TREBALL FINAL DE CARRERA

DISSENY I IMPLEMENTACIÓ D'UN FRAMEWORK DE PERSISTÈNCIA

Estudiant:

Cognoms: Artigas Bosch

Nom: Guillem

Llicència d'autor

Índex

Annex

1. Definició general del projecte

1.1. Descripció general i objectius del projecte

L'objectiu principal d'aquest treball, és profunditzar en la creació d'un framework de persistència, realitzant tot el treball previ de recerca, anàlisi i disseny. Crear un framework propi i una aplicació on es pugui veure el funcionament d'aquest framework dissenyat. Per poder desenvolupar aquest projecte, a part de la recerca (que és una part també important del treball), posarem en pràctica la base que hem aprés a les diferents assignatures cursades durant aquests anys amb la enginyeria.

Aquest projecte consisteix en la realització d'un framework de persistència per realitzar l'enllaç entre el model de dades relacional i el model de dades orientat a objectes, ja que la gran majoria de base de dades d'avui dia són relacionals. L'aplicació que implementarem per tal de provar el nostre framework consisteix en l'adaptació web d'un software financer dissenyat amb anterioritat per mi que serveix per tenir un control de la tresoreria.

Per tan, partim d'una base de dades amb MySQL ja dissenyada i implementada, crearem un framework de persistència per tal de realitzar l'enllaç amb la base de dades. Per tal de realitzar l'aplicació utilitzarem la tecnologia del framework ZK per realitzar la part web, i Apache Tomcat per la part del servidor. La tecnologia que utilitzarem per la creació del framework de persistència en principi serà la de Hibernate, tot i que després de realitzar l'estudi previ de totes les tecnologies i comparacions entre elles, si es troba una tecnologia més adient, lògicament la canviarem.

Primer de tot, realitzarem l'estudi previ de les 4 tecnologies: JDBC, Hibernate, JPA i iBATIS. Un cop estudiada cadascuna d'elles i realitzada la comparació, escollirem quina és la que s'adapta millor el nostre projecte i a la base de dades relacional que tenim. A partir d'aquí realitzarem l'anàlisi i disseny del nostre framework de persistència i finalment la implementació. Un cop creat el framework, farem la implementació de l'entorn web per tal de poder comprovar el funcionament del framework creat anteriorment.

L'aplicació serveix per portar el control dels comptes bancaris d'una família o bé d'una empresa. Aquesta aplicació podem entrar totes les dades de cobraments i pagaments segons l'entitat o entitats que treballem i també de l'empresa. Aquestes dades les podem entrar manualment, o bé utilitzant l'extracte bancari que ens proporciona la pròpia entitat. L'aplicació té la particularitat també de poder realitzar inversions i/o préstecs per tal de poder fer una estimació dels comptes bancaris.

Aquesta aplicació està pensada per facilitar la feina de la tresoreria, així com també visualitzar els resultats utilitzant el mètode per caixa. L'adaptació d'aquest software utilitzant la tecnologia del J2EE ve de l'evolució que han tingut les aplicacions els últims anys. La majoria de softwares que trobem avui dia són tots online.

1.2. Com va sorgir la idea

La idea va sortir d'un projecte actual que s'està duent a terme des de fa anys, en el qual es va dissenyar aquesta aplicació en entorn local. Com que la majoria de persones o empreses necessiten en tot moment consultar les dades financeres de manera immediata i en la societat que vivim, des de qualsevol lloc del món, fa temps que estàvem pensant en mirar de realitzar el projecte en entorn web, per tal d'obrir més mercat. A part, cada cop més empreses utilitzen les aplicacions en entorn web, ja que això comporta que necessiten molta menys infraestructura física, a part de la comoditat de poder utilitzar el software fàcilment des de qualsevol lloc, indiferentment de si es troben a la oficina o fora d'ella.

1.3. Estudi previ realitzat

Partim d'una aplicació ja dissenyada i funcionant en entorn local. Per tan, hem de dissenyar l'aplicació des de zero, sols podem utilitzar la base de dades i el seu disseny per fer la nova aplicació. El fet de renovar l'aplicació i fer-la de tipus online, ens permetrà poder posar en pràctica els coneixements adquirits aquests últims anys a les diferents assignatures impartides i on s'han donat la base d'aquestes tecnologies.

El fet de treballar en una empresa de desenvolupament de software a mida, ha ajudat molt a la introducció a la programació i els sistemes de gestió de base de dades. Tot i que no treballem amb tecnologia J2EE, de moment sembla que el futur de les aplicacions han de ser al núvol, i aquesta tecnologia és una d'elles per tal de poder realitzar un canvi en la manera de dissenyar i desenvolupar el software. Amb l'aprenentatge adquirit ens els estudis i la pràctica entre els estudis i el lloc de treball, han de servir per poder arribar a fer l'actualització del programa.

1.4. Planificació temporal

Tenint en compte totes les dates ja establertes per els consultors, farem una mica de resum temporal de les dates generals del projecte. En el Gantt que hi ha a la última pàgina de la planificació, es pot veure detalladament totes les dates concretes.

El diagrama de Gantt està pensat per no realitzar cap dia de festa, ja que degut a la gran quantitat d'hores de treball i també la feina que és de jornada laboral completa, s'han d'aprofitar molts cops els festius i caps de setmana per tal de poder realitzar la feina dels estudis. Considerant això, la planificació del projecte te previst el començament amb data inici 20/09/2010 (que és l'inici de la PAC1) fins el 13/01/2011 que en principi s'ha de realitzar l'entrega final.

Lògicament que hi podran haver petits canvis en la planificació, ja que sempre hi ha imprevistos, però sempre tenint en compte les entregues oficials.

2. Especificació del projecte

2.1. Composició del software

La tecnologia que farem servir per realitzar el framework i l'aplicació serà MySQL per el sistema de base de dades (ja que com hem dit anteriorment la base de dades ja la tenim dissenyada), per realitzar el framework de persistència en principi el realitzarem amb Hibernate i finalment utilitzarem ZK i Apache Tomcat per tal de fer la web i poder comprovar el funcionament del framework dissenyat. En principi podem veure que es tracten de tecnologies Open Source, i de fàcil instal·lació a qualsevol tipus de servidor web, ja que Apache Tomcat és fàcil de trobar un hosting que ens permeti la seva instal·lació.

2.2. Rols

En el nostre cas tenim un sol rol, ja que es tracta d'una aplicació bàsica. Es tracta de l'operari que es dedica a realitzar el control financer. Aquesta persona tenen la feina de controlar els ingressos i les despeses, per tan podran:

- Donar d'alta i baixa empreses
- Donar d'alta i baixes entitats
- Donar d'alta i baixa pagaments i despeses
- Donar d'alta préstecs

Aquestes son les funcionalitats bàsiques de l'operari, el programa podrà realitzar altres funcionalitats, però aquestes funcionalitats descrites anteriorment, són les que haurem d'implementar i s'encarregarà el nostre framework de persistència de realitzar l'enllaç amb la base de dades. A part d'això, el programa també tindrà altres funcionalitats, com per exemple treure les estadístiques i resultats o poder importar automàticament un extracte bancari.

El projecte es base en el framework de persistència, no en totes les funcionalitats que pot tenir el programa, com pot ser la gestió d'usuaris, sistemes d'identificació d'usuaris i seguretat o treure les estadístiques o resultats. Aquestes són altres funcionalitats que té el software, però nosaltres ens centrarem en tot l'apartat que és la gestió dels números.

2.3. Especificació de funcionalitats

Tot i que en el punt anterior ja hem explicat la majoria de funcionalitats del programa, farem un repàs de les funcionalitats separades per veure més clarament el que es podrà fer amb el programa:

- Gestió d'empreses: es donaran d'alta, es modificaran o es donaran de baixa les empreses que hem de gestionar. L'operari podrà realitzar l'acció convenient depenent de les necessitats que tingui per cada empresa.
- Gestió d'entitats: es donaran d'alta, es modificaran o es donaran de baixa les entitats amb les quals treballem. L'operari podrà realitzar l'acció convenient depenent de les necessitats que tingui cada empresa i les entitats associades. Normalment aquestes entitats son comunes per totes les empreses.
- Gestió d'ingressos i despeses: es donaran d'alta, es modificaran o es donaran de baixa els pagaments o els cobraments que té una empresa. L'operari podrà realitzar l'acció convenient depenent de les necessitats que tingui cada empresa amb els seus ingressos i despeses.
- Gestió de préstecs: es podran calcular i simular un préstec i en cas necessari es podrà donar d'alta com a ingrés i despeses automàticament en el sistema.

El nostre framework serà l'encarregat de la comunicació amb la base de dades i de gestionar totes les funcionalitats descrites anteriorment.

2.4. Resum Planificació

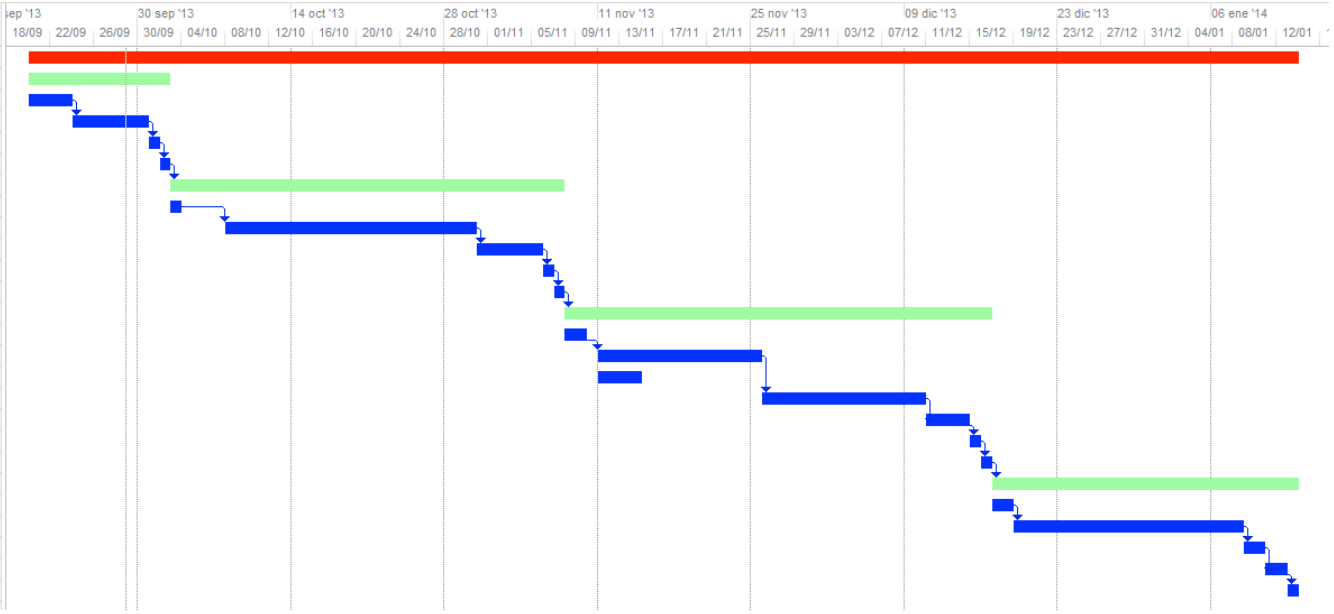
Bàsicament el projecte es separa en quatre parts, que es fan quadrar amb l'entrega de les quatre PACs. Per tan, simplificant quedarien els següents períodes:

- Elaboració del pla de treball (20/09/2013 – 02/10/2013)
- Recerca i anàlisi (03/10/2013 - 07/11/2013)
- Disseny framework i aplicació (08/11/2013 - 16/12/2013)
- Memòria i presentació final (17/12/2013 - 13/01/2014)

Aquests dates, són respecte a les entregues de les PACs, i és la estructura general. Cada apartat té els seus subapartats corresponents amb les dates d'elaboració depenent de la feina que comporta cada apartat. Tot això ho podem veure en el Gantt de la pàgina següent.

3. Gantt

	Nombre de tarea	Duración	Comienzo	Fin
1	**TFC Disseny del projecte**	**84 días?**	**vie 20/09/13**	**lun 13/01/14**
2	**1 - Disseny i desenvolupament del projecte**	**9 días?**	**vie 20/09/13**	**mié 02/10/13**
3	1.1 - Plantejament i estructura del Pla de Treball	2 días?	vie 20/09/13	lun 23/09/13
4	1.2 - Desenvolupament i redactat del Pla de Treball	5 días?	mar 24/09/13	lun 30/09/13
5	1.3 - Revisió del Pla de Treball	1 día?	mar 01/10/13	mar 01/10/13
6	1.4 - Entrega del Pla de Treball	1 día?	mié 02/10/13	mié 02/10/13
7	**2 - Recerca i anàlisi**	**26 días?**	**jue 03/10/13**	**jue 07/11/13**
8	2.1 - Recollida de requeriments	1 día?	jue 03/10/13	jue 03/10/13
9	2.2 - Recerca i Anàlisi	17 días?	mar 08/10/13	mié 30/10/13
10	2.3 - Redacció del document d'anàlisi	4 días?	jue 31/10/13	mar 05/11/13
11	2.4 - Revisió del document d'anàlisi	1 día?	mié 06/11/13	mié 06/11/13
12	2.5 - Entrega del document d'anàlisi	1 día?	jue 07/11/13	jue 07/11/13
13	**3 - Disseny**	**28 días**	**vie 08/11/13**	**lun 16/12/13**
14	3.1 - Formulació del disseny	1 día?	vie 08/11/13	sáb 09/11/13
15	3.2 - Disseny i implementació del framework	11 días?	lun 11/11/13	lun 25/11/13
16	3.3 - Disseny i implementació de la base de dades	4 días?	lun 11/11/13	jue 14/11/13
17	3.4 - Disseny i implementació de l'aplicació	11 días?	mar 26/11/13	mar 10/12/13
18	3.6 - Redacció del document de disseny	3 días?	mié 11/12/13	sáb 14/12/13
19	3.7 - Revisió del document de disseny	1 día?	dom 15/12/13	dom 15/12/13
20	3.8 - Entrega del document de disseny	1 día?	lun 16/12/13	lun 16/12/13
21	**4 - Redacció memòria completa**	**21 días?**	**mar 17/12/13**	**lun 13/01/14**
22	4.2 - Conclusions	2 días?	mar 17/12/13	mié 18/12/13
23	4.3 - Redacció memòria completa	15 días?	jue 19/12/13	mié 08/01/14
24	4.4 - Revisió de la memòria completa	2 días?	jue 09/01/14	vie 10/01/14
25	4.5 - Realitzar presentació	1 día?	sáb 11/01/14	dom 12/01/14
26	4.6 - Entrega de la memòria final	1 día?	lun 13/01/14	lun 13/01/14



4. Persistència d'Objectes

4.1. Introducció

Com a introducció a aquest projecte, presentem una sèrie de conceptes que utilitzarem al llarg del treball, i que ens serviran per construir el sistema de persistència. Explicarem els conceptes bàsica sobre tot lo relacionat amb la persistència d'objectes i més detalladament sobre la capa de persistència i també parlarem sobre les definicions de les diferents tecnologies actuals que tenim alhora de treballar amb la capa de persistència.

4.2. Capa de persistència

Les aplicacions que es desenvolupen actualment estructuren i guarden la informació en sistemes de gestió de dades amb l'objectiu de què puguin ser reutilitzats, o bé des de la mateixa aplicació o des d'una aplicació diferent. La capa de persistència és la peça que ens permet guardar, recuperar, actualitzar o eliminar l'estat dels objectes que poden ser persistents en un o més sistemes gestors de dades.

El sistemes gestors de dades poden ser un sistema OODBMS (Objected-Oriented Database Management System), un sistema RDBMS (Relational Database Management System) o qualsevol altre sistema, de manera que el programador no hagi de realitzar cap traducció dels objectes per convertir-los en persistents, sinó que sigui la capa la que s'encarregui d'aquesta transformació.

En el cas dels RDBMS, que són els sistemes gestors de dades més estesos, utilitzant un model relacional, i normalment els programes d'ordinador solen usar la programació orientada a objectes, que és molt diferent del model relacional, i per tan, és necessari un component de software que permeti la comunicació d'aquests dos instruments.

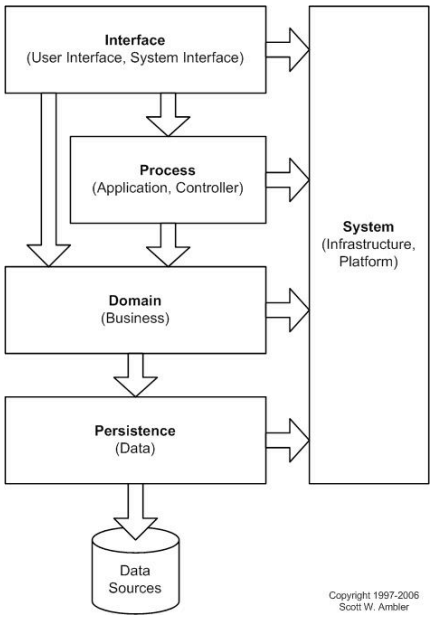
La capa de persistència tradueix entre els dos models de dades: de registres a objectes i d'objectes a registres. Quan el programa vol gravar un objecte crida el motor de persistència, que tradueix l'objecte registres i crida a la base de dades perquè es guardi aquests registres. De la mateixa manera, quan el programa vol recuperar un objecte, la base de dades recupera els registres corresponents, els quals son traduïts en format objecte per guardar objectes i recuperar objectes, com si estigues programat per una base de dades orientada a objectes. La base de dades sols veus que guarda i recupera registres, com si el programa estigues dirigint-se a ella de manera racional.

A aquesta última tècnica de programació es diu ORM (Objected-Relational Mapping). Un ORM és una capa que permet relacionar objectes amb un model de dades relacional, d'aquesta manera s'oculta tot el mecanisme de connexió al motor de gestió de la base de dades. També

ens estalvia d'haver d'escriure les sentències SQL necessàries per realitzar les consultes i/o modificacions als registres de la base de dades.

4.3. Objectius d'una capa de persistència

Un cop definit el concepte de capa de persistència, hem de veure quins són els objectius bàsics d'una capa. Scott Ambler en el seu estudi "The Design of a Robust Persistence Layer For Relational Databases", diu que aquests objectius son:



- Diferents tipus de mecanismes de persistència: Un mecanisme de persistència és qualsevol tecnologia que es pot utilitzar per guardar permanentment objectes per la posterior modificació, lectura o esborrar en qualsevol tipus de base de dades. Tan poden ser bases de dades relacionals, bases de dades orientades a objectes, bases de dades jeràrquiques, xml, etc.
- Encapsulació total del mecanisme de persistència: Només necessitarem enviar al mecanisme de persistència els missatges de guardar, esborrar, recuperar per guardar, esborrar o recuperar un objecte del sistema de persistència. Aquest s'encarrega de totes les operacions necessàries per dur a terme aquestes accions de manera transparent.
- Accions sobre múltiples objectes: És comú necessitar recuperar varis objectes a la vegada per generar un informe o el resultat d'una cerca per el que la capa de persistència haurà de permetre treballar sobre múltiples objectes. De la mateixa manera, es necessitarà treballar sobre múltiples objectes si volem esborrarlos.
- Transaccions: La capa de persistència ha de suportar transaccions, és a dir, ha de tenir la capacitat de realitzar diferents operacions sobre un mateix objecte o sobre una llista d'objectes, de manera que controli que totes les operacions són correctes o és

necessari realitzar un roll-back de les mateixes deixant l'objecte com estava originalment abans de començar les operacions.

- Extensibilitat: Al igual que hauríem de ser capaços d'afegir noves classes a les aplicacions, hauríem de ser capaços de substituir els mecanismes de persistència utilitzats i poder desenvolupar-lo amb noves funcionalitats.
- Identitat dels objectes persistents: La capa de persistència ha d'associar un identificador a cada objecte persistent, que serveixi per localitzar de forma unívoca, la imatge de l'objecte guardat en el sistema de gestió de dades.
- Cursors: La capa de persistència ha de recuperar el número d'objectes que se li demanen, ja que manejar grans quantitats de dades pot arribar a ser inviable.
- Múltiples arquitectures: La capa de persistència ha de poder-se adaptar a qualsevol tipus d'arquitectura existent, sigui client/servidor o de diferents capes.
- Diferents bases de dades: Ha de suportar diferents tipus de base de dades, de forma que si es canvia la base de dades, les aplicacions no es vegin afectades.
- Múltiples connexions: La capa de persistència ha de permetre obrir connexions a diferents bases de dades dins duna mateixa aplicació.
- Consultes SQL: Normalment les consultes SQL no estaran escrites dins del codi d'una aplicació, però a vegades és necessari fer-ho. La capa de persistència haurà de suportar llançar codi SQL directament des del codi de l'aplicació.

