Desenvolupament d'aplicacions per Internet amb tecnologia Java

Projecte **CreaTests**

Ramon Maria Gallart Escolà
ETIG

Vicenç Font Sagrista

17 de Gener de 2011

Aquesta pàgina s’ha deixat en blanc intencionadament

Dedicatòria i agraïments

Dedicat als meus pares,
ja em podré fer la foto amb el birret :-)

I a la meva parella, Marta,
en els moment complicats vas estar allí, t'm!!

I a la meva colla d'amics,
no els citaré per no deixar-me ningú
però gràcies per tots els cops que m'heu dit 'encara no has acabat!?!?'

Vull agrair a tota la comunitat UOC, consultors, tutors, personal de secretaria que sempre s'han dut molt bé amb mi i, finalment, al meu consultor d'aquest TFC, Vicenç Font Sagrista, el fet que aquest viatge que he fet per la UOC hagi estat tant profitós.

A la meva família, el suport que he tingut d'ells ens els bons i en els mals moments és impagable.

També vull agrair a la conjunció d'estrelles que han fet que aquest 2011 sigui l'any en què finalment acabo la carrera :-)

Moltes gràcies a tots,

Ramon Maria Gallart Escolà

Contingut

El present document conté les següents seccions generals:

- Un resum de la motivació per dur a terme aquest Treball de Final de Carrera incidint en el perquè del projecte i el perquè de la tecnologia escollida per dur-lo a terme.
- Tot desenvolupament d'un projecte comporta unes fases on el producte va prenent forma fins que arriva al seu estat final. Els capítuls centrals d'aquesta memòria tracten aquestes fases i com seguint-les s'aconsegueix el resultat final.
- Finalment, un Treball de Final de Carrera no té gaire sentit, o no deixa de ser una pràctica més, si no s'extreuen algunes conclusions de la motivació a dur-lo a terme i de tot el procés seguit en general. El darrer capítol d'aquesta memòria tracta precisament de les conclusions a les que he arribat sobre el propi projecte i la tecnologia emprada per aconseguir donar-li forma.

The present document contains the following sections:

- *An abstract about the motivation for the development of this Final Degree Work taking care about why this project has been chosen and why has it been developed with the technology to do it.*
- *Every project development goes through phases where the product takes form until it reaches its final state. The central chapters of this memory are about those phases and how following them we reach the final result.*
- *Finally, a Final Degree Work does not make sense, or is just another practice within it, if you can't work out some conclusions about the motivation of carrying it out and about all the process in doing so. The last chapter of this memory talks about the conclusions I have achieved about the project and the technology used to give it shape.*

Índex de continguts

Introducció 1
- **Objectius del TFC** 2
- **Enfocament i mètode seguit** 3
- **Planificació** 4
  - Il·lustració Planificació temporal 4
  - Il·lustració Diagrama de Gantt 4
- **Producte obtingut** 5
- **Contingut específic de la memòria** 6

L'aplicació CreaTests 7
- **Requeriments funcionals** 8
  - ***Gestor de tests*** *8*
  - ***Alumne*** *8*
  - ***Administrador*** *9*
- **Requeriments no funcionals** 9
- **Glossari de termes** 10
- **Definició d'actors i casos d'ús** 11
  - ***Actors*** *11*
  - ***Casos d'ús*** *11*
- **Casos d'ús dels actors** 12
  - Il·lustració Separació de casos d'ús en funció dels actors 12

Descripció dels casos d'ús 13
- **Alta de matèria** 13
  - Il·lustració Diagrama de col·laboració d'alta de matèria 13
- **Llistat de matèries** 14
  - Il·lustració Diagrama de col·laboració de llistat de matèries 14
- **Consulta/Modificació de matèria** 15
- **Baixa de matèria** 16
- **Gestió de tests** 17
- **Publicar un test real** 18
  - Il·lustració Diagrama de seqüència de publicar test real 18
- **Gestió de preguntes** 19
  - Il·lustració Diagrama de col·laboració de gestionar preguntes 19
- **Executar test de pràctiques** 20
- **Veure històric de qualificacions** 21
  - Il·lustració Diagrama de col·laboració d'històric de qualificacions 21
- **Gestionar usuaris** 22
  - Il·lustració Diagrama de col·laboració de gestionar usuaris 22
- **Entrar al sistema** 23
  - Il·lustració Diagrama de col·laboració d'entrada al sistema 23

