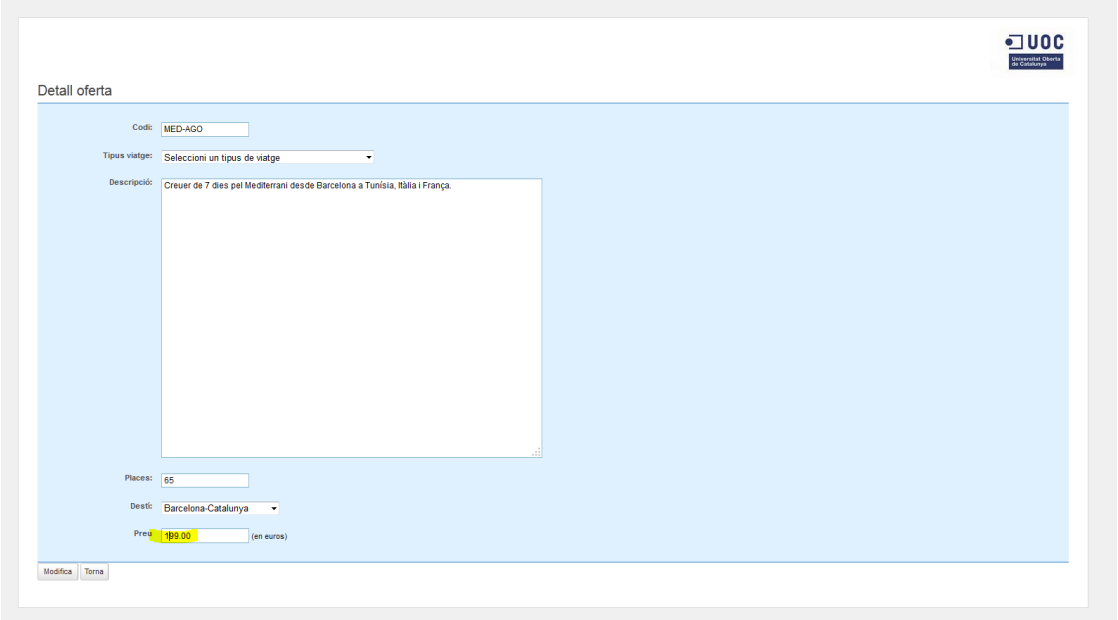


D'altra banda, en cas de produir-se un error es mostrarà en la mateixa pantalla la descripció dels motius pels que no s'ha pogut realitzar l'alta:

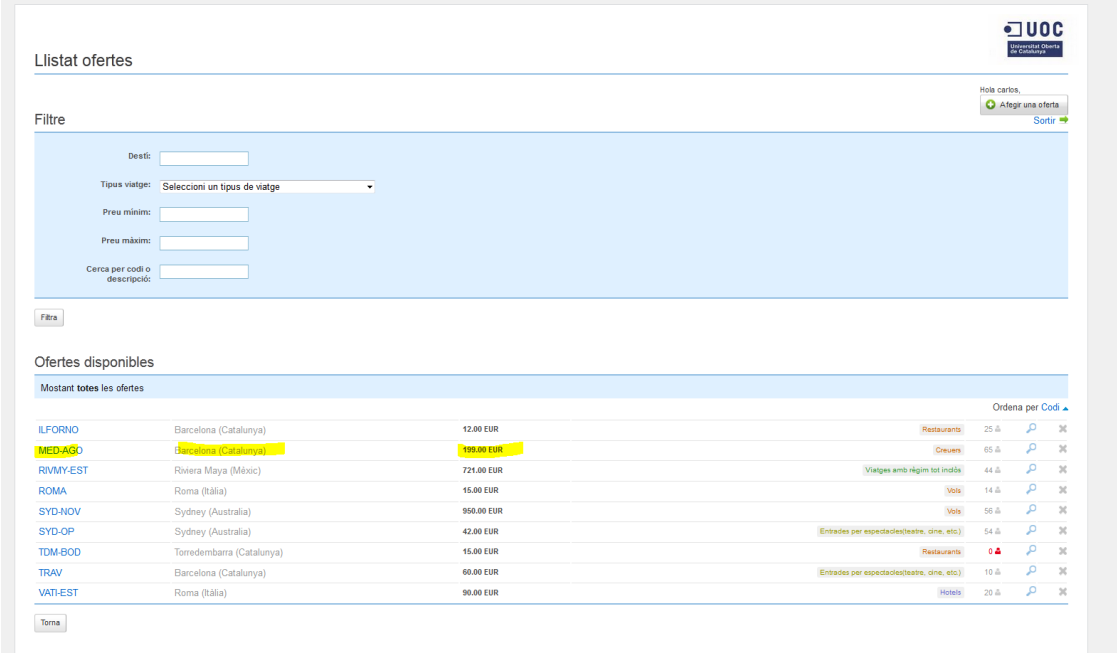


Modificació d'una oferta

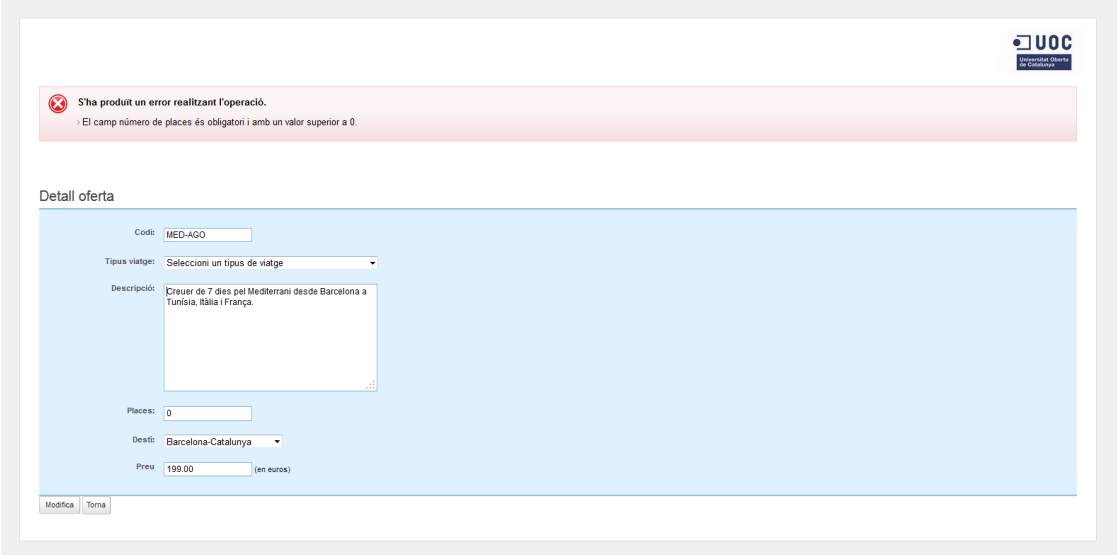
Només estarà disponible pels administradors del sistema. Per accedir-hi caldrà seleccionar la oferta a través del llistat (en cas de no ser administrador es mostraria el detall de la oferta):



Un cop introduïdes les dades de la oferta i, només en el cas que la operació es realitzi correctament, es redirigirà a la vista amb el llistat d'ofertes que inclourà les modificacions de la oferta seleccionada:

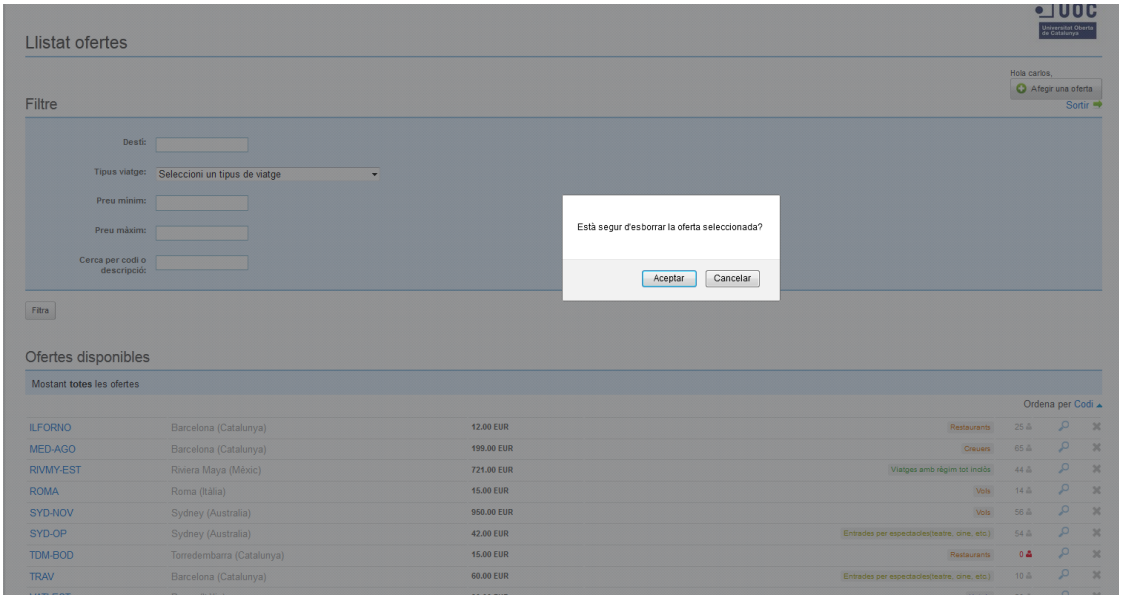


D'altra banda, en cas de produir-se un error es mostrarà en la mateixa pantalla la descripció dels motius pels que no s'ha pogut realitzar la modificació:

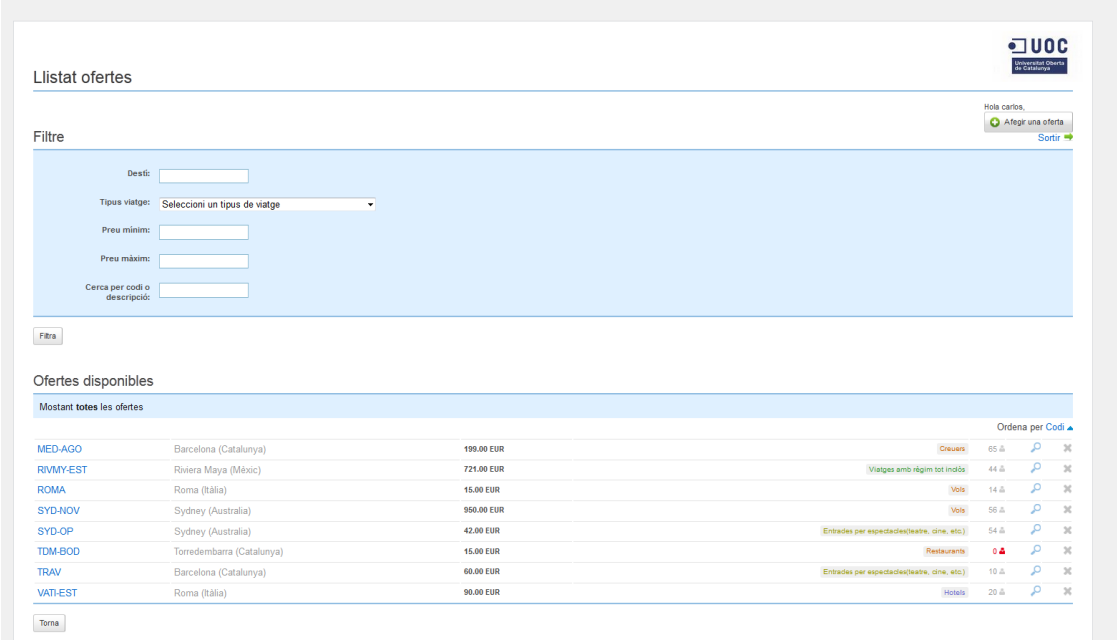


Eliminació d'una oferta

De la mateixa manera que l'alta i la modificació, aquesta operació només estarà disponible pels administradors. Per accedir-hi caldrà seleccionar la opció de eliminació d'una oferta del llistat. Un cop seleccionada la oferta que es vol eliminar – en aquest cas, s'ha seleccionat la d'ILFORNO-, el sistema mostrarà un missatge de confirmació de la operació.



En cas, de confirmar-ho s'eliminarà la oferta del sistema i es mostrarà el llistat d'ofertes sense la oferta que s'acaba d'esborrar.



D. LLIBRERIES UTILITZADES

En el present annex es mostrarà un llistat de les llibreries utilitzades en el desenvolupament del framework i de l'aplicació web donat que no s'han incorporat al projecte Eclipse entregat a través del RAC, per problemes amb l'espai de la distribució.

Així doncs, les llibreries utilitzades són les següents:

- Framework
    - lib

Nombre	Tamaño
cmeni-pfc-fw-bd.jar	64,85 KB
commons-beanutils-1.8.3.jar	226,58 KB
commons-beanutils-bean-collections-1.8.3.jar	25,95 KB
commons-beanutils-core-1.8.3.jar	201,87 KB
commons-lang3-3.1.jar	308,4 KB
commons-logging-1.1.3.jar	60,6 KB
hsqldb.jar	1,36 MB
log4j.jar	344,4 KB
mysql-connector-java-5.1.25-bin.jar	828,52 KB
ojdbc14.jar	1,47 MB

- Web d'adminitració:
  - libExt

Nombre	Tamaño
servlet.jar	78,18 KB

  - lib

Nombre	Tamaño
cmeni-pfc-fw-bd.jar	67,5 KB
cmeni-pfc-webapp-config.jar	827 bytes
cmeni-pfc-webapp.jar	71,77 KB
commons-beanutils-1.8.3.jar	226,58 KB
commons-beanutils-bean-collections-1.8.3.jar	25,95 KB
commons-beanutils-core-1.8.3.jar	201,87 KB
commons-collections.jar	545,29 KB
commons-digester.jar	164,5 KB
commons-lang3-3.1.jar	308,4 KB
commons-logging-1.1.3.jar	60,6 KB
commons-validator.jar	82,48 KB
jstl.jar	20,2 KB
log4j.jar	344,4 KB
mysql-connector-java-5.1.25-bin.jar	828,52 KB
ojdbc14.jar	1,47 MB
spring-2.5.6.SEC02.jar	2,81 MB
standard.jar	384,04 KB
struts-el.jar	206,8 KB
struts.jar	514,24 KB

Per a facilitar l'accés a les mateixes, s'han afegit totes al següent repositori públic:

```
https://www.dropbox.com/sh/9xd3k41kv0m2c4w/mt1SSIADOn
```