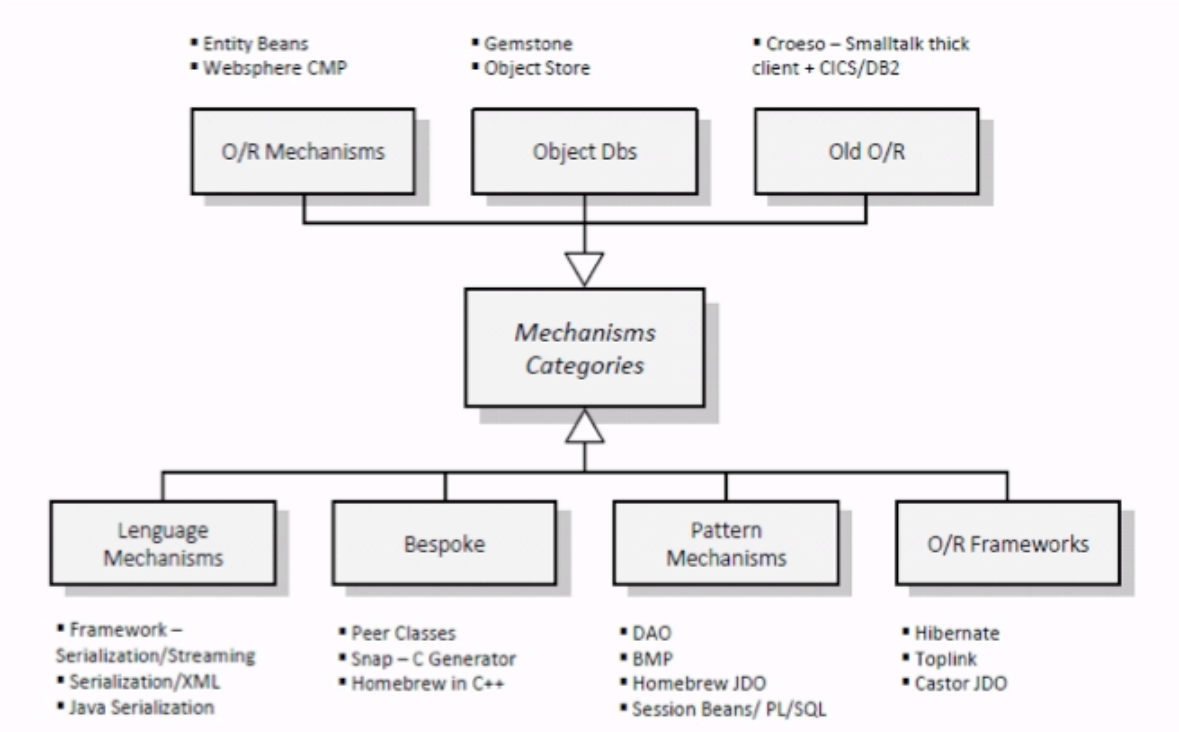
4.4. Models de sistema de persistència

La necessitat de la persistència dels objectes ha aconseguit al llarg del temps que s'hagin desenvolupat diferents sistemes amb la finalitat d'aconseguir aquest objectiu. Els sistemes més primitius van constituir la primera pedra a un procés que ha permès donar pas a sistemes més evolucionats, com son les capes de persistència: en l'últim esgraó evolutiu dins de l'escala de la persistència d'objectes. Quasi s'utilitzen tots els diferents sistemes de persistència que existeixen actualment.

Els principals sistemes d'emmagatzemat de dades amb els que es treballa habitualment son les bases de dades. Entre les bases de dades es distingeixen dos tipus fonamentals: relacionals i orientades a objectes. A més, tenim la forma de base de dades objecte-relacional, mescla dels dos sistemes de bases de dades. Altres sistemes que existeixen que ens permeten guardar dades, son els sistemes de fitxers, com poden ser fitxers ASCII o XML.

Altres sistemes que també tenen la seva importància son els que treballen amb arquitectures d'objectes distribuïts com EJB, i les bases de dades XML

Dins de l'evolució dels sistemes de persistència d'objectes s'ha treballat sobre tots els diferents sistemes d'emmagatzemat. Els primers principalment sobre fitxers i bases de dades relacionals. Els més evolucionats, que son les capes de persistència, ens permeten a més treballar sobre sistemes distribuïts, i es comença a parlar sobre treballar sobre bases de dades XML.



No obstant, és convenient destacar que dins de la diversitat de sistemes de persistència (tal i com podrem veure en la següent taula), només les capes de persistència estan orientades, tal i com diu Ambler, cap a múltiples sistemes d'emmagatzemat, fitxer, diferents bases de dades, etc.

La següent taula enumera alguns dels diferents sistemes i capes de persistència. Totes elles son comparades segons els criteris descrits per Ambler. Com podrem veure, els únics sistemes de persistència d'objectes que compleixen la majoria de les característiques son les capes de persistència.

Requeriment	**Serialització**	**JDBC-ODBC**	**ODBMS**	**EJB**	**JDO**	**ODMG**	**ADO**
Diferents tipus de mecanismes	Sistemes de fitxers	SGBDR	SGBDOO	SGBDR, EAI, sistemes fitxers, etc...	Sistema de fitxers SGBDR, SGBDOO, EAI, atres	Sistema de fitxers SGBDR, SGBDOO, EAI, atres	SGBDR
Encapsulació completa dels mecanismes	Si	Si	Si	Si	Si	Si	Si
Accions multiobjecte	No	Si	Si	Si	Si	Si	Si
Transaccions	No	Si	Si	Si	Si	Si	Si

Extensibilitat	No	No	No	No	Si	Si	Només per tipus d'objecte
Identificadors dels objectes	Si, Manual	Si, Manual	Si	Si	Si	Si	Si
Cursors	No	Si	No	Si	Si	Si	Si
Proxys	No	No	Si	Si	Si	Si	Si
Registres	Si, Manual	Si, Manual	Si	Si	Si	Si	Si
Suport per múltiples arquitectures	No	No	No	No	Si	No	No
BBDD diferents fabricants	No	Si	No	Si	Si	Si	Si
Connexions múltiples	No	No	Si	Si	Si	No	No
Ús controladors natius o externs	No	No natius	No	No natius	No natius	No natiu	No natius
Consultes SQL	No	Si	Si	Si	Si	No	Si

4.5. Conceptes de persistència d'objectes

4.5.1. Instància persistent i transitòria

Una instància persistent és aquella en la qual les seves dades perduraran en l'execució del procés que ha materialitzar la instància. Una instància transitòria o temporal, és tota aquella instància la qual les seves dades desapareixen quan finalitzen els processos que la manipulen. En els dos sentits, les instàncies en si, com a estructures de dades residents en memòria, desapareixen al finalitzar els processos que les han creat.

Un petit exemple: imaginem l'execució d'un programa que sol·licita introduir el nostre nom que serà utilitzat repetides vegades en diferents operacions. Si aquesta dada és recollida per una instància transitòria, quan finalitzi el programa i el tornem a executar, haurem d'introduir la dada de nou; però si està associada a una instància persistent, la dada introduïda podrà ser recuperada i mostrada en successives execucions del programa.

4.5.2. Servei de persistència d'objectes

El concepte de servei de persistència és un tema vigent de debat en investigació i desenvolupament, en el món acadèmic i en la industria. Acotar amb nitidesa què un servei de persistència és una qüestió oberta, perquè la separació entre aquest concepte i el de la base de dades és difús. Aquí, assumint que el destí final de les dades seran principalment els sistemes de gestió de dades de l'empresa, és adoptada la següent excepció en la major part dels textos:

"El servei de persistència és un sistema o mecanisme programat per possibilitar una interfície única per l'emmagatzemat, recuperació, actualització i eliminació de l'estat dels objectes que poden ser persistents en un o més sistemes gestors de dades"

La definició anterior considera que el sistema gestor de dades pot ser un sistema SGBDR, un sistema SGBDOO, o qualsevol altre sistema; que l'estat podria estar repartit sobre varis sistemes, inclús de diferents tipus; i el més important, que en un servei de persistència d'objectes persistents, a més del bolcat i recuperació de l'estat en els sistemes gestors de dades. I tot això, hauria de ser efectuat d'acord a la definició feta abans de persistència, sense la necessitat de traducció explicita per part del programador. En tots els casos, sigui quin sigui el tipus de gestor de dades, els serveis de persistència d'objectes faciliten la il·lusió de treballar amb un sistema de base de dades d'objectes integrat amb el llenguatge de programació, ocultant les diferències entre el model d'objectes del llenguatge i el model de dades del sistema utilitzat com a base de dades. Tot i el que s'ha dit, un servei de persistència no és un sistema de gestió de base de dades orientat a objectes. El servei de persistència és un component essencial de tot els sistema gestor de base de dades d'objectes (SGBDOO), que resol altres aspectes, a més de la persistència.

De les alternatives estàndards per fer persistir els objectes en Java només ODMG 3.0 i JDO poden tenir la consideració del servei de persistència d'objectes Java. També és possible proposar utilitzar un servei de persistència del OMG, PSS 2.0, per persistir objectes Java, però aquest estàndard considera opcionals les transaccions i la persistència transparent, que son funcions necessàries per oferir un servei de persistència eficient, conforme amb les definicions vistes de persistència. JDBC, SQL necessiten que el programador defineixi i implementi les operacions de persistència tenint en compte com convertir aquests objectes en tuples. Ambdós poden ser utilitzats en la construcció de serveis de persistència per a base de dades relacionals. La serialització és el servei de persistència més bàsic, només ofereix serveis per guardar i recuperar l'estat d'un objecte sobre fitxers i fluxes d'entrada/sortida.

Per l'altre part, els SGBD ofereixen serveis de persistència, que adopten una o vàries de les següents aproximacions de com es pot fer un objecte persistent:

- Per tipus. Un objecte pot arribar a ser persistent quan és creat d'algun tipus (classe) dotada de la capacitat de persistir o d'un subtipus (classe descendent) d'aquests. Els tipus persistents son diferents dels tipus els quals les seves instàncies son transitòries. El tipus podria ser identificat com a persistent en la seva declaració, o per herència d'algun dels tipus predeterminats per el sistema. De forma semblant, la serialització obliga que les classes implementin la interfície Serializable.

- Per invocació explicita. El programador invoca un mètode que provoca que un objecte passi a ser persistent. La trucada al mètode pot ser la creació de l'objecte o en qualsevol moment, segons la seva implementació.
- Per referència. Un objecte és fet persistent al ser referenciat per un altre que és persistent. Afegir un objecte a una col·lecció persistent, l'assignació d'un objecte a un atribut d'un altre persistent o l'assignació a certs tipus de referències, provoquen que un objecte passi de transitori a ser persistent.

4.5.3. Persistència ortogonal

Dues característiques seran ortogonals si l'ús d'una no afecta a l'altre, és a dir, son independents entre si. Programes i persistència seran ortogonals, si la manera en la que els objectes son manipulats per aquests programes és independent a la utilització de la persistència, eu els mateixos mecanismes operaran tan sobre objectes persistents com sobre objectes transitoris, les dues categories serien tractades de la mateixa manera amb independència de la seva característica de persistència. Ser persistent hauria de ser una característica intrínseca de l'objecte, suportada per la infraestructura de l'entorn de programació i persistència. La persistència d'un objecte ha de ser ortogonal a l'ús, tipus i identificació. És a dir, qualsevol objecte hauria de poder existir el temps que sigui precís, ser manipulat sense tenir en compte, si la duració de la seva vida, supera el procés que el va crear, i la seva identificació no estigui vinculada al sistema de tipus, com la possibilitat de donar noms als objectes.

A l'hora de plasmar l'ús de la persistència en els nostres programes, una persistència ortogonal ideal, portaria a no haver de modificar el codi de les nostres classes, excepte aquelles on haguem d'introduir operacions que provoquin la persistència per qualsevol objecte que sigui duraré. Els beneficis que aporta la persistència ortogonal son importants: majores cotes de facilitat de manteniment, correcció, continuïtat del codi i productivitat de desenvolupament. S'aconsegueix:

- Menys codi. Una semàntica per expressar les operacions de persistència més simple d'utilitzar i entendre. Evita la duplicitat de codi, un preparat per instancies transitòries i l'altre per instancies persistents.
- Evitar la traducció explicita entre l'estat dels objectes i la seva representació a la base de dades, que redunda en major facilitat de manteniment i menys codi.
- Facilitat la integritat i permetre que actuï el sistema de tipus subjacents, que automàticament podria verificar la consistència i la correspondència de tipus, entre estats a la base de dades i objectes en el programa, la integritat no seria responsabilitat del programador.

Tot el que hem vist en aquest apartat apunta a la conveniència d'utilitzar persistència ortogonal. En la pràctica, aconseguir persistència ortogonal completa no és fàcil, habitualment ens trobem limitacions. Hi haurà classes d'objectes que no son suportades per els serveis de persistència, bé per compromisos de disseny, com la dificultat d'implementació; bé perquè cal pensar que determinats objectes no tenen sentit fora del context d'execució concret d'un procés, com per exemple un port de comunicacions per IP.

4.5.4. Tancament de persistència

Els objectes solen referenciar a altres objectes, aquests a la seva vegada a altres, i així pot continuar successivament. Cada objecte pot tenir un gran nombre d'objectes dependents de manera directa i indirecta. Aquesta relació de dependències és part integrant de l'estat de cada objecte. Quan l'estat d'un objecte és guardat o recuperat, les seves dependències també haurien de ser guardades o recuperades. És a dir, quan l'objecte sigui recuperat, arribaria a estar incomplet, seria inconsistent respecte el que s'havia guardat.

Un mecanisme de persistència possibilita la persistència automàtica de les dependències d'un objecte, que hauran de persistir, es diu que admet el tancament de persistència. Quan l'estat d'un objecte és guardat, els estats dels objectes dependents que hagin de ser persistents, son també guardats, i així, successivament. És a dir, en la recuperació de l'estat d'un objecte, els estats dels objectes dependents son recuperats. El tancament de persistència determina el conjunt de referencies necessari, que ajuda a aconseguir les consistència entre l'estat de l'objecte en l'instant de guardar i l'estat resultant de la seva recuperació.

4.5.5. Persistència en profunditat

La persistència en profunditat és el procés de convertir automàticament en persistent tot objecte referenciat directe o indirectament per un objecte persistent, els objectes de tancament de persistència d'un objecto son fets persistents. És l'aplicació recurrent de l'estratègia de persistència per referència.

4.5.6. Transparència de dades

Quan un sistema o entorn de programació ofereix transparència de dades, el conjunt de les classes persistents i l'esquema de la base de dades és un, les classes defineixen de fet l'esquema a la base de dades. Els estats emmagatzemats en la base de dades son manipulats amb el llenguatge de programació escollit, no és necessari cap altre. Els objectes son recuperats de la base de dades automàticament, quan les referencies a aquests son accedides. També, les modificacions de l'estat d'objectes persistents són reflectides a la base de dades automàticament. Els estats dels objectes son recuperats i actualitzats de manera transparent; no hi ha canvis en la semàntica de referencia o d'assignació a l'aplicació. Objectes transitoris i persistents són manipulats d'igual forma. Les operacions pròpies de la persistència son efectuades sense la intervenció directa del programador, amb més codi. La frontera entre el llenguatge de programació i els serveis de dades desapareix als ulls del programador, evitant la falta de correspondència entre la base de dades i el llenguatge de programació.

No hem de confondre transparència amb persistència ortogonal, la primera és la conseqüència de la segona. Podem trobar transparència de dades sense disposar de persistència ortogonal total, és el cas de què determinades classes no puguin persistir, això concretament suposa que l solució no seria ortogonal, independent, respecte el tipus.

4.5.7. Falta de correspondència entre classes i dades

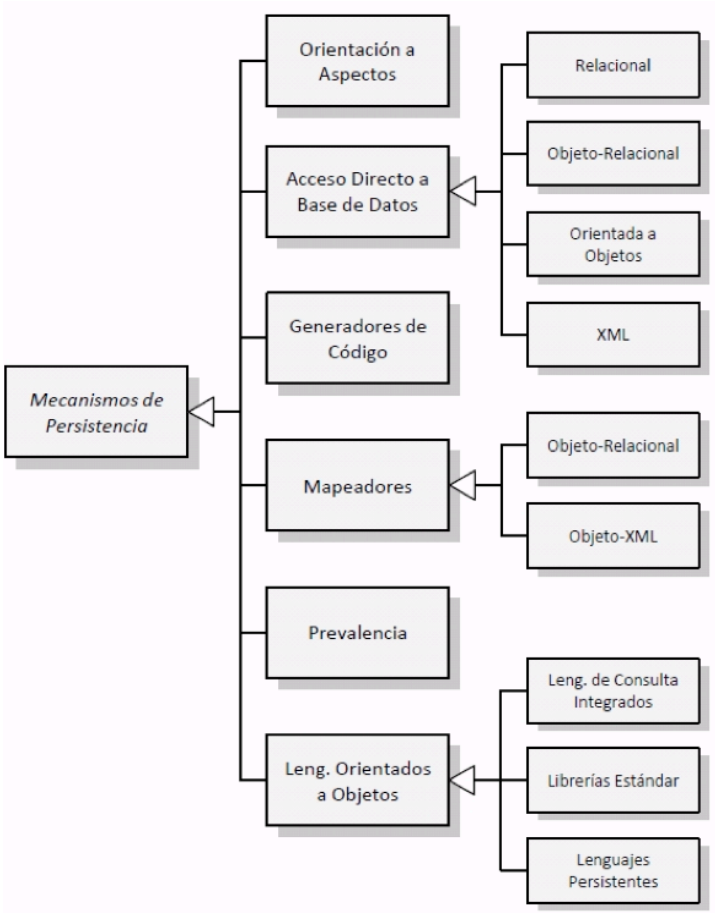
Quan es treballa amb sistemes gestors de dades, com bases de dades relacionals, fitxers, bases de dades documentals XML, etc., el qual el seu model de dades no té una equivalència directa amb el model d'objectes del llenguatge de programació utilitzat, hi ha una falta de correspondència entre la base de dades i el llenguatge de programació, és necessari establir un model de correspondència, que defineixi com una classe es converteix en dades sobre el model ofert per sistema que allotjarà l'estat dels objectes. A aquesta equivalència entre la classe i les dades, es denomina aquí correspondència classe – dades(object mapping). En el cas particular de la correspondència entre classe i taules en un SGBDR, és la correspondència objecte – registres. Es pot utilitzar el terme mapeig per senyalar el procés de definició de la correspondència entre les classes dels objectes persistents i els models de dades, el procés pot implicar canvis en els dos cantons de la correspondència, en el model de classes creat o modificat per assumir certs models de dades, i el contrari, el model de dades pot ser dissenyat o canviat, per permetre una correspondència més eficient i eficaç amb certs models de dades.

Les definicions de persistència que s'han vist inviten a utilitzar una persistència que no canviï la manera de treballar amb el llenguatge de programació, que actuï de forma transparent i consistent, on el model de classes i el model de dades son la mateixa cosa, i que sigui una persistència ortogonal. Els beneficis seran un codi amb menys línies, més fàcil de mantenir, més correcte i productiu.

4.6. Mecanismes de persistència

Qualsevol aplicació que necessiti del processament de dades pot ser dividida a termes generals en dos nivells: dades persistents i el processament dels mateixos. Els mecanismes utilitzats per accedir i interactuar amb les dades son de vital importància, no sols per el seu impacte en el desenvolupament final del sistema, sinó també des dels punts de vista com poden ser el manteniment, disseny, escalabilitat i reutilització.

La tècnica que permet resoldre la persistència d'objectes en el desenvolupament d'un sistema orientat a objectes es considera "Mecanisme de persistència".



Existeixen nombroses alternatives a l'hora de seleccionar un mecanisme per accedir a les dades i posteriorment tractar-les. Nombrem quines son aquestes alternatives i els fets característics de cadascuna d'elles.