Especificació dels casos d'ús 24
- **Entrada al sistema** 24
  - Il·lustració prototip entrada al sistema i pantalla principal 25
- **Sortida del sistema** 25
- **Llistat de tests** 25
  - Il·lustració prototip llistat de tests 27
- **Gestió de tests** 27
  - Il·lustració prototip creació / modificació test 29
- **Llistat de preguntes** 29
  - Il·lustració prototip llistat de preguntes 29
- **Gestió de pregunta** 30
  - Il·lustració prototip gestió de pregunta 30
- **Executar test de pràctiques** 30
  - Il·lustració prototip execució de test de pràctiques i resultat del test 32
- **Llistat de matèries matriculades** 32
  - Il·lustració prototip llista matèries matriculades 33
- **Veure històric de qualificacions** 33
  - Il·lustració prototip llista de qualificacions 33
- **Llistat d'usuaris** 34
  - Il·lustració prototip llistat d'usuaris 34
- **Gestió d'usuari** 35
  - Il·lustració prototip consulta / modificació d'usuari 35

Disseny tècnic del sistema 36
- **Entitats derivades de l'anàlisi** 36
  - Il·lustració diagrama estàtic d'anàlisi 36
- **Diagrama de classes del disseny** 37
  - Il·lustració Classes de disseny de test i de login 37

Il·lustració Classes de disseny de qualificacions 37
Il·lustració Classes de disseny d'usuaris 38
Disseny de l'arquitectura de l'aplicació 39
Arquitectura de la persistència amb Hibernate 39
Il·lustració Diagrama clàssic del patró DAO 39
Il·lustració Diagrama de classes típic per un DAO 40
Il·lustració Diagrama de classes amb un DAO genèric 40
Diagrama de l'arquitectura de persistència 41
Il·lustració Model físic de la base de dades 41
El patró de disseny MVC 42
Il·lustració Diagrama MVC 42
MVC i Java Server Faces 43
Integració de Hibernate 3 i JSF 2 amb Spring 3 44
Spring i Hibernate 44
Spring i JSF 2 46
La integració de tot plegat 48
Interfície d'usuari 49
Valoració econòmica 53
Conclusions 54
Bibliografia 55

Aquesta pàgina s'ha deixat en blanc intencionadament

Aquesta pàgina s'ha deixat en blanc intencionadament

Introducció

Duc treballant en el món de la informàtica, primer com a programador, després com a analista i actualment com a cap de projectes aproximadament uns 10 anys. De tot aquest temps, el llenguatge Java ha estat sempre present en la meva vida professional. Al llarg de vuit anys el vaig emprar a nivell professional creant alguna aplicació d'escriptori, però principalment aplicacions per a la web. Els darrers dos anys l'he emprat per acabar assignatures de la UOC i per anar fent projectes particulars.

Un petit resum de la meva vida laboral el podeu trobar a http://www.ramagaes.info/rmgs/blog/about/

Tot i això, tot i el fet d'haver estat emprant durant tant temps el Java per fer aplicacions web, sempre havia estat seguint les directrius d'algun bastiment desenvolupat a mida a l'empresa on estigués treballant. És per aquest motiu que, tot i fer molts anys que conec Java i la seva aplicació a la web, mai havia tingut l'ocasió de treballar-hi seguint les especificacions d'algun bastiment estàndard que em facilités el treball de poder crear una aplicació.

Aquesta ha estat la meva principal motivació a l'hora d'escollir com a Treball de Final de Carrera el desenvolupament d'una aplicació amb arquitectura J2EE; el poder afegir al meu bagatge el coneixement d'eines estàndard per poder realitzar aplicacions amb aquesta tecnologia. Els bastiments escollits, explicats en més profunditat al capítol de l'arquitectura, són JSF 2.0, Spring 3 i Hibernate 3. Aquests tres bastiments cobreixen les necessitats de desenvolupament d'una aplicació web.