4.6.1. Accés directe a base de dades

El mecanisme implica l'ús directe de la base de dades per implementar la persistència del sistema. Aquest últim es refereix al accés a les dades utilitzant per exemple una interfície estandarditzada en conjunt amb algun llenguatge de consulta suportat directament per el sistema de gestió de la base de dades. En aquest cas, no existeix cap tipus de capa entre l'aplicació i el sistema de gestió de base de dades utilitzat.

4.6.2. Mapeig

Son aquells mecanismes que es basen 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. El mapeig és l'encarregat de convertir les dades entre dos paradigmes diferents i fer la conversió entre ells.

4.6.3. Generador de codi

En el desenvolupament d'aplicacions empresarials existeixen tasques repetitives que el programador es veu obligat a realitzar. En aquest cas, es poden utilitzar eines que generin codi correcte i basat en patrons per resoldre la persistència del sistema, permeten al desenvolupador centrar-se en el codi que resol la lògica del negoci.

4.6.4. Orientació a aspectes

La Programació Orientada a Asptectes (POA) és un paradigma de programació relativament recent i que la seva intenció és permetre una adequada modularitat de les aplicacions i possibilitar una millor separació de conceptes. Gràcies a POA es poden encapsular diferents conceptes que composen una aplicació en entitats ben definides, eliminant les dependències entre cadascun dels mòduls. D'aquesta manera s'aconsegueix raonar millor sobre els conceptes, s'elimina la dispersió de codi i les implementacions resulten més comprensibles i adaptables.

4.6.5. Llenguatge orientat a objectes

Es tracta de resoldre la persistència de dades utilitzant funcionalitats previstes per el propi llenguatge de programació evitant la inclusió de frameworks o altres eines independents del llenguatge i persistint les dades utilitzant fitxers plans o fitxers XML usant tècniques de serialització.

4.6.6. Esquemes de prevalença

Son marcs de treball que mantenen els objectes de memòria volàtil i periòdicament utilitzen esquemes de serialització d'objectes per guardar una imatge dels objectes de l'aplicació. És un apropament molt simple i ràpid a la persistència tot i que presenta alguna problemes d'escalabilitat. Es tracta d'una manera diferent d'utilitzar base de dades per aconseguir la persistència.

4.7. Equivalència classe-taules relacionals

Veurem el model relacional de dades (que és el més estès avui dia), i en el qual ens centrarem, i també veurem com persistir els objectes d'aquest tipus en aquest tipus de model. Com que el framework que desenvoluparem es basarà en bases de dades relacionals (en el nostre cas MySQL) i els motors de persistència més coneguts (iBATIS, Hibernate, etc,) funcionen amb aquests models. El problema el trobem a l'hora de fer correspondre objectes en relacions o viceversa, i de com trobar una solució al problema de la correspondència classe-taula.

Classe i taula son dos conceptes diferents, utilitzades per modelar les realitats que després manipulen les aplicacions. Una classe defineix els atributs i el comportament de la realitat que modela, defineix objectes. Una taula especifica les dades de l'entitat que representa, concreta una relació entre valors dels dominis de les dades triades. Els dos capturen característiques d'interès per l'aplicació, però la taula ignora aquelles característiques lligades al comportament d'execució, cosa que si modela la classe. Per la classe, la persistència és una característica més, en canvi, per la taula té el sentit essencial de servir al propòsit de persistir i compartir les dades que modela. Per tan, no existeix una equivalència exacta entre els dos conceptes.

Model d'objectes	**Model relacional**
Identitat	Clau primària
Classe	Relació (Taula, Vista)
Instància (Objecte)	Fila
Atribut d'instància persistent amb domini d'un tipus bàsic o primitiu	Columna
Atribut persistent referència a objectes de tipus conegut	Clau forana
Atribut persistent del tipus col·lecció	No hi ha equivalència directa, ha de ser una relació, si la cardinalitat és (m,n), una entitat d'associació. Si és (1,n) llavors atributs en les relacions corresponents
Atribut persistent de classe	Una taula amb les característiques comunes a totes les instàncies d'
Herència	Una relació afegida amb tots els atributs d'una instància de la jerarquia herència completa o múltiples taules

5. Frameworks de persistència

5.1. Introducció

Un framework representa una arquitectura de software que modela les relacions que existeixen entre les entitats del domini i el mateix temps proporciona una metodologia i una estructura de treball que es pot estendre a les aplicacions de domini que en alguns casos puguin utilitzar-los.

Un framework d'aplicacions és un terme utilitzat per referir-se a un conjunt de biblioteques o classe utilitzades per implementar la estructura estàndard d'una aplicació. Dit d'una altre manera, és un esquema o patró per el desenvolupament o implementació d'una aplicació completa, o d'una part específica d'ella.

Es podria entendre per aquesta definició que hem donat, que un framework és una simple llibreria de classes. Existeixen vàries diferències entre una llibreria de classes i un framework que s'exposen en la següent taula.

Característica	Frameworks	Llibreries de Classe
Model de l'aplicació	Els frameworks són l'aplicació, de manera que manipulen el flux de control	L'usuari ha de crear el control de l'aplicació i la lògica per invocar els components
Estructura de l'aplicació	Múltiples frameworks cooperant	Una sola aplicació monolítica consisteix en un conjunt de llibreries de classes
Obtenció de serveis	Els frameworks son el servei	Heretant funcions de llibreries de classe
Creació del sistema	Els frameworks invoquen el codi de l'usuari. Aquest codi pot derivar parts del framework	Derivant o creant noves classes
Granularitat del control	Mitja. L'usuari sols pot derivar algunes parts del framework	L'usuari pot derivar qualsevol classe de llibreria
Abstracció dels serveis	Alta. Oculten la seva complexitat. Automatitzen les característiques	L'usuari ha de determinar quins mètodes ofereix la llibreria i en quin ordre han

	estàndards i ofereixen un mecanisme d'excepcions	de ser invocats
Quantitat de nou codi a implementar	Molt poc	Mitjà
Cost de manteniment	Baix	Mitjà
Reducció de complexitat	Molta. L'usuari escriu petites porcions de codi	L'usuari ha de crear noves classes per cada una de les aplicacions que desenvolupi del programa
Temps necessari de desenvolupar una aplicació	Baix	Depèn del grau de reutilització que es pugui realitzar de la llibreria
Reutilització del codi	Molt alt	Alt

Els avantatges d'utilitzar un framework son les següents:

- Infraestructura prefabricada: Els frameworks redueixen la codificació i sobretot la posta en marxa, ja que proporcionen subsistemes que sabem que ja funcionen. En definitiva, proporcionen codi que no es tindrà de mantenir ni reescriure.
- Proporcionen una arquitectura: Qualsevol programador que treballi amb un framework no haurà d'invertir una gran part del seu temps en buscar classes necessàries, interconnectar-les o descobrir els mètodes que contenen. Els frameworks oculten tota aquesta complexitat donat un nivell alt d'abstracció.
- Reducció del manteniment
- Bona base per la industria de components: Els frameworks ben dissenyats, permeten que terceres companyies puguin subministrar components o parts de components que els usuaris finals podran afegir.

Els desavantatges son:

- Limitació de flexibilitat: Els components que un usuari construeixi a partir d'un framework han de residir dintre les restriccions imposades per l'arquitectura del framework.
- Dificultat d'aprenentatge: L'usuari ha d'estudiar què proporciona el framework i com s'ha d'utilitzar. Aquest aprenentatge és difícil el principi, però el desenvolupament serà més fàcil.
- Reducció de la creativitat: Pot succeir que l'usuari no pugui canviar el comportament d'una classe del framework si aquesta no permet que els seus serveis puguin reescriure's o redefinir-se.

5.2. Frameworks de persistència

Els frameworks de persistència introdueixen tots els avantatges comentats a l'apartat anterior aplicats a la comunicació de les aplicacions i el sistema de persistència. En el cas del llenguatge de programació Java, la persistència de dades ens els sistemes relacionals ve facilitada per el mapeig Objecte/Relacionals (ORM), que permet enllaçar un llenguatge orientat a objecte amb una base de dades relacional.

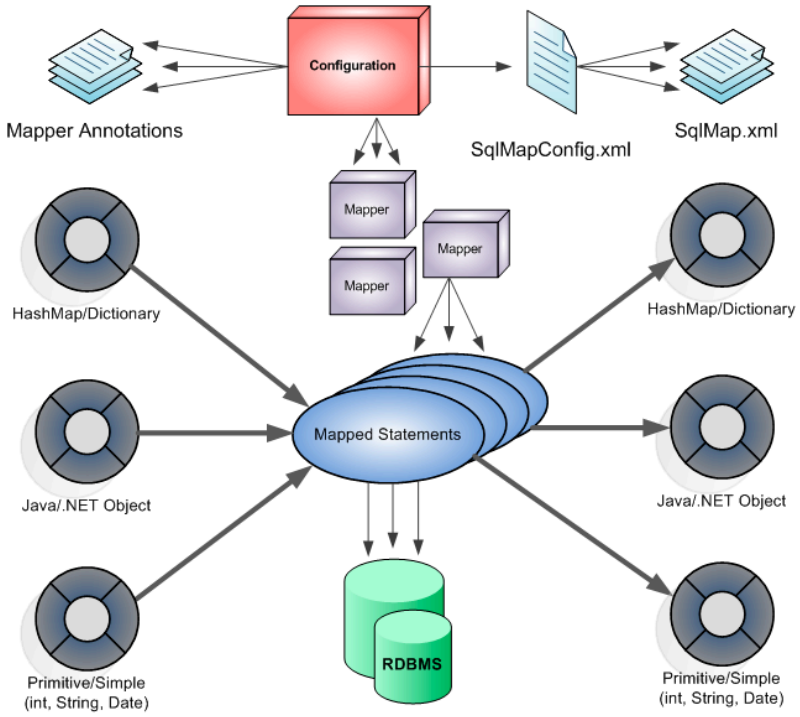
Aquests frameworks permeten als desenvolupadors mantenir una perspectiva orientada a objectes i centrar-se a la programació de la lògica del negoci, encarregant-se d'automatitzar les tasques al connectar-se a la base de dades i recuperar, guardar o esborrar la informació de la base de dades i convertir-la en objectes, amb independència de la base de dades que estem utilitzant.

En aquest apartat ens centrarem en les alternatives de frameworks de persistència per el llenguatge de programació Java que és el més utilitzat en el desenvolupament d'aplicacions informàtiques basades en JDBC o les característiques més similars dels mateixos.

5.2.1. iBATIS

iBATIS SQL Maps és un framework opensource sobre JDBC que proporciona un mitjà simple i flexible de moure dades entre un objecte Java i una base de dades relacional. Aquesta plataforma ajuda a reduir la quantitat de codi Java que és necessari per accedir a una base de dades relacional. El framework permet relacionar una Java Bean a una sentència SQL mitjançant l'ús d'arxius XML que ens permeten crear consultes complexes, ja que en aquest arxiu XML és on escriurem les sentències SQL d'acord amb el sistema de gestió de base de dades. Per tan, si es canvia de base de dades, serà necessari reescriure les consultes, ja que iBATIS no és independent del proveïdor de la base de dades.

Per aquesta última raó, iBATIS no és un ORM pur, ja que es tenen que escriure les sentències SQL necessàries per interactuar amb la base de dades.



D'aquesta manera, iBATIS permet escriure manualment els comandos SQL i relacionar-los amb un objecte Java. Una vegada relacionada la capa de persistència a un model d'objecte, llavors ja es pot utilitzar dins de qualsevol aplicació Java. Per tan, ja no és necessari buscar fonts de dades, obtenir connexions, crear les sentències sql ni interpretar els resultats obtinguts, o guardar les dades en memòria. iBATIS ens fa tot això per un.

5.2.2. Hibernate

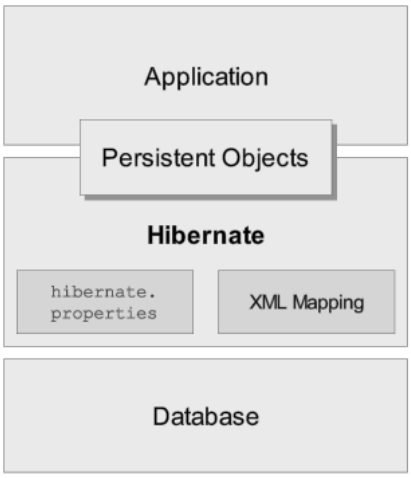
Hibernate és una eina ORM per Java. Hibernate permet al desenvolupador fer objectes persistents, seguint les propietats del llenguatge Java tals com associació, herència, polimorfisme, composició i tot el que son Java Collections.

Hibernate no sols prové relacions entre classes Java i taules de bases de dades (i de tipus de dades Java a tipus de dades SQL), sinó també proveeix la recuperació de dades a través d'un mecanisme propietari de consulta anomenat HQL (Hibernate Query Language). Tot això permet reduir els temps de desenvolupament de manera dràstica evitant la manipulació manual de SQL i JDBC.

L'objectiu final d'Hibernate és poder permetre al desenvolupador reduir fins un 95% les tasques comunes de persistència que es poden automatitzar.

Hibernate existeix sota la llicència LGPL la qual és suficientment flexible per permetre utiltizar aquesta eina tan en projectes comercials com en projectes de codi obert.

L'arquitectura d'alt nivell d'Hibernate es descriu a la figura següent.



Hibernate fa ús de classes POJO (Plain Old Java Object) Java en conjunt amb fitxers XML que permeten relacionar aquests objectes amb la capa de la base de dades. En lloc d'utilitzar processament de byte code o generació de codi, Hibernate utilitza en temps d'execució una característica de Java anomenada reflexió, la qual permet realitzar introspecció d'un objecte en temps d'execució.

Hibernate no és restringit en l'ús de cap tipus de dades Java ni objectes no primitius. Tots aquests tipus es poden relacionar, inclús classes que pertanyen a la plataforma Java Collections. Aquestes es poden associar amb valors o associacions a altres entitats en bases de dades. En Hibernate existeix una propietat especial que és el camp id el qual representa l'identificador o la clau primària de la classe.

Els objectes que es vulguin fer persistents es defineixen en un document de mapeig (que és un arxiu XML), i que permet descriure els camps persistents amb les seves respectives associacions, així com també les subclasses que posseeix l'objecte en qüestió. Aquests documents son compilats en el moment d'inicialització i proveeixen a l'eina de la informació necessària per cada classe.

Addicionalment, Hibernate és capaç de generar esquemes de bases de dades a partir del model de classes i inclús crear classes Java a partir del model de dades.

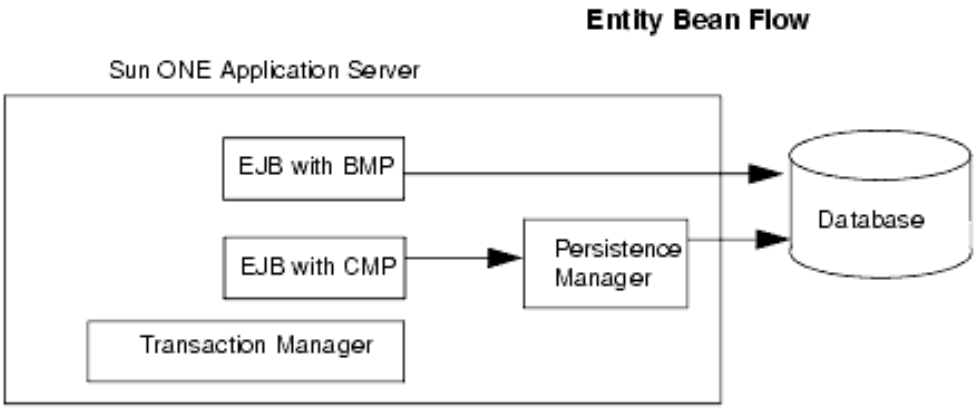
5.2.3. EJB d'entitat

Els EJB d'entitat son components que representen les dades persistents del model de negoci de l'aplicació. Es tracte d'una representació Java, en memòria i en forma d'objecte de la informació emmagatzemada en algun medi persistent.

Dins dels EJB d'entitat, hi ha dues maneres diferents de gestionar la persistència: si deixem que sigui el desenvolupador el que especifiqui el comportament persistent, estarem parlant de EJB d'entitat amb persistència gestionada pel component (BMP, Bean Managed Persistence); si és el contenidor el que gestiona la persistència, estarem parlant de EJB d'entitat amb persistència gestionada per el contenidor (CMP, Container Managed Persistence).

En els EJB d'entitat amb persistència gestionada pel component, el desenvolupador del component és el responsable de programar la persistència del component. Això implica que el desenvolupador del component ha de codificar el codi per tal que permeti mantenir sincronitzades les dades el EJB i les dades de la font de dades. Des del punt de vista del desenvolupador, la persistència gestionada per el component requereix més esforç que la persistència gestionada per el contenidor, ja que comporta haver d'escriure explícitament la lògica de la persistència en la classe EJB.

En els EJB d'entitat amb persistència gestionada pel contenidor, el desenvolupador no ha de programar la persistència del component. El contenidor implementa la gestió de la persistència segons les definicions que proporciona el desenvolupador al descriptor del component.



El desenvolupador del component ha de definir els camps i les relacions persistents del component que es vol desenvolupar, i el contenidor s'encarrega de generar el codi de gestió de la persistència segons aquesta definició en el temps de desplegament del component.

Els EJB d'entitat tenen mètodes Select i Finder que serveixen per realitzar cerques d'aquests. Si el EJB utilitza persistència gestionada per el component, aquests mètodes de cerca es programen amb codi JDBC estàndard, però si s'utilitza persistència gestionada per el contenidor es defineixen de forma declarativa en el descriptor. Per definir de forma declarativa el comportament dels EJB d'entitat amb persistència gestionada per el contenidor, l'especificació de EJB defineix un llenguatge molt similar al SQL, anomenat EJB QL.

El EJB QL s'escriu com el XML en els descriptors de desplegament i permet als programadors descriure el comportament de mètodes de consulta d'una forma abstracta, fent d'aquestes consultes transportables a bases de dades de diferents fabricants.

5.3. Comparació frameworks

Un cop hem descrit les tres tecnologies més importants comparem les tres tecnologies i així podrem descriure en quin cas utilitzar un tipus de tecnologia o una altre.

Característiques	iBATIS	Hibernate	JPA
Simplicitat	Excel·lent	Bo	Bo
Solució ORM completa	Normal	Excel·lent	Excel·lent
Adaptable a canvis en el model de dades	Bo	Normal	Normal
Complexitat	Excel·lent	Normal	Normal
Dependència SQL	Bo	Normal	Normal
Rendiment	Excel·lent	Excel·lent	-
Portabilitat a diferents motors de base de dades	Normal	Excel·lent	-
Portabilitat a plataformes diferents de Java	Excel·lent	Bo	-
Documentació i suport	Normal	Bo	Bo

iBATIS

- És un framework de persistència que ens ofereix els beneficis del SQL però evita la complexitat de JDBC
- Fomenta l'ús directe del SQL i assegura que tots els beneficis del SQL siguin anul·lats per el propi framework
- La simplicitat és el seu major avantatge, ja que ens proporciona un mapeig senzill i la capa API ens pot servir per construir codi d'accés a dades
- Permet que el model de dades i el model d'objectes sigui independents un de l'altre

Quan utilitzar iBATIS:

- És la millor opció quan es necessita un control complet del SQL
- També és útil quan s'han de realitzar consultes SQL més complexes

Quan no utilitzar iBATIS:

- No s'ha d'utilitzar quan es té control total tan de l'aplicació com de la base de dades, ja que en aquest cas l'aplicació pot modificar l'estructura de la base de dades, o viceversa
- És inapropiat per les base de dades no relacionals, ja que aquest tipus de bases de dades no suporten les transaccions o altres característiques clau que utilitza iBatis

Hibernate

- És de codi obert, és una solució lleugera de mapeig objecte-relacional
- Inclou un llenguatge de consulta molt poderós anomenat Hibernate Query Language, que és molt semblant a SQL i a més conté algunes característiques extres
- HQL suporta moltes característiques avançades de paginació i perfils dinàmics que SQL no suporta
- HQL no necessita de cap "join" explícit quan treballem amb múltiples taules
- Hibernate fa un mapeig senzill objecte-relació mitjançant l'assignació dels metadates en un fitxer XML que defineix la taula a la base de dades que ha de ser assignat a una classe particular

Quan utilitzar Hibernate:

- És la millor opció per aprofitar el mapeig. Proporciona una solució ORM completa, però deixa el control sobre les consultes
- És la millor opció per els programadors orientats a objectes que estan menys familiaritzats amb SQL

JPA

- L'API Java Persistence és el mapeig estàndard objecte-relacional i la interfície de gestió de persistència per la plataforma Java EE 5. Com a part de l'especificació EJB 3, està suportat per la gran majoria de proveïdors Java
- JPA és un model de persistència estàndard POJO per ORM. És una part de l'especificació EJB 3 i substitueix els Beans d'entitat
- JPA utilitza anotacions metadates i/o fitxers descriptors XML per configurar el mapeig entre els objectes Java de l'aplicació i de les taules en una base de dades relacional
- Defineix un llenguatge similar de consultes SQL (JPQL) que és diferent del EJB-QL, que és el llenguatge que utilitza els Beans d'entitat.

Quan utilitzar JPA:

- JPA s'ha d'utilitzar quan es necessita una solució estàndard de persistència basada en Java
- APP recolza l'herència i el polimorfisme, ambdues característiques de programació orientada a objectes

Quan no utilitzar JPA:

- Per emmagatzematge en memòria cau, que no està clarament definit en APP, però està ben recolzat per Hibernate
- JPA està definit per treballar només amb bases de dades relacionals

6. Disseny del framework de persistència

6.1. Introducció

Tal i com hem vist i estudiat en les diferents assignatures d'enginyeria del programari, la solució del model de classes està basada en els patrons SOLID d'enginyeria del programari. Amb tot el que s'ha estudiat, podem dir que l'aplicació que es construeix està basada en:

- Patró Singleton: el patró de disseny singleton (instància única) està dissenyada per a restringir la creació d'objectes pertanyents a una classe o 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 (SessionFactory, ConfigurationFactory i ConnectionFactory) i en el SQLHelper que realitzarà la traducció dels POJOs a registres de bases de dades.
- Don't Repeat Yourself (DRY): 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 i per tan, en cas de modificacions no hi haurà problemes de replicacions.
- Inversion of Control (IOC): El principi d'inversió de control consisteix en què 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 altres classes.
- Simple Responsability Principle (SRP): Cada classe només disposarà d'una única responsabilitat. D'aquesta manera, es facilita la reutilització dels components.
- Interface Segregation Principle (ISP): Una classe X que té una dependència d'una classe Y no ha de dependre dels mètodes de la classe Y que no hagi d'utilitzar.
- Open Closed Principle (OCP): 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 l'aplicació sense impacte en el codi. Per exemple si es vol afegir més motors de base de dades només caldrà modificar el fitxer de configuració i desenvolupar les noves implementacions, per tan, pel codi aquest canvi serà transparent.

6.2. Diagrama de casos d'ús

Tindrem els següents actors al sistema:

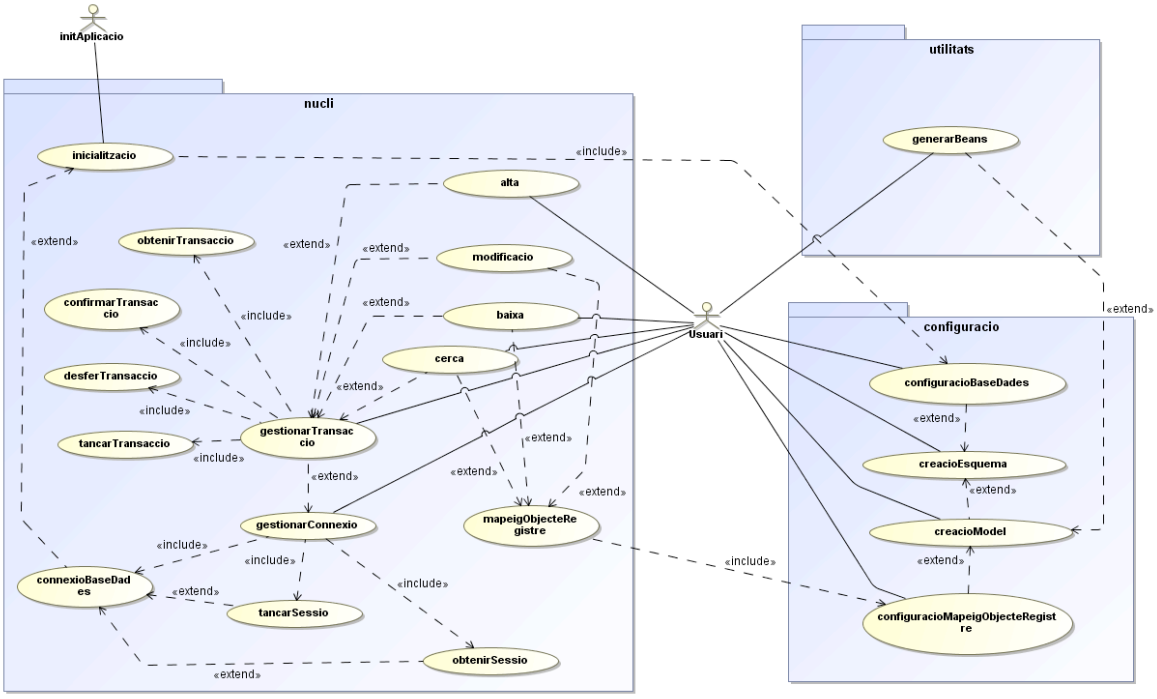
- Usuari: Correspon a l'usuari que utilitzarà el framework per abstreure's de la complexitat inherent a la capa de persistència de les dades. Generalment correspondrà al rol d'un del desenvolupador de programari.

- 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 l'executi.

Agruparem els diferents casos d'ús en tres paquets en relació a la base de la funcionalitat que descriuen:

- configuracio: Correspon a la funcionalitat de configuració del framework. Aquesta configuració estarà diferenciada en dos nivells diferents, i per tan, 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 (les dades de connexió). Finalment, es disposarà d'un segon cas d'ús en el que l'usuari especificarà els mapejos que existiran entre els beans d'accés a la base de dades i les taules de la base de dades. Aquesta configuració es realitzarà mitjançant el mecanisme d'Annotacions que ofereix la plataforma Java a partir de la versió 1.5.
- nucli: 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 i que permetrà realitzar les operacions d'alta, baixa, modificació i cerca, així com la gestió d'aquestes operacions (gestió de transaccions i sessió).
- utilitats: 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 d'accés al model de dades 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 mapeig de SQL a Java.

Després de descriure l'estructura, ens queda el diagrama de casos d'ús següent:



Tot seguit realitzarem la descripció de cadascuna de les funcionalitats del diagrama de casos d'ús anterior

6.2.1. configuracio

configuracioBaseDades

La seva funcionalitat és configurar les dades de connexió que utilitzarà el framework per a comunicar-se amb la base de dades

Flux principal:

- L'usuari crea el fitxer de configuració (bdconfig.properties) a la ubicació corresponent
- L'usuari omple les dades de connexió en la propietat corresponent
    - Introdueix la URL de la base de dades
    - Introdueix l'usuari de la base de dades on es disposarà del model
    - Introdueix la contrasenya de l'usuari per accedir a la base de dades

configuracioMapeigObjecteRegistre

La seva funcionalitat és configurar el mapeig entre els beans de base de dades que s'utilitzaran a l'aplicació i les taules disponibles al model

Flux principal:

- L'usuari crea un POJO que representa una taula del model
    - Especifica la taula a la que referencia mitjançant l'anotació @Taula
    - Crea un atribut a l'objecte per a especificar una columna de la taula a la que referencia mitjançant l'anotació @Columna
    - Es repeteix el pas anterior per a cadascuna de les columnes de la taula
- Es repeteix el primer pas per cadascuna de les taules del model que es vol utilitzar a l'aplicació

creacioEsquema

La seva funcionalitat és crear l'usuari sobre el que es crearà el model de dades

Flux principal:

- Es crea l'usuari de la base de dades amb la contrasenya d'accés

creacioModel

La seva funcionalitat és crear el model a la base de dades

Flux principal:

- Accedim a l'esquema amb les credencials en el que es crearà el model
- S'executa l'script de creació del model

Flux alternatiu:

- Si les credencials no són correctes, es genera un erro d'accés
- L'script no és consistent (hi ha qualsevol error), es genera un error

6.2.2. nucli

alta

La seva funcionalitat és afegir el registre a una de les taules del model de dades

Flux principal:

- Crear una nova instància del POJO que representa la taula a la que es volen afegir dades
- Afegir el contingut dels atributs de la instància que es vol afegir a la taula
- Executar el mètode create disponible a l'objecte Sessió passant per paràmetre el POJO instanciat
- Obtenir el resultat de l'alta

Flux alternatiu:

- En cas de voler gestionar la transacció, requerirà obtenir de la sessió una nova transacció especificant el setAutocommit a fals. En cas contrari, el sistema executarà la operació de manera automàtica obtenint la transacció i tancant-la al finalitzar la operació
- En cas d'error, es generarà una excepció indicant el motiu de l'error

baixa

La seva funcionalitat és esborrar el registre existent a una de les taules del model de dades

Flux principal:

- Crear una nova instància del POJO que representa la taula de la que es volen eliminar dades
- Obtenir la referència del registre que es vol eliminar
- Executar el mètode delete disponible a l'objecte Sessio passant per paràmetre el POJO instanciat
- Obtenir el resultat de la baixa

Flux alternatiu:

- En cas de voler gestionar la transacció, requerirà obtenir de la sessió una nova transacció especificant el setAutocommit a fals. En cas contrari, el sistema executarà la operació de manera automàtica obtenint la transacció i tancant-la al finalitzar la operació
- En cas d'error, es generarà una excepció indicant el motiu de l'error

modificacio

La seva funcionalitat és modificar els valors d'un registre existent a una de les taules del model de dades

Flux principal:

- Crear una nova instància del POJO que representa la taula de la que es volent modificar dades
- Obtenir una referència del registre que es vol modificar
- Afegir el nou contingut dels atributs de la instància que es vol modificar
- Executar el mètode update disponible en l'objecte Sessio passant per paràmetre el POJO instanciat
- Obtenir el resultat de la modificació

Flux alternatiu:

- En cas de voler gestionar la transacció, requerirà obtenir de la sessió una nova transacció especificant el setAutocommit a fals. En cas contrari, el sistema executarà la operació de manera automàtica obtenint la transacció i tancant-la al finalitzar la operació
- En cas d'error, es generarà una excepció indicant el motiu de l'error

cerca

La seva funcionalitat és obtenir una sèrie de registres que compleixin amb unes determinades condicions de cerca

Flux principal:

- Crear una nova instància del POJO que representa la taula (o la cerca)
- Obtenir una nova sessió a través de l'objecte Sessio
- Executar el mètode find disponible a l'objecte Sessio passant per paràmetre el POJO instanciat
- Obtenir els resultats de la cerca

Flux alternatiu:

- Si el POJO que es passa per paràmetre no és un bean de la base de dades, es genera un error
- En cas d'error, es generarà una excepció indicant el motiu de l'error

mapeigObjecteRegistre

La seva funcionalitat és realitzar la traducció del codi Java introduït pel desenvolupador al codi SQL que interpreta la base de dades

Flux principal:

- L'usuari realitza una operació (CRUD)
- El sistema recupera la configuració de la base de dades per a poder realitzar el mapeig en funció del motor de base de dades que tinguem configurat
- El sistema accedeix a la taula gràcies el mapeig dels objectes implicats en la operació
- El sistema tradueix la operació en el codi SQL del motor de base de dades configurat
- El sistema executa la consulta
- El sistema recupera les dades dels registres i els transforma a POJO
- El sistema retorna les dades

Flux alternatiu:

- Error en la recuperació de la configuració, es genera una excepció amb l'error específic
- Error en la traducció del codi Java a SQL, es genera una excepció detallant l'error
- Error a l'executar la consulta, es genera una excepció detallant l'error
- Error en la traducció del codi SQL a Java, es genera una excepció detallant l'error

gestioTransaccio

La seva funcionalitat és gestionar les operacions de base de dades en els casos que ho requereixi. Inclou els casos d'ús d'obtenció, confirmació, cancel·lació i tancament de la transacció

Flux principal:

- L'usuari sol·licita una nova transacció mitjançant l'objecte Sessio (obtenirTransaccio)
- Un cop realitza les operacions necessàries, les confirma (confirmarTransaccio)
- L'usuari tanca la transacció (tancarTransaccio)

Flux alternatiu:

- En cas d'error es desfan els canvis (desferTransaccio)

obtenirTransaccio

La seva funcionalitat és obtenir una nova transacció per a poder realitzar una sèrie d'operacions

Flux principal:

- L'usuari sol·licita una nova transacció mitjançant l'objecte Sessio especificant el setAutocommit a fals
- El sistema accedeix a la base de dades i crea una nova sessió
- El sistema retorna la nova sessió creada

Flux alternatiu:

- En cas de no disposar d'una sessió activa, en crea una de nova
- En cas d'error es genera una excepció, detallant l'error

confirmarTransaccio

La seva funcionalitat és fer efectives totes les modificacions realitzades en una transacció

Flux principal:

- L'usuari recupera una transacció mitjançant l'objecte Sessio
- L'usuari confirma les modificacions realitzades mitjançant l'objecte SessioFactory

Flux alternatiu:

- En cas d'error és genera una excepció, detallant l'error

desferTransaccio

La seva funcionalitat és 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:

- L'usuari recupera una transacció mitjançant l'objecte Sessio
- L'usuari desfà els canvis realitzats en la transacció mitjançant l'objecte SessionManager

Flux alternatiu:

- En cas d'error es genera una excepció, detallant l'error

tancarTransaccio

La seva funcionalitat és tancar la transacció creada

Flux principal:

- L'usuari recupera una transacció (obtenirTransaccio)
- L'usuari tanca la transacció

Flux alternatiu:

- En cas d'error es genera una excepció, detallant l'error

gestioSessio

La seva funcionalitat és gestionar les connexions a la base de dades. Inclou els casos obtenirSessio, tancarSessio i connexioBaseDades

Flux principal:

- L'usuari obté una nova sessió (obtenirSessio)
- Realitza les operacions necessàries
- Tanca la sessió (tancarSessio)

Flux alternatiu:

- En cas d'error, es genera l'excepció corresponent i es realitza una traça de l'error

obtenirSessio

La seva funcionalitat és obtenir una nova sessió

Flux principal:

- L'usuari sol·licita una sessió a través del mètode corresponent de la classe SessionFactory
- El sistema es connecta a la base de dades
- El sistema retorna una nova sessió

Flux alternatiu:

- En cas d'error es genera una excepció, detallant l'error

tancarSessio

La seva funcionalitat és tancar una sessió oberta

Flux principal:

- L'usuari recupera una sessió (obtenirSessio)
- L'usuari tanca la sessió

Flux alternariu:

- En cas d'error es genera una excepció, detallant l'error

connexioBaseDades

La seva funcionalitat és connectar amb la base de dades

Flux principal:

- El sistema obté les dades del fitxer de configuració
- El sistema carrega la configuració i realitza la connexió amb el motor de base de dades

Flux alternatiu:

- En cas d'error es genera una excepció, detallant l'error

inicialitzacio

La seva funcionalitat és inicialitzar el framework amb les dades de configuració per a poder obtenir el datasource associat al pool de connexió

Flux principal:

- L'aplicació carrega les dades de configuració i executa el mètode d'inicialització del framework
- El sistema connecta amb la base de dades (connexioBaseDades)
- El sistema inicialitza totes les estructures de dades del framework
- Retorna el datasource

Flux alternatiu:

- En cas d'error es genera una excepció, detallant l'error

6.2.3. utilitats

generarBeans

La seva funcionalitat és generar les classes Java relacionades amb les taules de l'esquema de la base de dades de la configuració. D'aquesta manera, es dóna l'opció a l'usuari de no realitzar la pesada tasca de relacionar el model de dades amb una classe Java 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 configuracio

Flux principal:

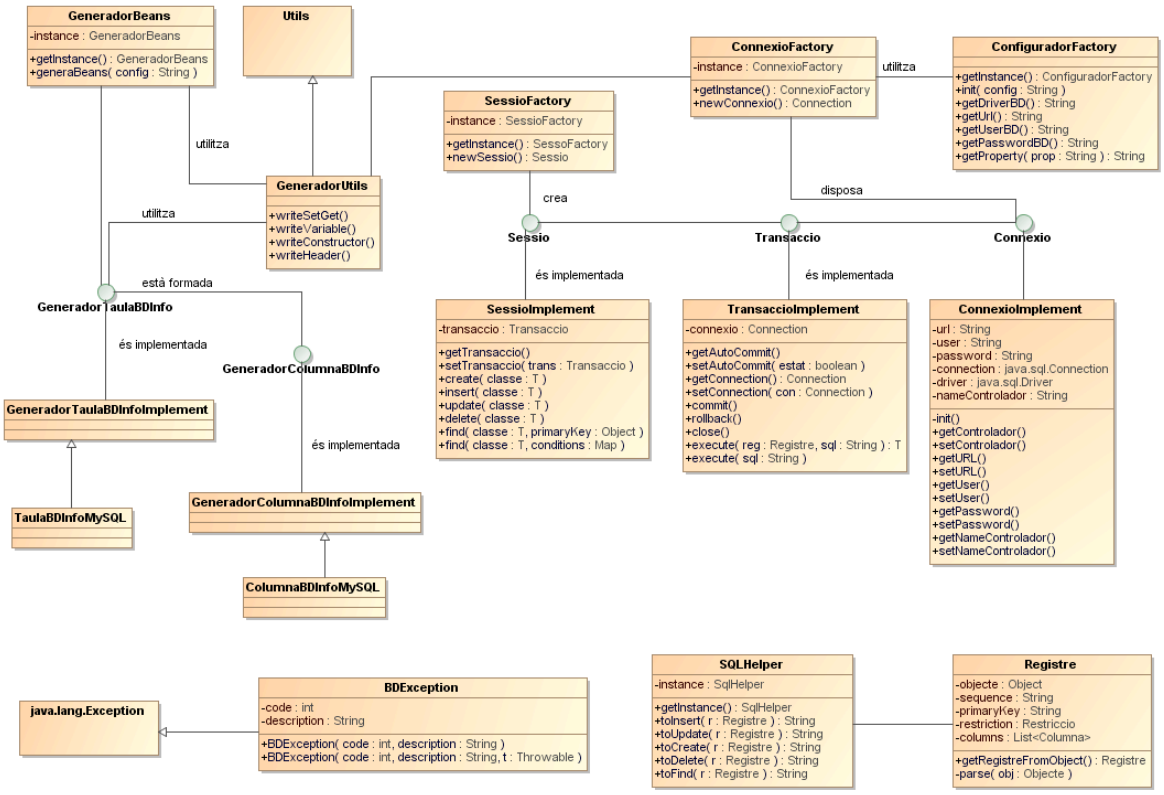
- L'usuari especifica la configuració on es generaran les classes Java
- L'usuari sol·licita la generació dels beans que relacionaran les taules amb la classe
- El sistema connecta amb la base de dades (connexioBaseDades)
- El sistema genera les classes Java relacionades amb cadascuna de les taules del model en la ubicació especificada per l'usuari

Flux alternatiu:

- En cas d'error es genera una excepció, detallant l'error i s'atura l'execució

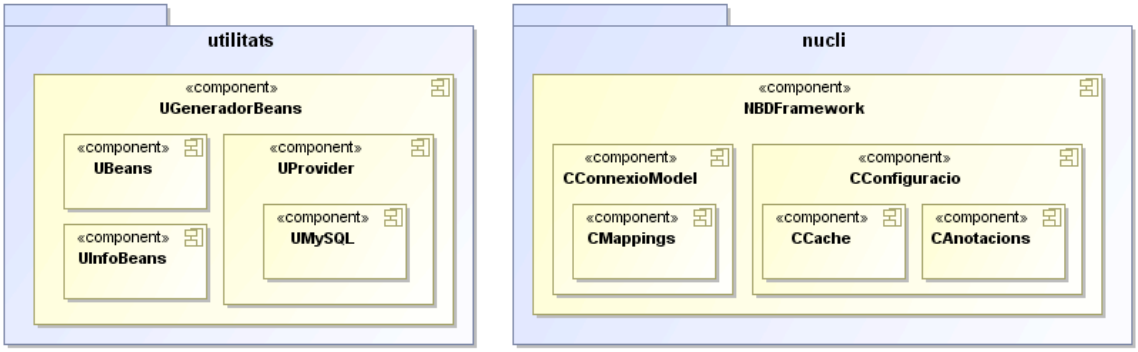
6.3. Diagrama de classes

A continuació es mostra el diagrama de classes amb els mètodes més rellevants del disseny en funció dels patrons i principis descrits en l'apartat anterior