L'objecte del TFC ha estat el desenvolupament d'una aplicació per poder executar tests i així poder avaluar els coneixements dels alumnes. El desenvolupament d'aquesta aplicació compleix amb els requisits de funcionalitat demanada en l'anàlisi funcional i implementa els casos d'ús definits al document realitzat durant la fase de disseny de l'aplicació.

El propi desenvolupament de l'aplicació ha obligat a realitzar alguns petits canvis respecte de les especificacions definides al document de disseny. Tot i això, aquests canvis no han estat mai per eliminar alguna funcionalitat definida, si no per afegir-ne alguna que facilités la seva implementació i fer l'aplicació més flexible i, a la fi, més adaptable a canvis futurs.

Finalment, dir que l'experiència ha estat molt positiva. Si bé no he incrementat el meu coneixement del llenguatge Java, més que en alguns punts molt concrets com la internacionalització, sí que ho he fet amb els tres bastiments que he usat per dur l'aplicació a bon terme. D'aquest coneixement espero treure'n un benefici tant personal com professional, en tant en quant ja les puc posar a la meva maleta de bagatge de coneixements.

Objectius del TFC

Són molt coneguts els diferents Blueprints associats a les diferents tecnologies Java.
Veure la web: http://www.oracle.com/technetwork/java/blueprints-141945.html

Els principals objectius d'aquest TFC han estat dos. El primer, aprendre més sobre les diferents tecnologies que el Java té per oferir-me per poder crear aplicacions web. És de sobres sabut que el Java és un llenguatge molt adequat per l'ús de les bones pràctiques a l'hora de programar, i estava encuriosit per aprendre més sobre aquestes eines i bones pràctiques.

El segon objectiu, la realització, ha estat la creació d'una aplicació que permetés realitzar l'avaluació de coneixements d'alumnes sobre matèries concretes. És per això que l'aplicació té una part de gestió, des d'on es poden gestionar els diferents usuaris que tenen accés a l'aplicació; una altra de gestió de tests, que serien els exàmens, les proves, que els professors posarien als alumnes per anar superant i practicant; i una darrera on l'alumne hauria de dur a terme la prova definida pel professor, és a dir, l'execució del test. També l'alumne tindrà la possibilitat de consultar quines han estat les seves notes al llarg dels tests realitzats.

Enfocament i mètode seguit

Tot projecte d'enginyeria requereix d'un procés per poder dur-lo a terme. Un projecte d'enginyeria informàtica no és menys en aquest aspecte. El programari no és una obra d'art, si no un producte de consum utilitari i massiu. Com a tal des de la seva concepció com a idea, fins la seva elaboració com a producte final, un projecte d'aquest tipus ha de passar per unes fases que l'han de conduir al seu estat final com a producte industrial.

Vegeu punt 1.2 del mòdul 1 de l'assignatura Enginyeria de programari

Les fases de construcció del programari **CreaTests** han estat les habituals per a aquest tipus de projecte, a saber: presa de requeriments, anàlisi, disseny i implementació, proves i desplegament.

S'ha seguit el mètode iteratiu i incremental per al seu desenvolupament. Emmarcat en aquesta tècnica, s'ha creat un prototip de com hauria de ser l'aplicació. En aquest prototip s'ha primat la rapidesa i el mostrar com haurien de ser les diferents funcionalitats que conformen l'aplicació. S'ha optat per aquesta tècnica ja que amb ella les probabilitats de descobrir mancances en les especificacions inicials o el seu disseny són més altes. Això redunda al final en una millor definició i implementació del producte. Com a conseqüència el resultat obtingut és més robust i més adequat a les necessitats reals del negoci al que va encarat.