6.4. Diagrama de components

En aquest punt realitzarem el detall del diagrama de components de la solució implementada. Els components del sistema sense refinar són els següents

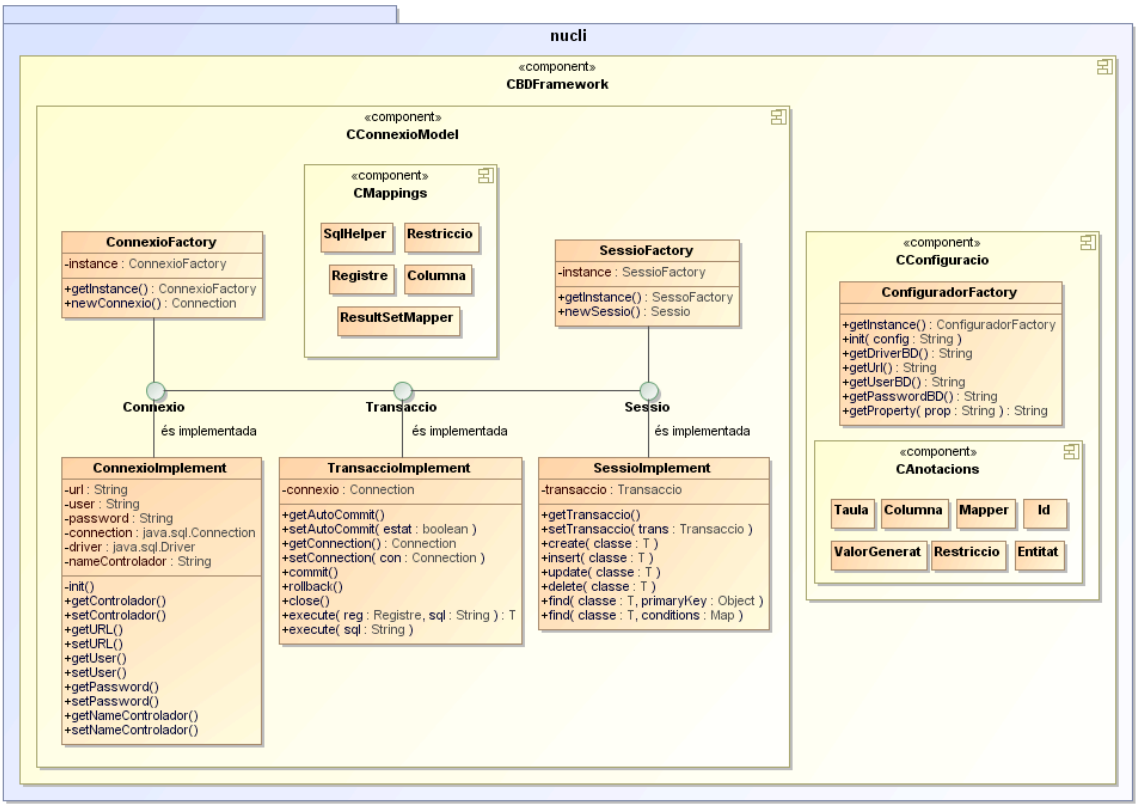


6.4.1. nucli

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

CConnexioModel és el codi encarregat d'abstreure al desenvolupador de la complexitat de la persistència. Així doncs, la seva responsabilitat serà la d'accedir al model de dades i realitzar les lectures o 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 mapeig entre la base de dades i el codi Java.

CConfiguracio és el component encarregat de la configuració del framework, és a dir, disposarà de les estructures de dades necessàries per la configuració del framework. En aquest sentit, disposarà d'un component CAnotacions, en el que s'han creat Annotations Custom per a poder crear el mapeig de beans a SQL. Finalment disposa de la classe estàtica encarregada de la lectura de les dades del fitxer de configuració.



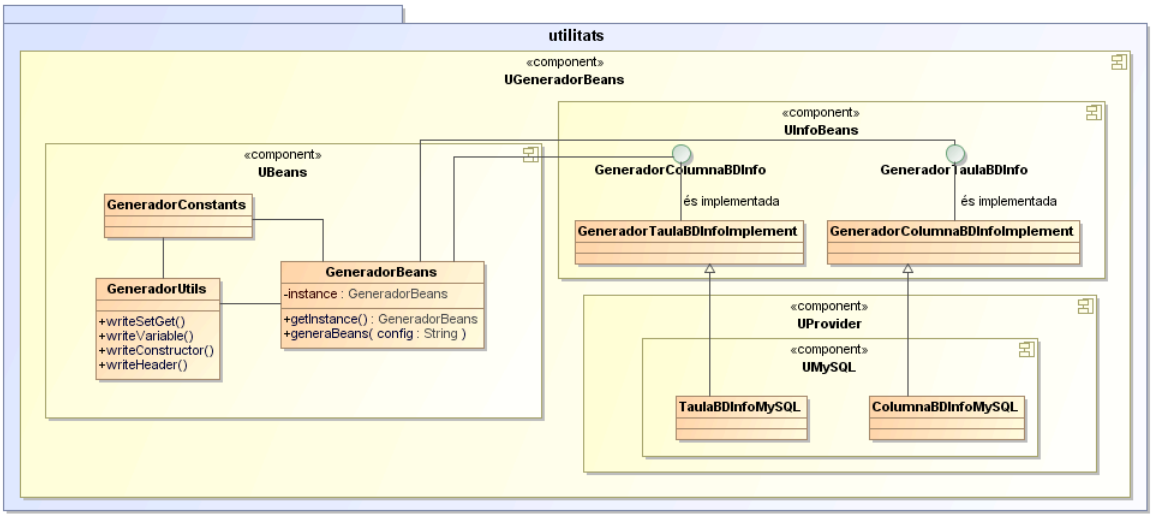
6.4.2. utilitats

El component d'utilitats està format per tres subcomponents: UBDBeans, UBInfoBeans i UProvider.

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-lo, utilitzarà el mòdul de configuració del nucli que li permetrà accedir al model i obtenir les metadates de cadascuna de les taules i columnes del motor. Aquest mètode haurà de crear POJOs al directori especificat a la configuració, i per aquest motiu, requerirà mètodes de creació de classes. Aquests mètodes estaran centralitzats en la classe GeneradorUtilis que incorporarà totes les utilitats necessàries per a què 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 què el 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 les columnes (GeneradorColumnaBDInfo) i d'una implementació abstracta amb els mètodes comuns per a totes les implementacions del component UProvider.

UProvider és el component encarregat de la implementació específica del model. Aquest model està format per N subcomponents, un per cadascuna de les implementacions amb les que el framework és compatible. Inicialment, ho serà per MySQL. Així doncs, aquest component disposarem de dues classes (una per la taula i una per la columna) que estenen de les classes abstractes definides al component UBDInfoBeans (GeneradorTaulaBDInfoImplement i GeneradorColumnaBDInfoImplement). Així doncs, aquestes classes pròpies de cada proveïdor incorporaran dades especifiques del motor en qüestió seguint les especificacions de les interfícies definides al component UBDInfoBeans. Aquesta solució segueix el principi OCP i permet afegir més funcionalitats sense impactar en el codi existent.



7. Disseny de l'aplicació d'exemple d'ús del framework

Per tal de poder veure el funcionament d'ús del framework prèviament dissenyat i implementat, realitzarem una descripció de l'anàlisi de l'aplicació que utilitzarà aquest framework. Tanmateix especificarem els requisits, objectius i la descripció del què es pretén aconseguir mitjançant la seva implementació.

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

- Servidor de base de dades correctament instal·lat i arrencat. Per a les proves que es realitzaran en aquest projecte caldrà disposar d'un motor per a poder validar el correcte comportament del framework desenvolupat. En el nostre cas, treballarem amb MySQL.
- Script SQL de base de dades amb el model i l'estructura.
- Servidor d'aplicacions Java en el que desplegar l'aplicació. Utilitzarem Apache Tomcat.
- WAR de distribució (o en el seu defecte els fonts necessaris per a poder generar el distribuïble)

En relació a l'aplicació web, es tracte d'una web on poder gestionar tota la part financera d'una empresa o persona utilitzant el mètode de caixa, és a dir, s'hauran d'introduir les dades financeres (ingressos i despeses) i ens mostrarà les estadístiques econòmiques. A part d'això, també es poden realitzar previsions o crèdits i veure les necessitats futures financeres que tindrem. Serà una aplicació que facilitarà molt a la persona poder veure els resultats financers.

L'usuari que utilitzarà l'aplicació, tindrà la possibilitat de donar d'alta, baixa o modificar els pagaments o cobraments. A part, podrà realitzar una simulació de crèdit bancari i en cas que es vulgui, automàticament convertir-lo a real per així poder veure els futurs resultats. Finalment podrà realitzar una inversió per així veure les futures necessitats econòmiques que pot necessitar. Finalment hi haurà la opció de poder carregar l'extracte bancari, per tal de poder entrar les dades automàticament. L'usuari podrà treure diferents estadístiques sobre la informació financera introduïda.

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 posposarà per l'aplicació web a implementar 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, 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 específiques).

En funció de les visites que disposi la pàgina web, es podrien afegir vàries bases de dades, i per tan, es crearia una quarta capa (servidor d'integració), que s'encarregaria de recollir les dades distribuïdes i de presentar-les a l'aplicació com si estiguessin disponibles a una sola base de dades. Aquest fet, també incrementaria la tolerància a les fallades, donat que en cas de fallada d'una de les bases de dades, el servidor d'integració podria localitzar les dades en una altre d'una manera transparent per l'usuari. D'aquesta manera, es reduirien les caigudes del sistema en "pics" de connexió massiva de clients, fet que podria suposar la pèrdua dels clients.

Finalment, 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 sigui necessari), 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 d'un únic actor que serà l'usuari que utilitzarà el programa en entorn web, que és l'encarregat d'introduir tots les dades amb la finalitat de poder veure els resultats financers.

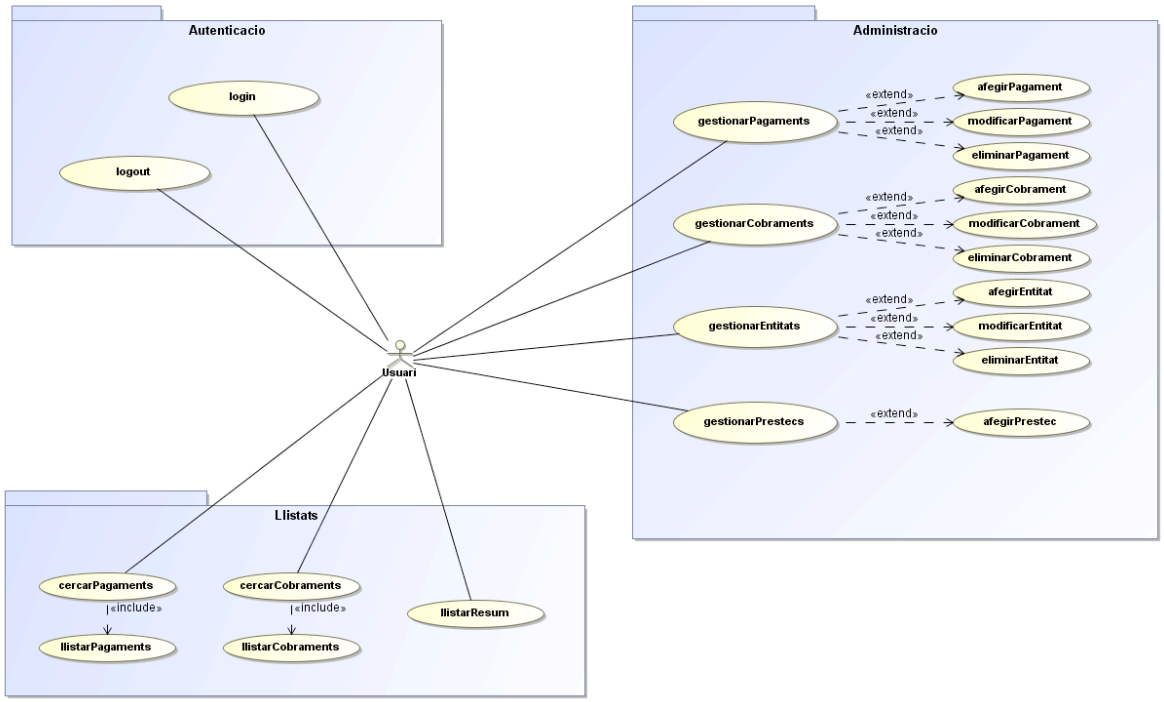
D'altra banda, i de la mateixa manera que s'ha realitzat el framework, s'ha decidit agrupar els casos d'ús en diferents paquets en relació a la base de la funcionalitat que descriuren. Així doncs disposarem dels següents paquets:

- Autenticació: Agrupa els casos d'ús relacionats amb la funcionalitat d'autenticació al sistema. Per tan, es disposarà de casos d'ús: login i logout.
- Administració: És la funcionalitat que correspon a les altes, baixes i modificacions de cobraments i pagaments. Així doncs, 9 casos d'ús afegirCobrament, modificarCobrament, eliminarCobrament, afegirPagament, modificarPagament, eliminarPagament, afegirEntitat, modificarEntitat i eliminarEntitat que seran els que facin ús gestionarCobraments, gestionarPagaments i gestionarEntitats. També

tindrem la gestió dels préstecs, amb el cas d'ús afegirPrestec que serà el que faci ús gestionarPrestecs.

- Llistats: En aquest darrer paquet s'agrupen els casos d'ús que estan relacionats amb les estadístiques. Així doncs, es disposarà dels casos d'ús de llistar estadístiques, cerca i visualització dels pagaments i cerca i visualització dels cobraments.

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



7.2.1. Autenticació

login

La seva funcionalitat és accedir a la part de l'usuari per gestionar les finances

Flux principal:

- L'usuari accedeix a la pàgina web i indica les seves credencials
- El sistema mostra la pàgina d'inici de la gestió de finances

Flux alternatiu:

- En cas d'error es mostrarà un missatge d'error especificant el motiu

logout

La seva funcionalitat és abandonar la part privada de l'usuari de la pàgina web

Flux principal:

- L'usuari selecciona la opció de sortir de la seva part privada
- El sistema redirigeix a la pàgina principal de l'aplicació

7.2.2. Administració

afegirPagament

La seva funcionalitat és incorporar un nou pagament

Flux principal:

- L'usuari accedeix a la pàgina web i indica les seves credencials
- L'usuari selecciona la opció d'afegir un nou pagament
- El sistema mostra un formulari amb les dades que cal omplir
- L'usuari omple les dades amb la informació del nou pagament i selecciona la opció afegir
- El sistema mostra un missatge indicant la correcta execució de la operació

Flux alternatiu:

- En cas d'error es mostrarà un missatge d'error especificant el motiu

modificarPagament

La seva funcionalitat és modificar les dades d'un pagament seleccionat

Flux principal:

- L'usuari accedeix a la pàgina web i indica les seves credencials
- L'usuari selecciona la opció mostrar pagaments amb els filtres corresponents
- El sistema mostra un llistat amb els pagaments segons el filtre
- L'usuari seleccionar el pagament que es vol modificar i selecciona la opció de modificar
- El sistema mostra un formulari per poder modificar les dades del pagament
- L'usuari modifica les dades del pagament i selecciona la opció modificar
- El sistema mostra un missatge indicant la correcta execució de la operació

Flux alternatiu:

- En cas d'error es mostrarà un missatge d'error especificant el motiu

eliminarPagament

La seva funcionalitat és esborrar un pagament seleccionat

Flux principal:

- L'usuari accedeix a la pàgina web i indica les seves credencials
- L'usuari selecciona la opció mostrar pagaments amb els filtres corresponents
- El sistema mostra un llistat amb els pagaments segons el filtre
- L'usuari seleccionar el pagament que es vol esborrar i selecciona la opció de eliminar
- El sistema mostra un missatge de confirmació de la operació
- El sistema mostra un missatge indicant la correcta execució de la operació

Flux alternatiu:

- Si l'usuari cancel·la la operació es redirigeix a la pàgina anterior
- En cas d'error es mostrarà un missatge d'error especificant el motiu

afegirCobrament

La seva funcionalitat és incorporar un nou cobrament

Flux principal:

- L'usuari accedeix a la pàgina web i indica les seves credencials
- L'usuari selecciona la opció d'afegir un nou cobrament
- El sistema mostra un formulari amb les dades que cal omplir
- L'usuari omple les dades amb la informació del nou cobrament i selecciona la opció afegir
- El sistema mostra un missatge indicant la correcta execució de la operació

Flux alternatiu:

- En cas d'error es mostrarà un missatge d'error especificant el motiu

modificarCobrament

La seva funcionalitat és modificar les dades d'un cobrament seleccionat

Flux principal:

- L'usuari accedeix a la pàgina web i indica les seves credencials
- L'usuari selecciona la opció mostrar cobraments amb els filtres corresponents
- El sistema mostra un llistat amb els cobraments segons el filtre
- L'usuari seleccionar el cobrament que es vol modificar i selecciona la opció de modificar
- El sistema mostra un formulari per poder modificar les dades del cobrament
- L'usuari modifica les dades del cobrament i selecciona la opció modificar

- El sistema mostra un missatge indicant la correcta execució de la operació

Flux alternatiu:

- En cas d'error es mostrarà un missatge d'error especificant el motiu

eliminarCobrament

La seva funcionalitat és esborrar un cobrament seleccionat

Flux principal:

- L'usuari accedeix a la pàgina web i indica les seves credencials
- L'usuari selecciona la opció mostrar cobraments amb els filtres corresponents
- El sistema mostra un llistat amb els cobraments segons el filtre
- L'usuari seleccionar el cobrament que es vol esborrar i selecciona la opció de eliminar
- El sistema mostra un missatge de confirmació de la operació
- El sistema mostra un missatge indicant la correcta execució de la operació

Flux alternatiu:

- Si l'usuari cancel·la la operació es redirigeix a la pàgina anterior
- En cas d'error es mostrarà un missatge d'error especificant el motiu

afegirEntitat

La seva funcionalitat és incorporar un nova entitat

Flux principal:

- L'usuari accedeix a la pàgina web i indica les seves credencials
- L'usuari selecciona la opció d'afegir una nova entitat
- El sistema mostra un formulari amb les dades que cal omplir
- L'usuari omple les dades amb la informació de la nova entitat i selecciona la opció afegir
- El sistema mostra un missatge indicant la correcta execució de la operació

Flux alternatiu:

- En cas d'error es mostrarà un missatge d'error especificant el motiu

modificarEntitat

La seva funcionalitat és modificar les dades d'una entitat seleccionada

Flux principal:

- L'usuari accedeix a la pàgina web i indica les seves credencials
- L'usuari selecciona la opció mostrar entitats amb els filtres corresponents
- El sistema mostra un llistat amb les entitats segons el filtre
- L'usuari seleccionar l'entitat que es vol modificar i selecciona la opció de modificar
- El sistema mostra un formulari per poder modificar les dades de l'entitat
- L'usuari modifica les dades de l'entitat i selecciona la opció modificar
- El sistema mostra un missatge indicant la correcta execució de la operació

Flux alternatiu:

- En cas d'error es mostrarà un missatge d'error especificant el motiu

eliminarEntitat

La seva funcionalitat és esborrar una entitat seleccionada

Flux principal:

- L'usuari accedeix a la pàgina web i indica les seves credencials
- L'usuari selecciona la opció mostrar entitats amb els filtres corresponents
- El sistema mostra un llistat amb les entitats segons el filtre
- L'usuari seleccionar l'entitat que es vol esborrar i selecciona la opció de eliminar
- El sistema mostra un missatge de confirmació de la operació
- El sistema mostra un missatge indicant la correcta execució de la operació

Flux alternatiu:

- Si l'usuari cancel·la la operació es redirigeix a la pàgina anterior
- En cas d'error es mostrarà un missatge d'error especificant el motiu

afegirPrestec

La seva funcionalitat és incorporar un nou préstec

Flux principal:

- L'usuari accedeix a la pàgina web i indica les seves credencials
- L'usuari selecciona la opció d'afegir un nou préstec
- El sistema mostra un formulari amb les dades que cal omplir
- L'usuari omple les dades amb la informació del nou préstec i selecciona la opció afegir
- El sistema mostra un missatge indicant la correcta execució de la operació

Flux alternatiu:

- En cas d'error es mostrarà un missatge d'error especificant el motiu

7.2.3. Llistats

cercarPagaments

La seva funcionalitat és cercar els pagaments (en funció d'uns determinats criteris de cerca)