Vegeu punt 2.2.2 del mòdul 1 de l'assignatura Enginyeria de programari

Tenint en compte que aquest hauria estat un producte *per vendre* i no pas un producte que ens hauria demanat un client, els seus requisits tant funcionals com tècnics han vingut definits per les necessitats del negoci. Això significa que els requeriments han estat els adequats gairebé des del principi del projecte, així com el seu anàlisi i el seu disseny. Tot i això, hi han hagut petites inconsistències que s'han anat solucionant al llarg de la implementació. Aquest comportament s'ha pogut donar gràcies a que l'anàlisi i el disseny s'ha anat fent sobre les diferents funcionalitats de l'aplicació i no sobre l'aplicació en general (com hagués sigut si s'hagués seguit el model clàssic de cicle de vida en cascada o clàssic).

Vegeu punt 2.1 del mòdul 1 de l'assignatura Enginyeria de programari

Planificació

La planificació ha tingut canvis al llarg del TFC. Al principi la feina estava concentrada en els caps de setmana, per motius laborals. Al mes de començar el TFC vaig tenir un problema al peu que, de moment, fa que ja dugui gairebé dos mesos de baixa laboral. Això significa que ja no està concentrada tota la feina els caps de setmana però, per contra, els dies que fa molt mal tampoc puc fer res.

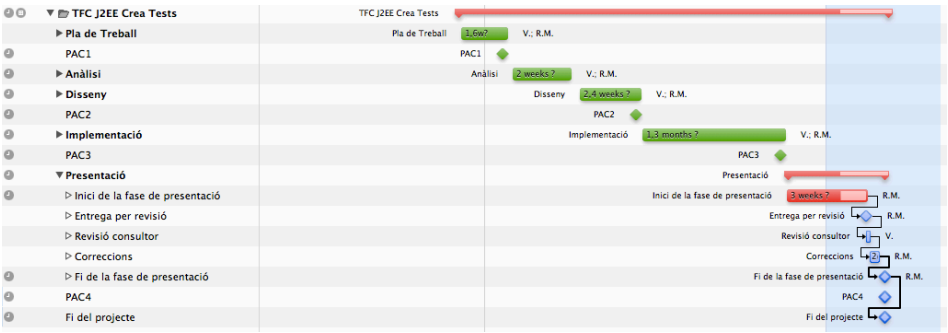
Aquesta és la distribució en el temps de les diferents fases en les que el projecte s'ha dut a terme:

L'eina emprada per fer el seguiment i el diagrama de Gantt s'anomena Merlin. Està creada per funcionar amb sistemes Mac. Es pot trobar més informació de l'eina a http://www.projectwizards.net/en/

0	TFC J2EE Crea Tests		24/09/10	17/01/11	Vicenç; Ramo...	88%
1	Pla de Treball		24/09/10	06/10/10	Vicenç; Ramo...	100%
11	PAC1		06/10/10	06/10/10		100%
12	Anàlisi		08/10/10	24/10/10	Vicenç; Ramo...	100%
23	Disseny		26/10/10	11/11/10	Vicenç; Ramo...	100%
34	PAC2		11/11/10	11/11/10		100%
35	Implementació		12/11/10	20/12/10	Vicenç; Ramo...	100%
46	PAC3		20/12/10	20/12/10		100%
47	Presentació		21/12/10	17/01/11	Vicenç; Ramo...	17%
48	Inici de la fase de presentació	44	21/12/10	11/01/11	Ramon Maria	50%
50	Entrega per revisió	48	11/01/11	11/01/11	Ramon Maria	0%
52	Revisió consultor	50	12/01/11	12/01/11	Vicenç	0%
54	Correccions	52	12/01/11	15/01/11	Ramon Maria	0%
56	Fi de la fase de presentació	54	17/01/11	17/01/11	Ramon Maria	0%
58	PAC4		17/01/11	17/01/11		0%
59	Fi del projecte	56	17/01/11	17/01/11		0%

Il·lustració Planificació temporal

En forma de diagrama de Gantt, les tasques queden distribuïdes de la manera següent:



Il·lustració Diagrama de Gantt

Les funcionalitats indicades per al seu desenvolupament en el document de disseny s'han acomplert al 100%. Aproximadament corresponen al 80% de la càrrega total de funcionalitat que hauria de tenir l'aplicació tal i com es va concebre en la seva fase de requeriments.

Producte obtingut

El producte obtingut s'anomena **CreaTests**. Aquesta eina està orientada al món de l'ensenyament. Té diferents objectius. Per una banda, els professors tenen una eina on poden crear contingut en forma de tests que després els alumnes hauran d'executar. Alhora, s'estalvien de corregir els exàmens, ja que indiquen quines són les respostes correctes a les preguntes al moment de crear-les i la mateixa aplicació dóna la nota a l'alumne quan aquest finalitza el test. D'altra banda, els alumnes tenen una plataforma per posar a prova els seus coneixements amb els tests de pràctiques. Poden seguir la seva evolució en blocs determinats de les matèries a les que estiguin matriculats des de l'opció d'històric de qualificacions. Finalment, l'execució dels exàmens es fa des de la mateixa plataforma en un dia i hora determinats. Quan acaben els tests, ja siguin de pràctiques o no, poden veure quines preguntes han fallat i quines eren les seves respostes correctes.

El desenvolupament del producte s'ha dut a terme amb tecnologia J2EE. Aquesta és l'especificació de Java per aplicacions del costat de servidor. Defineix i proporciona eines per poder desenvolupar aplicacions en diferents capes, essent potser les més comunes, les de client-servidor.

També s'ha usat JavaServer Faces, JSF. Aquesta és una tecnologia que defineix components de la capa de vista a la part del servidor per veure's en el client (usualment un navegador web). A més, també empra el patró MVC, cosa que simplifica molt el procés de navegació dintre de l'aplicació. És el bastiment oficial d'Oracle per desenvolupar aplicacions sota J2EE. Tot i això no n'és l'únic. Una de les grans avantatges que té el Java per sobre d'altres plataformes és que la varietat per poder escollir bastiments amb els que crear aplicacions és molt gran. Organitzacions com Apache realitzen aportacions molt interessants al món Java, essent en aquest cas el bastiment Struts una de les més notables.

Vegeu http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

Vegeu http://struts.apache.org/2.x/index.html

Per a la capa de persistència s'ha fet ús de Hibernate 3. Amb aquest bastiment de persistència les operacions CRUD (les operacions CRUD són les bàsiques que es realitzen sobre una entitat, crear -**C**reate-, obtenir -**R**etrieve-, actualitzar -**U**pdate- i eliminar -**D**elete-) sobre les entitats del negoci se simplifiquen de manera important. A més també permet fer consultes elaborades amb un llenguatge propi que és independent d'implementacions d'SQL per diferents venedors, cosa que redunda en una major portabilitat de l'aplicació si s'ha de canviar el sistema gestor de base de dades sobre el que en un inici s'havia pensat.

Finalment, com a capa d'integració, s'ha usat Spring 3. Amb aquesta capa hem lligat Hibernate amb JSF. Un gran avantatge que presenta Spring és a l'hora de gestionar la transaccionalitat de les operacions de base de dades. Amb aquest bastiment d'integració és molt senzill controlar aquest aspecte, ja que ho fa ell de forma automàtica, cosa que redunda en menys errors de codi associats a aquesta circumstància.

Contingut específic de la memòria

El contingut de la memòria és el que segueix a continuació:

- Requeriments de l'aplicació
- Anàlisi del sistema
- Descripció dels casos d'ús implementats
- Disseny dels casos d'ús
- Arquitectura de la persistència
- Arquitectura de l'aplicació
- Descripció de la interfície de l'usuari
- Conclusions

L'aplicació CreaTests

L'aplicació **CreaTests** s'emmarca dintre de la necessitat d'aportar valor i confiança al nou sistema educatiu. Fins ara, la forma tradicional de saber per part d'un professor si els seus alumnes progressaven de forma adient era fent exàmens que després s'havien de corregir a mà. Amb aquest sistema l'alumne serà capaç de saber en tot moment quin és el seu grau de maduració dels coneixements que va adquirint. Alhora, el professor serà capaç de detectar de forma senzilla quines són les matèries que més costen als alumnes al mateix temps que podrà extreure estadístiques que ho corroborin.