Flux principal:

- L'usuari especifica mitjançant el filtre de la pàgina els criteris que consideri oportuns
- El sistema mostra una pàgina amb tots els pagaments que compleixen els criteris especificats

Flux alternatiu:

- En cas de no haver-hi cap resultat, es mostra un missatge descriptiu

cercarCobraments

La seva funcionalitat és cercar els cobraments (en funció d'uns determinats criteris de cerca)

Flux principal:

- L'usuari especifica mitjançant el filtre de la pàgina els criteris que consideri oportuns
- El sistema mostra una pàgina amb tots els pagaments que compleixen els criteris especificats

Flux alternatiu:

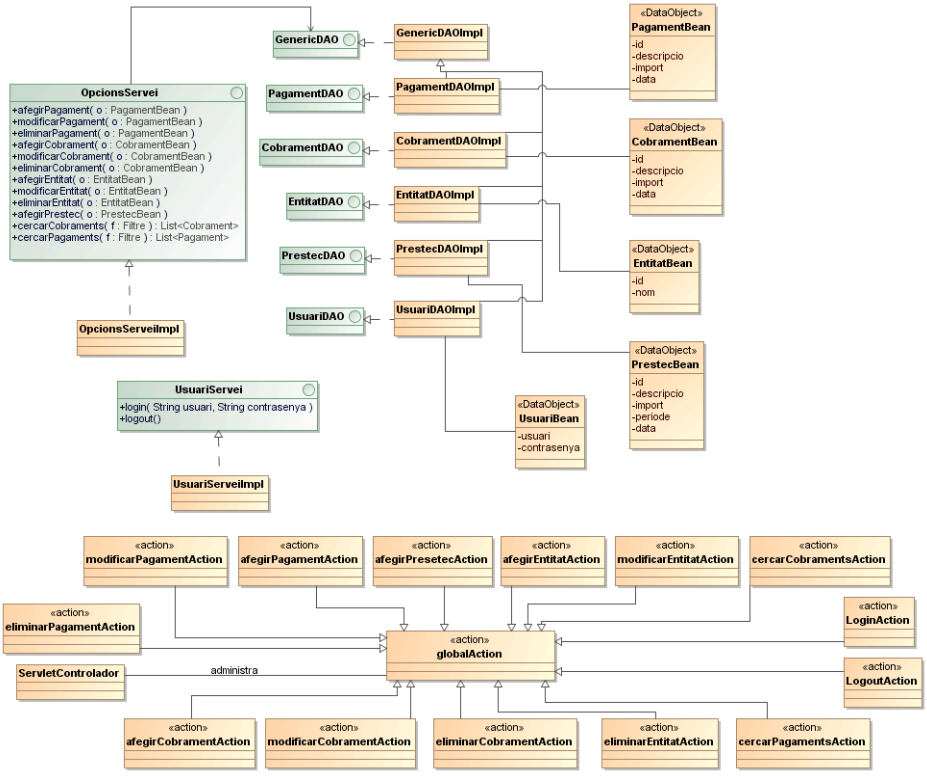
- En cas de no haver-hi cap resultat, es mostra un missatge descriptiu

7.3. Diagrama de classes

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

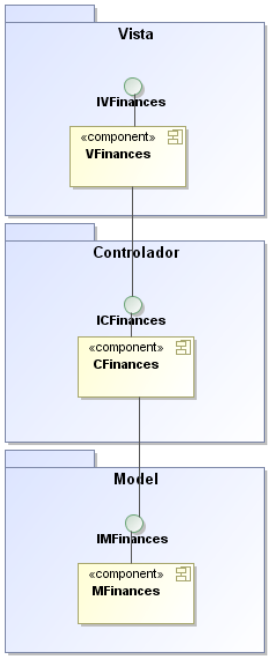
- Patró DAO (Data Acces Object). Amb aquest patró s'aconsegueix separar les responsabilitats de negoci i persistència, i per tan, aportarà flexibilitat a la solució. En aquest cas s'ha optat per utilitzar un GenericDAO amb els mètodes comuns a tots els DAOs, i d'aquesta manera seguir el principi DRY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

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 Finances

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 aquest Model, és propi de la capa de lògica del negoci, podem dividir el component VFinances, en dos components: el component que farà de controlador, FinancesControlador i el que farà de vista, FinancesVista.

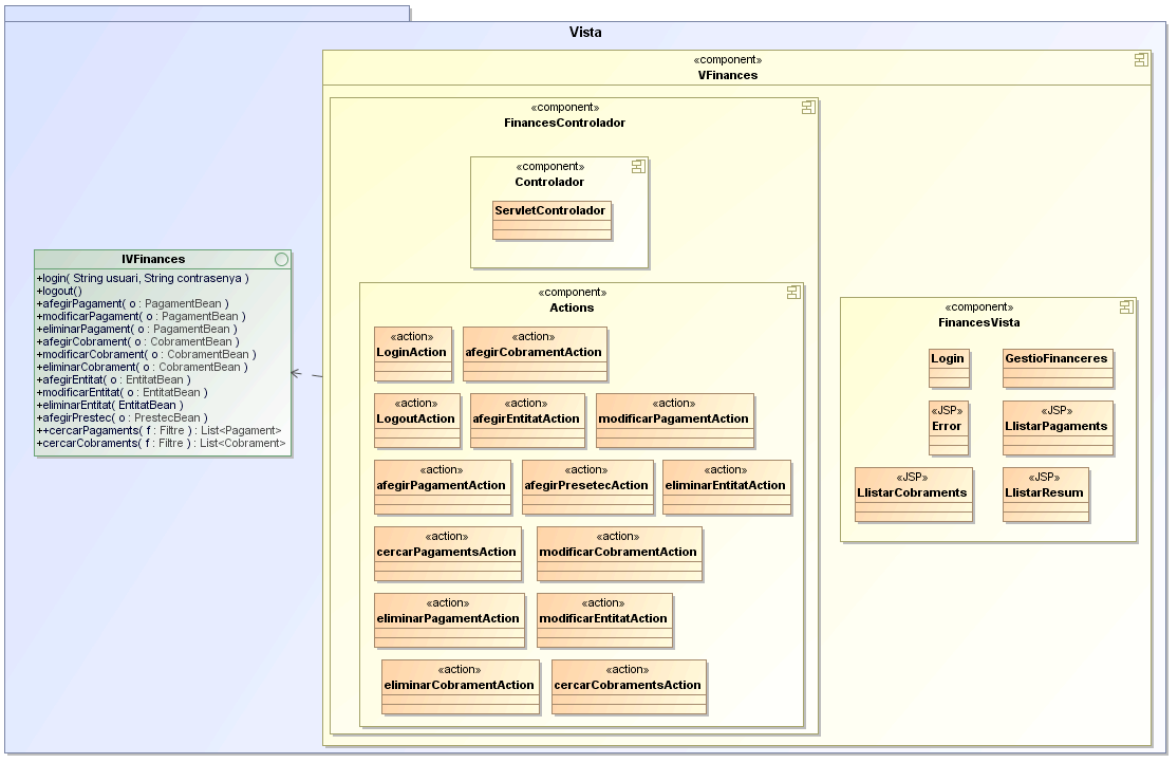
El controlador s'implementarà seguint el patró de disseny Command. Aquest patró emfatitza l'ús d'objectes com accions executables, és a dir, un objecte encapsula una simple acció (mètode) que implementa una interfície, el qual és invocat per un objecte Controlador. Amb aquest patró, s'aconsegueix tenir desacoblada la part de gestió (això ho farà utilitzant el Controller), de les operacions (es farà mitjançant Actions), el que permet afegir al sistema noves funcionalitats de forma més senzilla, i és 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 estaran tipades com a 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 de la pàgina web (gestió financera).
    - Gestió financera: Correspon a la pàgina principal mitjançant la qual es podran realitzar totes les operacions disponibles a la gestió.
    - LlistarPagaments: Vista amb el llistat de pagaments que compleixen un determinat criteri. Així doncs, disposarà d'un formulari per a poder filtrar-les.
    - LlistarCobraments: Vista amb el llistat de cobraments que compleixen un determinat criteri. Així doncs, disposarà d'un formulari per a poder filtrar-les.
    - LlistarResum: Vista amb el llistat de resum de totes les dades econòmiques anualment.
    - 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, els errors controlats de negoci, cadascuna de les vistes disposarà de la seva pròpia gestió d'errors per poder garantir la usabilitat de l'aplicació i facilitar a l'usuari la interacció amb la mateixa.
- Actions
    - LoginAction: Valida que existeix l'usuari i que la contrasenya introduïda sigui la correcta.
    - LogoutAction: Redirigeix a l'usuari a la part pública de la pàgina web.
    - AfegirPagamentAction: Afegeix un nou pagament.
    - ModificarPagamentAction: Modifica les dades d'un pagament.
    - EliminarPagamentAction: Elimina un pagament seleccionat.
    - AfegirCobramentAction: Afegeix un nou cobrament.
    - ModificarCobramentAction: Modifica les dades d'un cobrament.
    - EliminarCobramentAction: Elimina un cobrament seleccionat.
    - AfegirEntitatAction: Afegeix una nova entitat.
    - ModificarEntitatAction: Modifica les dades d'una entitat.
    - EliminarEntitatAction: Eliminar una entitat seleccionada.
    - AfegirPrestecAction: Afegeix un nou préstec.
    - CercarPagaments: Llista els pagaments en funció d'un filtre.
    - CercarCobraments: Llista els cobraments en funció d'un filtre.

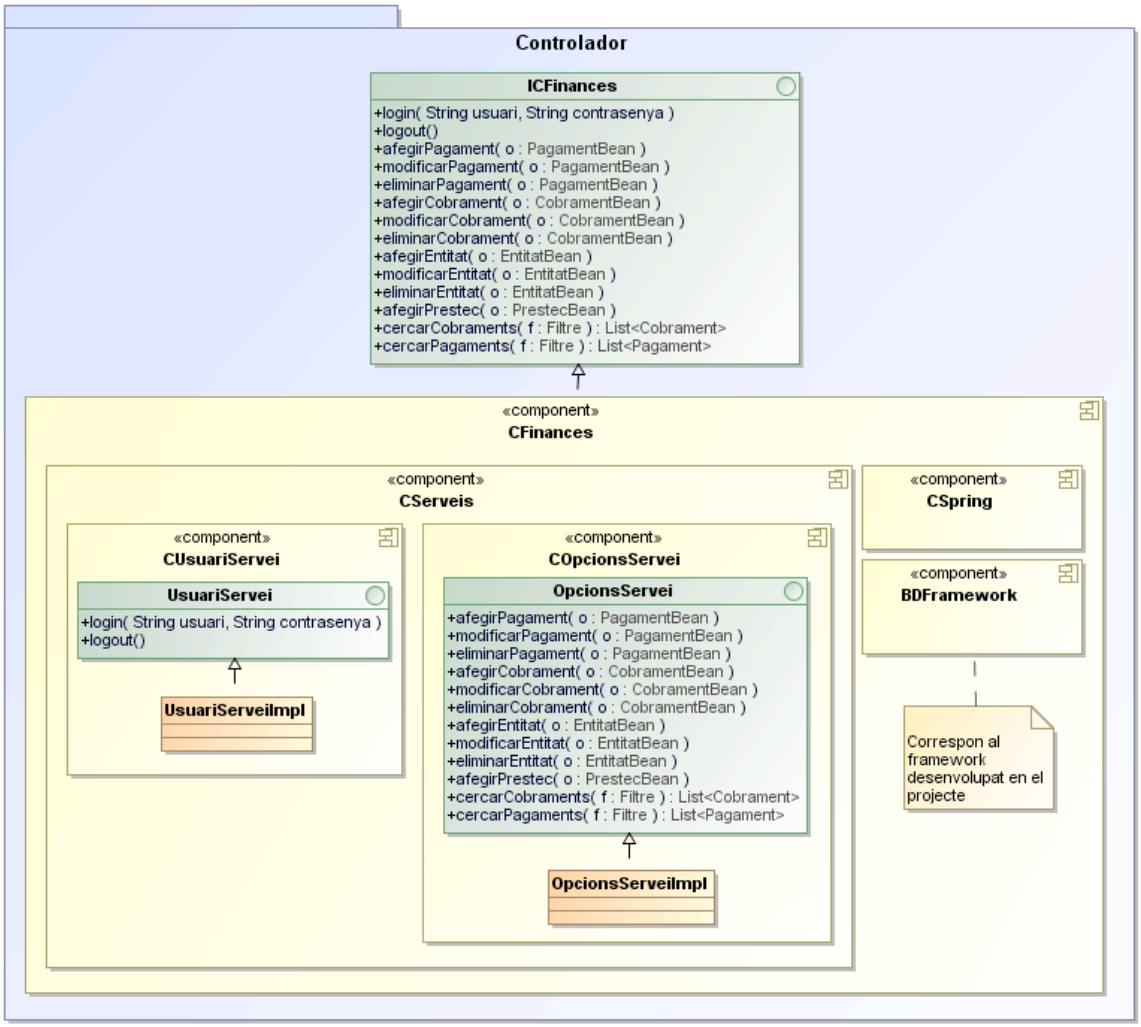
Per tan, el disseny de la capa de vista serà el següent:



7.4.2. Controlador

Per a la implementació d'aquesta capa s'utilitzarà el framework de persistència desenvolupat en aquest projecte. 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

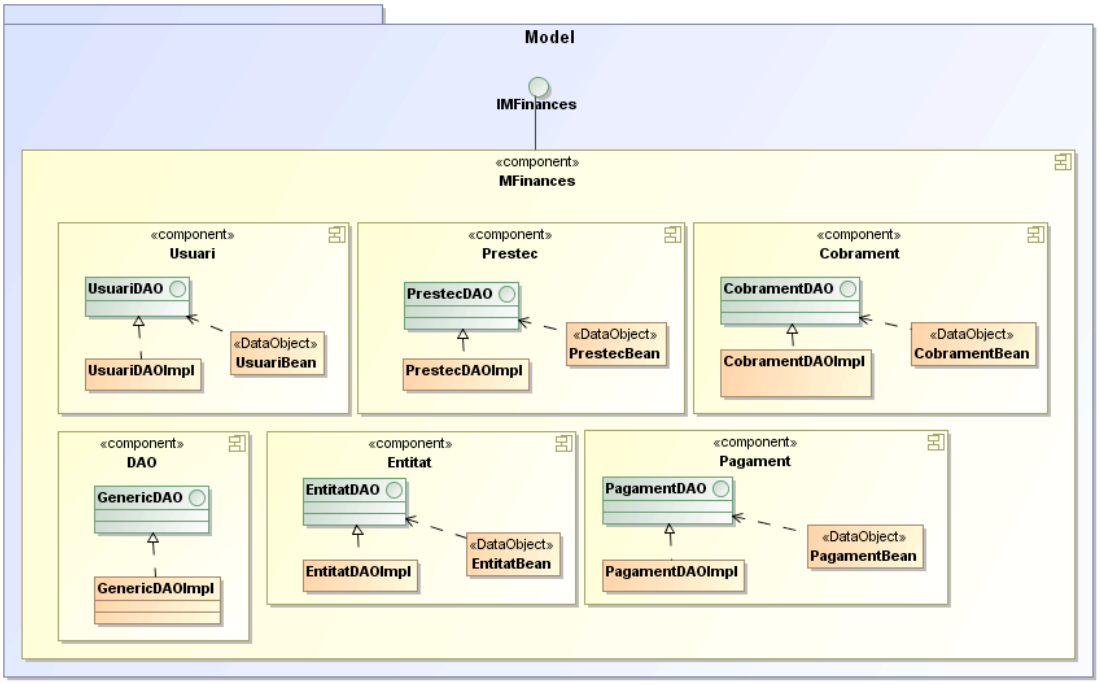
La capa del model estarà formada per tants components com taules disponibles tinguem en el model i pel component DAO. En aquest cas, s'ha optat per utilitzar un GenericDAO amb els mètodes comuns a tots els DAOs. Aquesta decisió aporta flexibilitat a la solució i evitar la repetició de codi – les operacions comunes 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 dels components propis del model estan formats per els següents objectes:

- <Taula>DAO: Interfície que disposa dels mètodes necessaris per a la persistència de les dades. Aquesta interfície estén de GenericDAO.
- <Taula>DAOImpl: Implementació dels mètodes de la interfície. Només implementa aquells mètodes específics de la taula donat que estén de GenericDAOImpl que incorpora els mètodes comuns a tots els DAOs.
- <Taula>Bean: Correspon als beans de base de dades. És un POJO clàssic amb el mapeig de la taula a codi Java. En cas d'utilitzar l'eina del mòdul d'utilitat (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 aprofundir en el core d'algunes de les solucions existents al mercat i ampliar el coneixement que disposava abans de la realització del projecte.

Per una banda, el desenvolupament m'ha permès reforçar els coneixements de les anotacions Java i fer ús més extens de l'API de reflection de Java i dels genèrics tan útils per evitar la replicació de codi. El fet de crear el generador de beans, m'ha permès aprofundir el l'API de JDBC per a poder generar dinàmicament les classes Java a partir dels metadates 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 partint de zero, fet que m'ha servit per 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 en 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 ens 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 de beans a partir de la base de l'esquema de base de dades, donat que actualment només genera les dades mínimes. Finalment es podria incloure algun mecanisme de cache per millorar el rendiment quan fos necessari.

Finalment, podríem concloure afirmant que gràcies a l'arquitectura de la solució proposada qualsevol modificació que es vulgui realitzar – tan en el framework com a la web – es podrien realitzar d'una manera senzilla.

9. Bibliografia

- CAMPDERRICH, BENET. Enginyeria del programari; Anàlisi orientada a objectes . UOC.
- BOOCH, G; RUMBAUGHT,J:JACOBSON,I. El lenguaje unificado de modelado. Ed. Adison Wesley
- Pressman, Roger S. "Software Engineering". McGrawHill, 1991.
- Álvarez Caules, Cecilio. "Arquitectura Java Sólida". 2012.
- Crawford, W.; Kaplan, J. "J2EE Desgin Patters". O'Reilly, 2003.
- http://www.ambysoft.com/downloads/persistenceLayer.pdf
- http://www.uow.edu.au/~nabg/399/JPA/JPA.pdf
- http://www.slideshare.net/ikercanarias/persistencia-de-datos-en-java
- http://carlos-henriquez.blogspot.com.es/2010/07/hibernate-framework-para-el-mapeo-de.html
- http://ca.wikipedia.org/wiki/Mapatge_d%27objectes_relacional
- http://www.agiledata.org/essays/mappingObjects.html
- http://www.programacion.com/articulo/conceptos_basicos_de_orm_object_relational_map ping_349
- http://dis.um.es/~jmolina/Persistencia%20de%20Objeto%20JDO.pdf
- http://ldc.usb.ve/~mgoncalves/IS3/Clase%201c%20IS-Persistencia.pdf
- 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.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2007/rapporter07/lindgren_tim_07069.pdf
- http://soa101.files.wordpress.com/2012/04/persistencia.png
- http://www.crazyteam.es/java/persistencia-i-hibernate-basico/
- https://github.com/zkoss/zkessentials/blob/master/src/main/java/org/zkoss/essentials/chapter8/AuthenticationInit.java
- http://books.zkoss.org/wiki/ZK%20Essentials/Chapter%208:%20Authentication

- http://www.programacion.com/articulo/persistencia_de_objetos_java_utilizando_hibernate_306
- http://forum.zkoss.org/question/19822/user-validation-and-session-management/
- http://stackoverflow.com/questions/15110937/how-to-create-a-session-and-role-based-access-using-zk-framework
- http://javaisjava.blogspot.com.es/2009/05/hacia-un-framework-de-persistencia.html
- http://www.genbetadev.com/java-j2ee/spring-framework-el-patrn-dao
- http://www.aprendiendojava.com.ar/index.php?topic=54.0
- https://today.java.net/pub/a/today/2007/12/18/adopting-java-persistence-framework.html
- http://sg.com.mx/revista/38/conceptos-diseño-patrones-tácticas-y-frameworks

Annex

El present annex és una guia realitzada per facilitar el procés de configuració de l'entorn de desenvolupament i la instal·lació del framework i l'aplicació desenvolupats en el present projecte. Principalment es descriurà els processos per tal d'executar i comprovar el funcionament tant de l'aplicació com del framework. Tot i això, també es descriurà els passos a seguir per tal de poder importar els dos projectes a Ecplise que és l'IDE que s'ha utilitzat en els dos casos per tal de poder veure com generar els fitxers corresponents per tal de realitzar a posterior la instal·lació (aquest pas no és necessari per tal de posar en funcionament el projecte, sinó que és un complement a poder veure tot el procés sencer per arribar al producte final).

10. Instal·lació de l'aplicació i framework

Prerequisits software

Els prerequisits per portar a terme la configuració i funcionament del framework i l'aplicació creada són els següents:

- Java JDK 1.7 o superior
- Apache Tomcat 7 o superior
- MySQL 5 o superior
- Script de BBDD executats sobre el motor (bd_tresor.sql)

Instal·lació software i configuració entorn

Primer de tot hem de procedir a instal·lar (en cas que no els tinguem ja instal·lats) les versions corresponents del Java JDK, Apache Tomcat i MySQL. Un cop instal·lats, hem de procedir a importar l'script sql al motor de base MySQL instal·lat. Finalment, hem de copiar dins la carpeta webapps de la carpeta on hem instal·lat l'Apache Tomcat el fitxer de distribució gartigas_webapp_tresor.war i engegar el servei del Tomcat, executant l'arxiu startup de la carpeta bin del Tomcat.

Un cop realitzat això, el mateix Tomcat ja s'encarrega de descomprimir automàticament tota la informació que tenim en el .war i ja podem obrir el navegador d'internet i anar a la direcció http://localhost:8080/gartigas_webapp_tresor/login.zul on ja tindrem el Login de la nostra aplicació per tal de poder ja treballar amb l'aplicació i el framework implementat.

Segurament s'hauran de canviar els paràmetres per tal d'accedir a la informació de la base de dades, ja que normalment la informació de connexió cap al motor de base de dades és diferent. Per tal de modificar aquesta informació, hem de modificar el fitxer uoc-bd-config.properties que trobem dins la carpeta gartigas_webapp_tresor\WEB-INF\classes\ de la

carpeta webapps del Tomcat. En aquest fitxer que tot seguit explicarem l'estructura, podem modificar els paràmetres de la connexió amb el motor de la base de dades i un cop modificats, només hem de reiniciar el servidor Tomcat i refrescar la pàgina web.

Per motius de tamany de les dades, s'ha creat una carpeta pública en el SkyDrive amb el directori Install Framework on a dins hi ha el fitxer war i l'script sql necessaris per poder utilitzar i comprovar el funcionament del framework i l'aplicació. En el següent enllaç es pot descarregar aquesta informació:

http://sdrv.ms/1bFADFX

10.1. Entorn de desenvolupament Framework de persistència

Tot seguit descriurem l'estructura del framework de persistència implementat i la importació a l'entorn de desenvolupament (que en el nostre cas s'ha realitzar utilitzant Eclipse). A la carpeta Framework de l'enllaç del SkyDrive tenim tot el contingut necessari per poder importar el projecte a Eclipse amb totes les llibreries corresponents.

Estructura de les fonts

L'estructura resumida de les fonts del projecte és la que es detalla:

gartigas_fw_bd

- bin: Directori on es compilen totes les classes i on hem de posar el fitxer .properties amb la informació de la base de dades
- config: Directori amb els recursos de configuració
- dist: Directori on es genera el JAR de la distribució
- lib: Directori amb les llibreries necessàries per l'execució del framework
- src: Directori amb el codi font del projecte

Compilació del framework

Un cop haguem importat el projecte a l'Eclipse i s'hagi enllaçat totes les referències amb les llibreries externes (prèviament haurem hagut d'instal·lar el motor de base de dades, importar el fitxer sql tal i com s'ha descrit en el pas anterior i modificat l'arxiu properties de la carpeta bin del projecte amb les dades corresponents a la connexió amb la base de dades) ja es pot procedir a generar el JAR de la distribució del framework per tal d'utilitzar-lo posteriorment per l'aplicació implementada per comprovar el funcionament del mateix. Per tal de generar el

JAR corresponent, hem de seleccionar el projecte gartigas_fw_bd i exportar-lo amb un JAR seleccionant les dependències corresponents (carpeta src i lib).

Per comprovar el correcte funcionament del framework, s'ha creat en el Package testing dues classes. La classe TestFwBD.java s'utilitza sols per comprovar la connexió amb la base de dades i les funcionalitats bàsiques d'insert, update i delete d'un objecte. La classe GeneraBDBeans.java que és la que utilitzarem en el framework posterior a l'aplicació per tal de generar automàticament tots els Beans amb les dades de les taules de la base de dades seleccionada.

10.2. Entorn de desenvolupament Aplicació

Tot seguit descriurem l'estructura de l'aplicació implementada i la importació a l'entorn de desenvolupament (que en el nostre cas s'ha realitzar utilitzant Eclipse). A la carpeta Aplicacio de l'enllaç del SkyDrive tenim tot el contingut necessari per poder importar el projecte a Eclipse amb totes les llibreries corresponents.

Prerequisits software

Per tal de poder compilar i generar el WAR corresponent a l'aplicació del framework, hem d'instal·lar el Plugin del ZK Studio a l'Eclipse. Aquest Plugin el podem descarregar directament de la pàgina http://www.zkoss.org/download/zkstudio i arrossegar l'enllaç del Marketplace cap a l'Eclipse. Es necessita registrar gratuïtament el plugin per tal de poder-lo utilitzar.

Estructura de les fonts

L'estructura resumida de les fonts del projecte és la que es detalla:

gartigas_webapp_tresor

- build: Directori on es compilen totes les classes i on hem de posar el fitxer .properties amb la informació de la base de dades
- config: Directori amb els recursos de configuració
- dist: Directori on es genera el JAR de la distribució
- lib: Directori amb les llibreries necessàries per l'execució del framework
- src: Directori amb el codi font del projecte
- WebContent: Directori amb els arxius .zul
    - images: Directori amb les imatges que utilitzem a la web

- WEB-INF\lib: Directori amb totes les llibreries externes utilitzades. En aquest directori a part de les llibreries que ja tenim automàtiques del ZK, hem d'afegir les llibreries externes del lib, juntament amb el JAR generat anteriorment del nostres framework

Compilació del framework

Un cop haguem importat el projecte a l'Eclipse podem enllaçar el projecte de l'aplicació amb el framework directament a l'Eclipse o bé importar el JAR generat anteriorment. Aquest pas només el necessitem per tal de generar els beans automàticament abans de generar el WAR que treballarem finalment.

Per tal de generar els Beans automàticament amb la informació de la base de dades, sols hem d'executar la classe GeneradorBeans.java que trobem en el Package edu.uoc.pfc.db.generador.beans la qual ens generarà automàticament tots els .java corresponents segons la informació de la base de dades.

Un cop generats els beans, sols ens queda exportar el projecte, seleccionant el projecte gartigas_webapp_tresor i exportant-lo amb un WAR. Llavors aquest fitxer WAR és el que hem de copiar al directori webapps del Tomcat (Tal i com hem explicat anteriorment) per tal de poder-lo desplegar en el servidor. Tot i això per tal de realitzar les proves i no haver de fer aquest pas cada cop, des del mateix Eclipse, podem executar el projecte seleccionant el fitxer Login.zul del WebContent i ell mateix ja engega el servidor Tomcat i ens carregar la pàgina web corresponent.

10.3. Recursos de configuració de l'aplicació

Tal i com hem comentat anteriorment, hem de configurar el fitxer .properties amb les dades corresponents a la configuració del nostre motor de base de dades. Aquest fitxer ha de contenir les següents propietats:

Nom	Descripció
bd.connection.driver	Classe de driver de connexió amb la base de dades
bd.connection.driver.id	Identificador del driver. S'utilitza per crear els beans
bd.connection.url	URL d'accés a la base de dades
bd.connection.user	Usuari de la base de dades
bd.connection.pass	Contrasenya de la base de dades

bd.connection.bdbeans.prefix.package	Configuració del proveïdor mitjançant el qual es realitzaran el mapeig corresponent del motor de base de dades
bd.connection.bdbeans.prefix.class	Prefix de la classe amb el mapeig
bd.connection.bdbeans.sufix.class	Sufix de la classe amb el mapeig
bd.cache.enabled	Flag per especificar si el cache esta activat
bd.beans.basedir	Directori on generem els Beans
bd.beans.package	Package Java en el qual crearem els Beans

Un exemple de configuració del fitxer descrit anteriorment seria el següent:

```
# 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/bd_tresor
bd.connection.user=root
bd.connection.pass=xxx

# Config provider
bd.connection.bdbeans.prefix.package=edu.uoc.pfc.db.nucli.provider
bd.connection.bdbeans.prefix.class=TaulaBDInfo
bd.connection.bdbeans.sufix.class=Impl


# Generacio de BD Beans
bd.beans.basedir=C:/Users/Guillem/workspace/gartigas_webapp_tresor/src
bd.beans.package=edu.uoc.pfc.webapp.beans
```

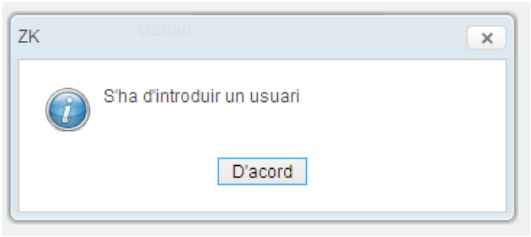
11. Aplicació Web – Manual d'Usuari

En aquest apartat es mostra el funcionament de l'aplicació web implementada juntament amb el framework implementat en aquest projecte.

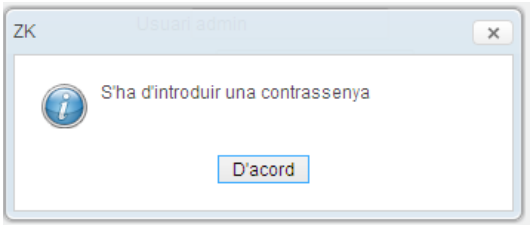
Pàgina Login

La pàgina d'inici és la pàgina d'identificació d'usuari. L'administrador de l'aplicació ha de proporcionar un usuari i una contrasenya amb el qual identificar-se per tal de poder entrar a l'aplicació. S'ha realitzat un control de sessions, el qual si intentem entrar a la resta de pàgines de l'aplicació i no hem realitzar el Login, o ens dóna un error d'identificació o ens redirigeix cap a la pàgina del Login per tal d'autenticar-se de nou. En cas de realitzar el Login correctament, ens redirigeix cap a la pàgina del menú de l'aplicació on tenim totes les opcions a realitzar.

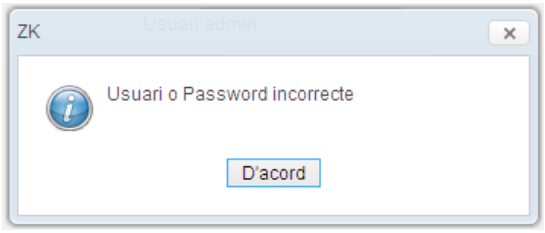
En cas de no posar Usuari ens sortirà l'error



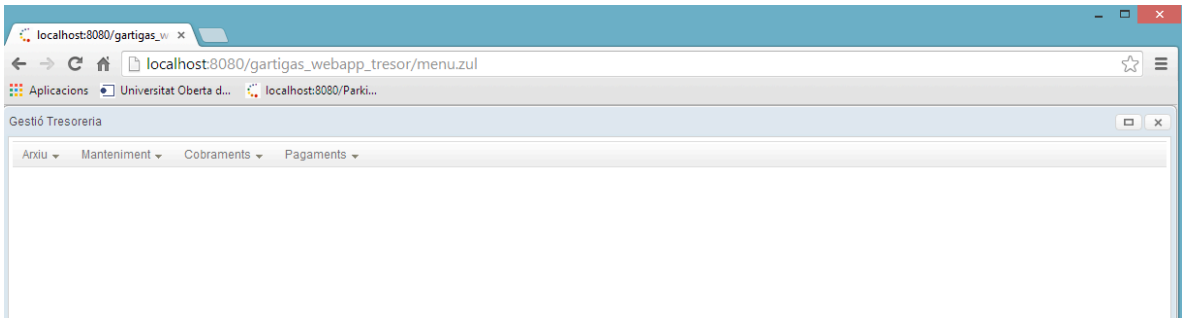
En cas de no posar Contrasenya ens sortirà l'error



Si les dades introduïdes no coincideixen amb cap usuari i contrasenya corresponent, ens apareixerà l'error corresponent

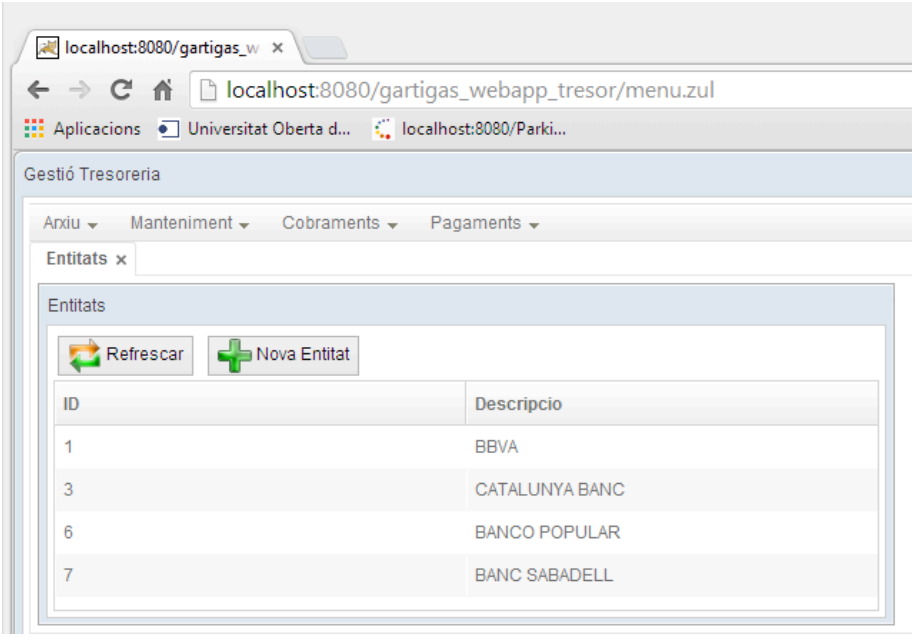


Si les dades introduïdes són correctes, ens apareixerà el menú de l'aplicació on tindrem totes les funcions disponibles



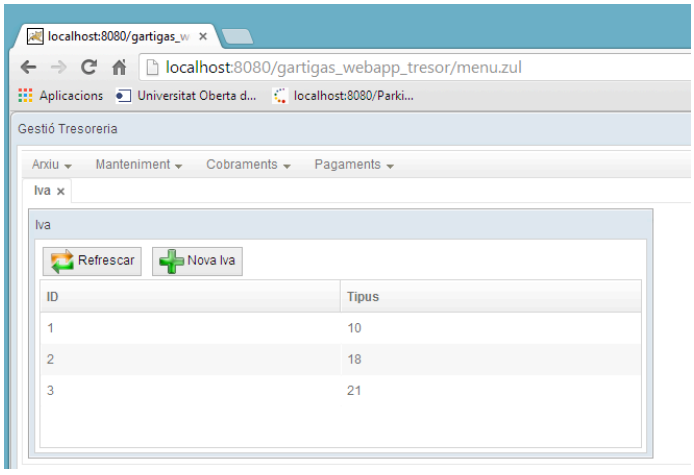
Pàgina Entitats

La pàgina d'entitats que tenim en el menú de manteniment és on podem afegir, modificar o esborrar les entitats que després haurem de seleccionar a l'hora d'introduir un cobrament o un pagament.



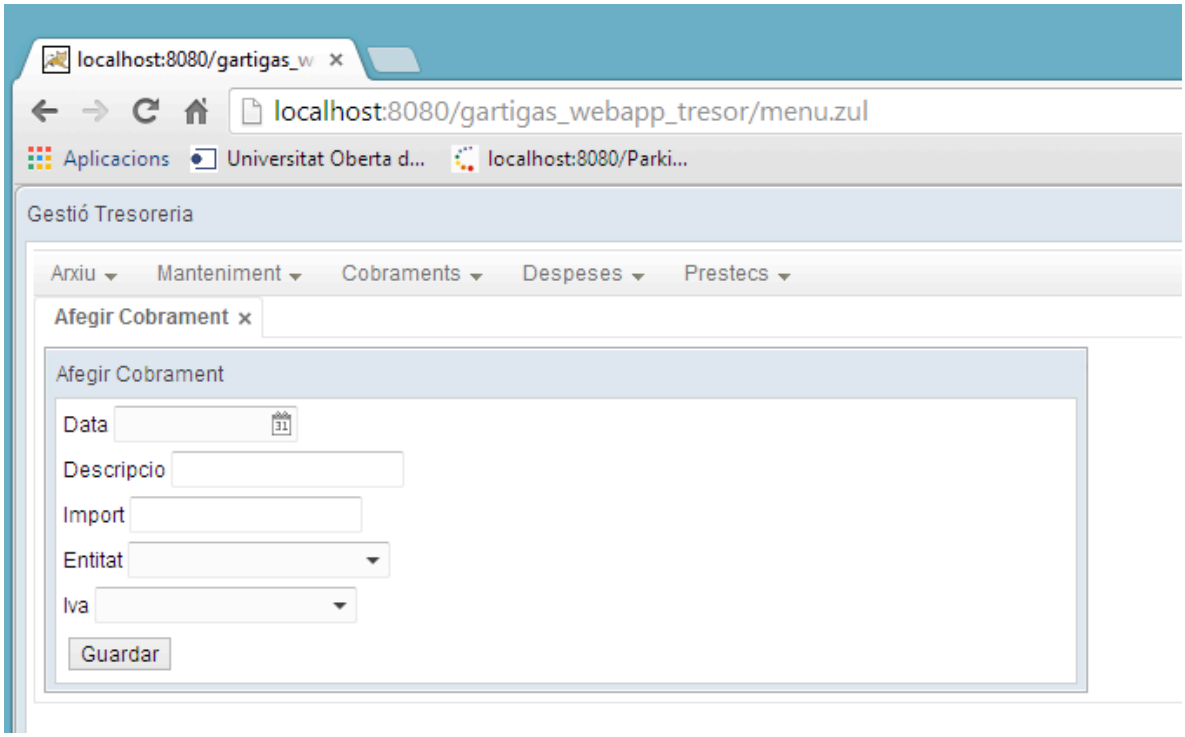
Pàgina Iva

La pàgina d'Iva que tenim en el menú de manteniment és on podem afegir, modificar o esborrar els tipus d'iva que després haurem de seleccionar a l'hora d'introduir un cobrament o un pagament.



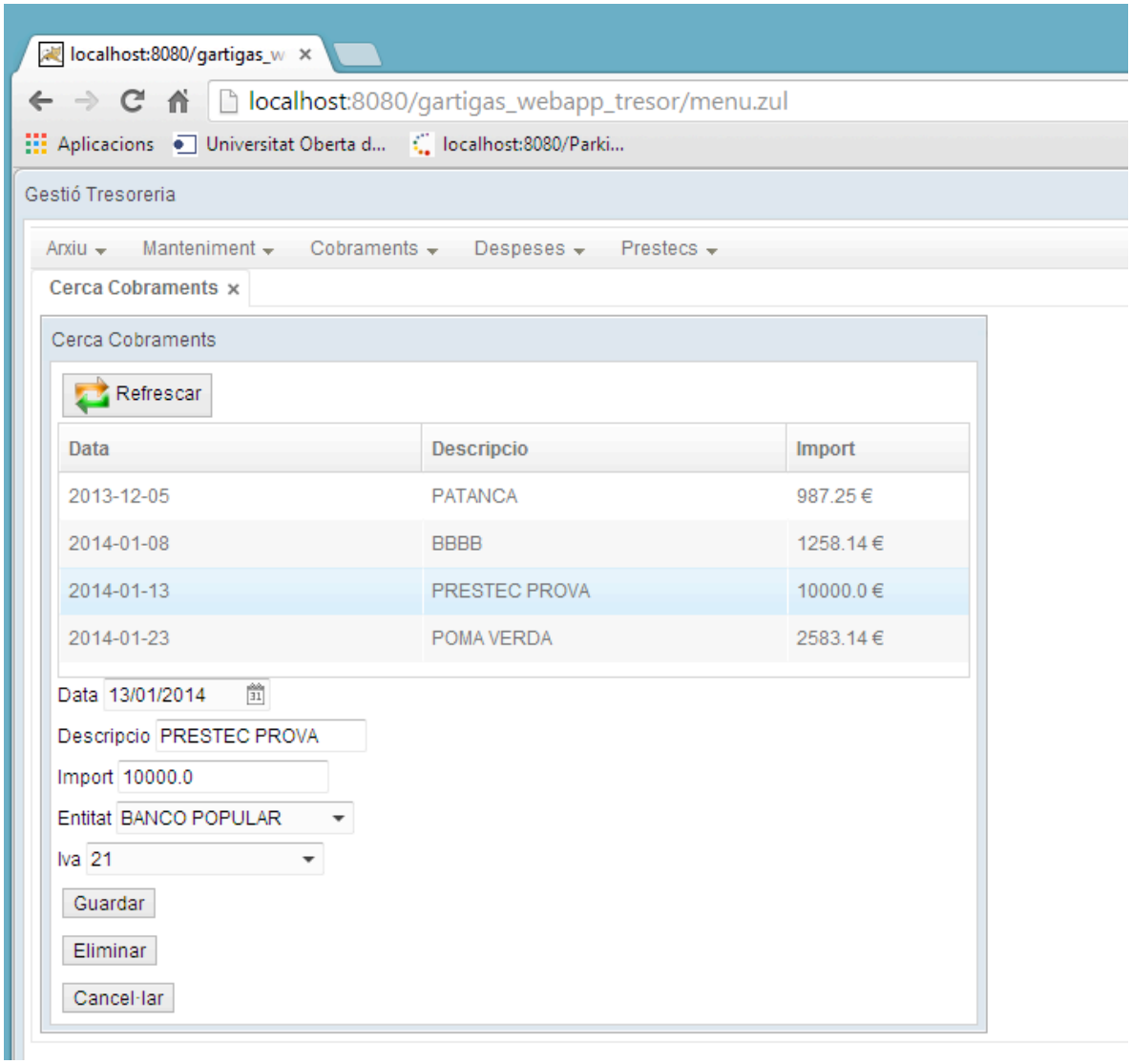
Pàgina Afegir Cobrament

La pàgina d'afegir cobrament que trobem dins del menú de Cobraments podrem introduir les dades per afegir els nous cobraments



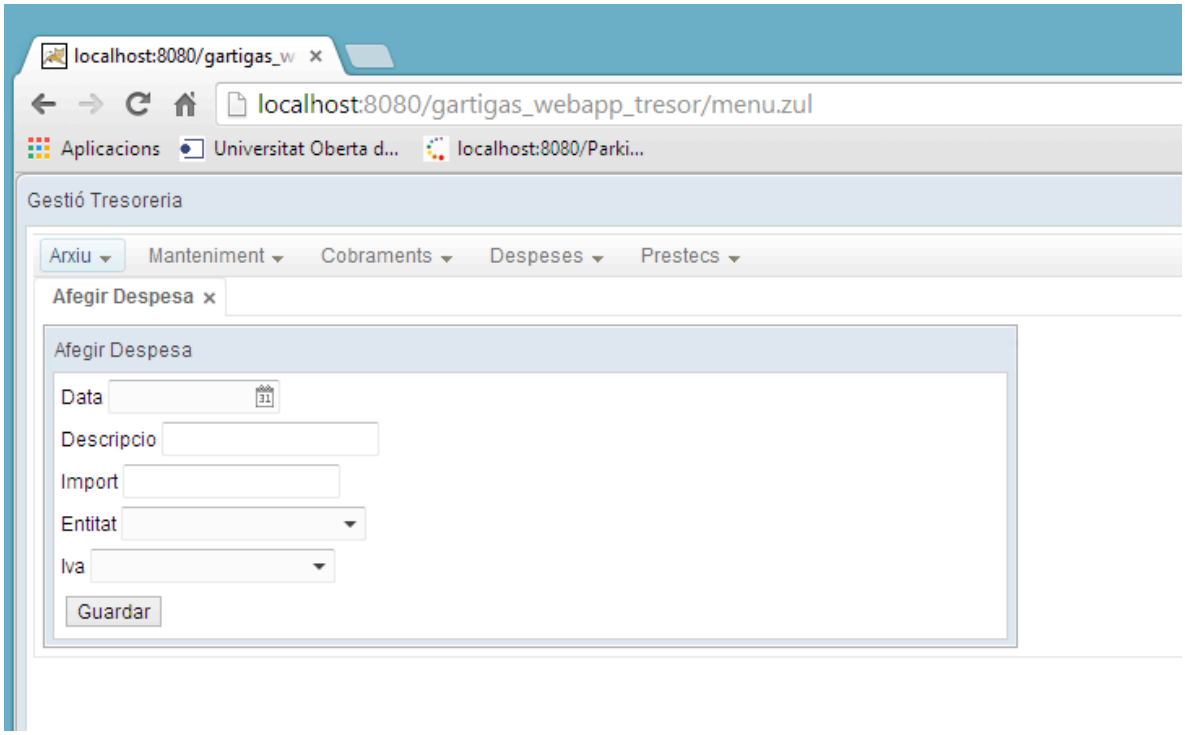
Pàgina Cerca Cobraments

La pàgina de cerca de cobraments ens surten tots els cobraments que tenim i si seleccionem un cobrament, ens mostra les dades per tal de poder modificar el cobrament seleccionat o esborrar-lo



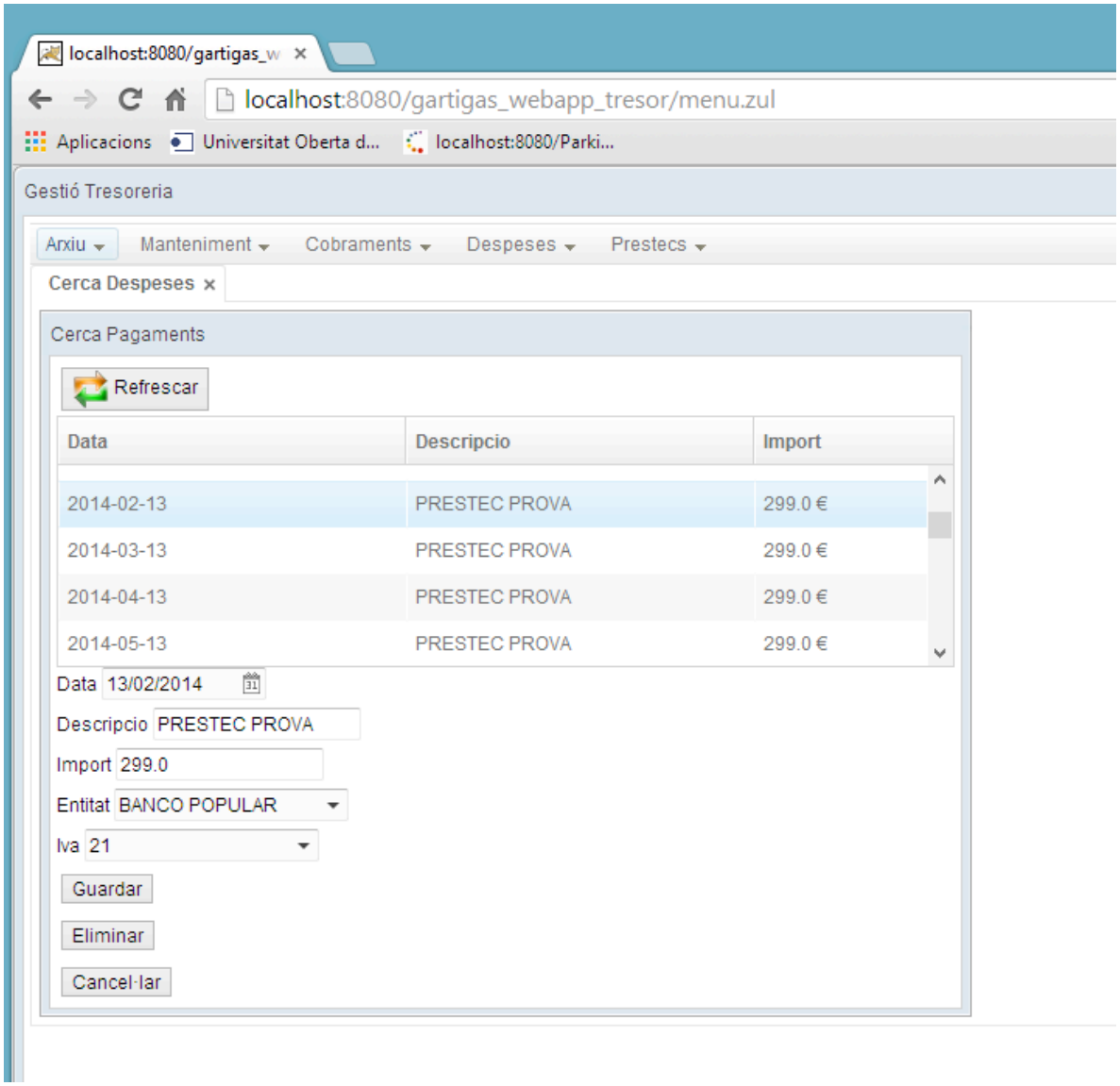
Pàgina Afegir Despesa

La pàgina d'afegir despesa que trobem dins del menú de Despeses podrem introduir les dades per afegir noves despeses



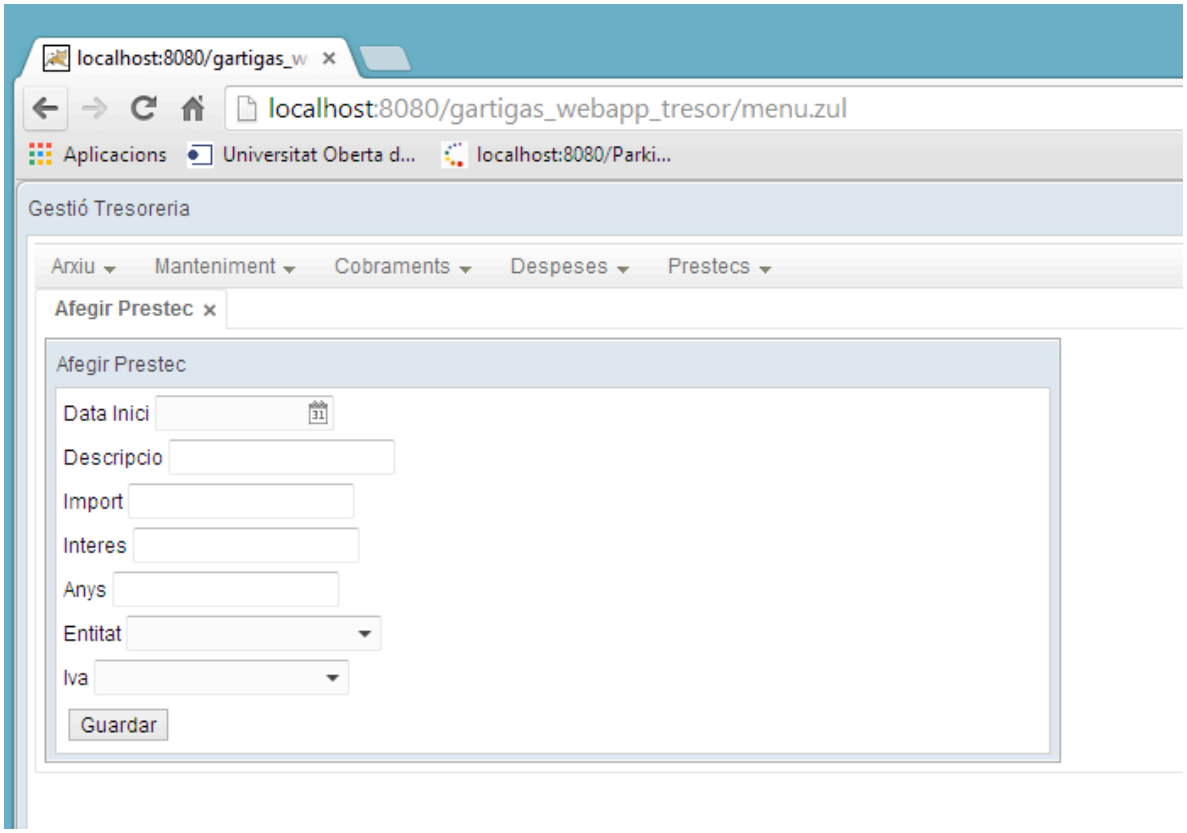
Pàgina Cerca Despeses

La pàgina de cerca de despeses ens surten totes les despeses que tenim i si seleccionem una despesa, ens mostra les dades per tal de poder modificar la despesa seleccionada o esborrar-la



Pàgina Afegir Préstec

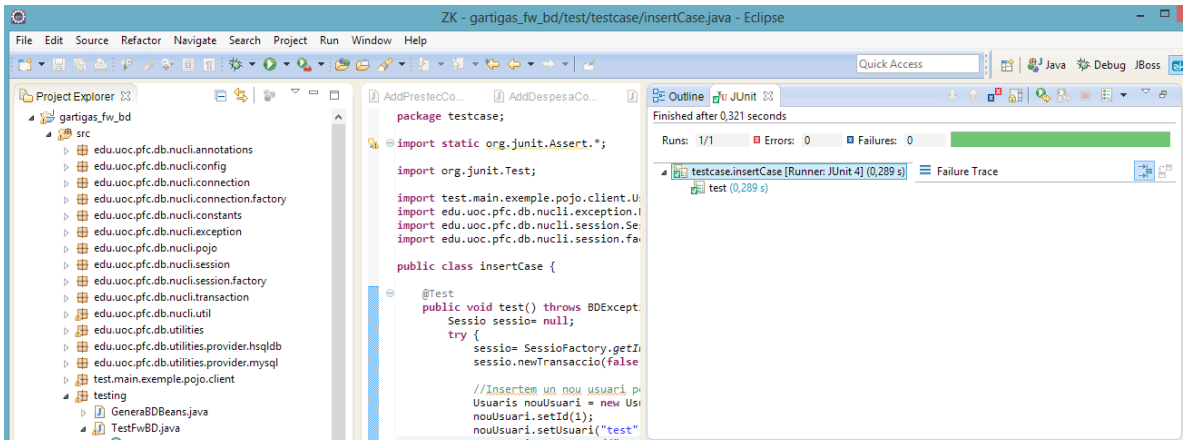
La pàgina d'afegir préstec que trobem dins el menú Préstecs, ens permet afegir un préstec anual indicant les dades corresponents, ens afegeix un cobrament amb el total de l'import del préstec a la data escollida, i ens calcula automàticament les quotes mensuals i les afegeix com a despesa cada mes a partir del mes següent del cobrament del mateix fins el nombre total de quotes pels anys indicats.



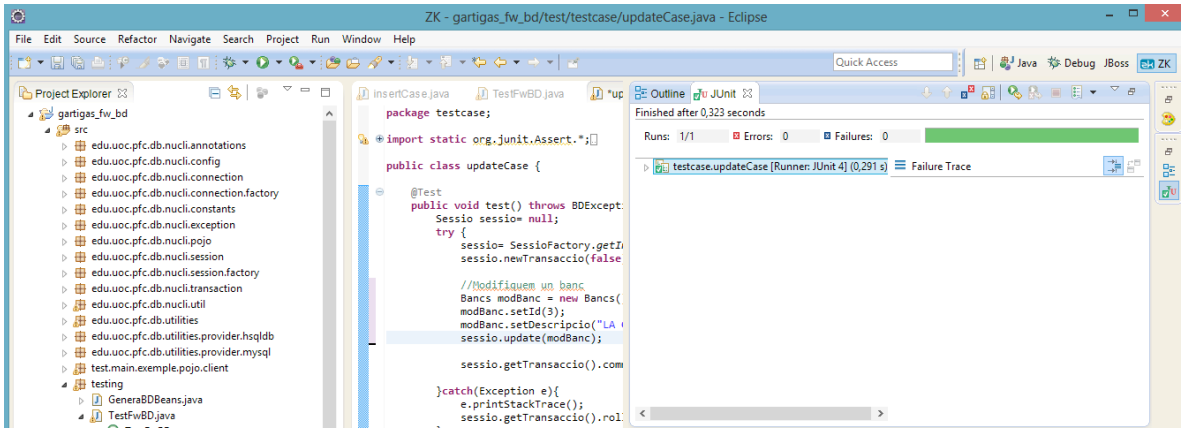
12. Test Cases Framework de Persistència

Utilitzant el JUnit amb l'Eclipse, s'ha realitzat els test cases de les funcionalitats bàsiques que utilitzem en el nostre framework de persistència. El test de les comunicacions cap al motor de base de dades es testeja en cada prova realitzada.

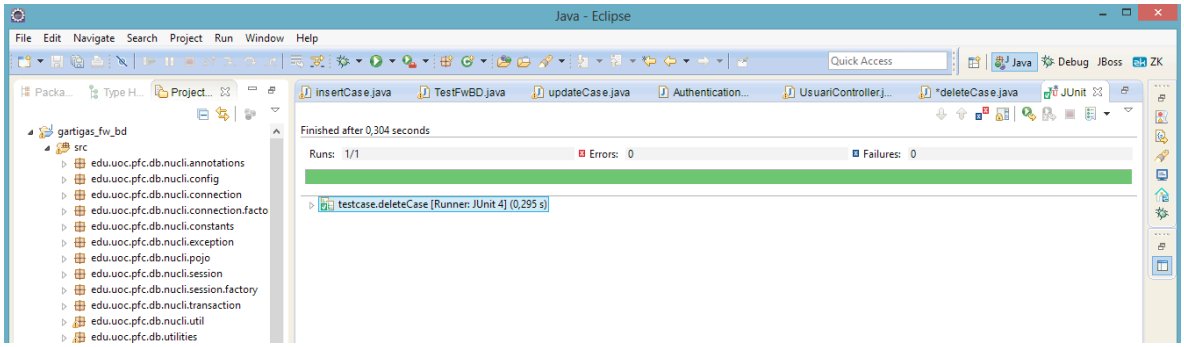
Com podem veure en la imatge següent, hem fet un test d'inserció de dades generant un Bean nou d'Usuari



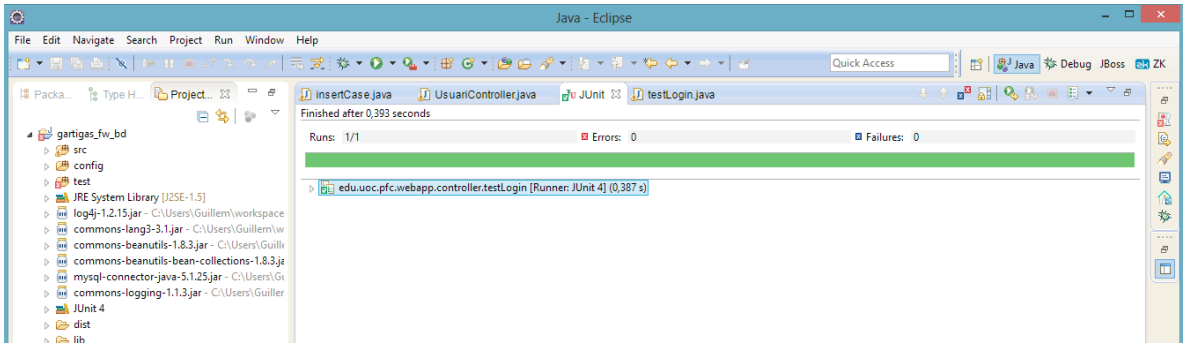
Com podem a la següent imatge, hem realitzat un test de modificació de dades d'una Entitat. Hem canviat el nom d'una entitat.



Com podem a la següent imatge, hem realitzat un test d'eliminació. En aquest cas hem procedit a eliminar una despesa.



Com podem a la següent imatge, hem realitzar un test del Login d'usuari per tal de comprovar el correcte funcionament de la cerca de dades a la base de dades.



Tal i com podem veure en aquests casos bàsics els quals hem realitzat els tests, en tots els casos hem obtingut el resultat esperat i no ens han fallat els valors introduïts. En principi tots els valors que ens escriuen en els camps via pàgina web, són controlats per codi en l'aplicació usuari i no ens el framework de persistència. Amb això ens assegurem que qualsevol interacció que tinguem cap al framework de persistència tinguem les dades correctes i esperades. Amb això evitem per exemple en cas de tenir l'aplicació a un servidor i la base de dades a un altre servidor, podem gestionar molt millor les transaccions i només utilitzar l'ample de banda en el cas que totes les dades que tenim siguin correctes.