CreaTests ha de permetre la creació i manteniment dels tests per que després puguin ser executats. L'aplicació podrà ser usada tant en entorns d'eLearning, per poder comprovar si els alumnes han assolit el nivell necessari en alguna assignatura, com per particulars que estiguin estudiant i vulguin comprovar el seu nivell d'assoliment de comprensió d'allò que estudien, etc... Per poder aconseguir aquestes fites, la construcció del programari ha de ser robusta, a la vegada que flexible per poder incloure noves especificacions de forma poc traumàtica i així poder ampliar la seva funcionalitat fàcilment.

Pel que fa a la gestió dels tests, s'han de poder donar d'alta de forma senzilla, tant el que són les preguntes i les seves possibles respostes, com el que serien les regles per puntuar-lo. Per normes podem entendre si descompten les preguntes no contestades, si no es poden fallar més de x preguntes en total, etc... De tests n'hi haurà de dos tipus: de pràctiques i reals. Els tests de pràctiques serviran per que l'alumne pugui comprovar el seu nivell de coneixements abans d'enfrontar-se a un test real. Aquests tests de pràctiques no tenen un temps límit per la seva realització i a més poden comptar amb ajudes per cada pregunta del test en forma de pistes que l'alumne podrà visualitzar abans de contestar la pregunta.

Al finalitzar un test, l'alumne ha de poder veure quina és la nota que ha tret en funció de les regles que tingui el test. A més, per a les preguntes fallades i les no contestades ha de poder visualitzar quina era la resposta correcta i, en cas d'haver-la fallat, poder veure quina era la resposta que havia seleccionat.

Requeriments funcionals

Tenint en compte el tipus d'usuari que accedeix a l'aplicació el funcionament de la mateixa ofereix unes opcions o unes altres. Els fluxos generals són els següents:

Al llarg del document ens referirem a la gestió de X (on X pot ser qualsevol cosa) com a les típiques operacions de donar d'alta, consultar, modificar i eliminar. Ho fem així per evitar ser massa extensius amb detalls que no aporten gaire valor al document.

Gestor de tests

L'usuari gestor de tests quan entra a l'aplicació té les opcions següents disponibles:

- **Gestió de matèries**. Els alumnes estaran associats a assignatures de les que podran realitzar tests. Des d'aquesta opció es pot gestionar el referent a cada matèria.
- **Gestió de blocs**. Les assignatures es composen de blocs. En aquest apartat és on es definiran aquests blocs. Per exemple, l'assignatura de matemàtiques es podria dividir en blocs com àlgebra, càlcul, geometria...
- **Gestió de temes**. Els blocs d'una matèria es poden descomposar en temes. Per exemple, àlgebra podria tenir els temes d'àlgebra lineal, àlgebra booleana...
- **Gestió de tests**. Amb aquesta funcionalitat, el gestor de tests pot donar d'alta els tests i modificar-los o mantenir-los. Quan donem d'alta un test definim totes les seves característiques, des de l'assignatura a la que pertany, fins com s'ha de puntuar.

Alumne

L'alumne tindrà disponibles les funcionalitats següents:

- **Executar test de pràctiques**. Amb aquesta opció, l'alumne podrà executar un test de qualsevol de les assignatures que tingui associades. Per cada assignatura tindrà disponibles els tests dels diferents blocs que s'hagin definit. L'alumne pot executar aquests tests un nombre indefinit de vegades. Els resultats d'aquests tests es guardaran en un registre històric que posteriorment podrà ser consultat.
- **Executar test real**. L'execució real d'un test és una opció que no sempre estarà disponible. La disponibilitat dependrà del gestor de tests que haurà de programar quan es pot realitzar el test. Aquest tipus de test tindrà un temps límit d'execució, passat el qual s'emmagatzemarà automàticament i quedarà tancat sense possibilitat de modificació per part de l'alumne. Finalitzat el test, l'alumne sabrà de forma immediata quina és la nota que ha tret del test. Al igual que amb els tests de pràctiques, podrà revisar les preguntes no contestades i les fallades podent comprovar quines eren les respostes correctes i les contestades per ell mateix.
- **Històric de qualificacions**. Des d'aquest punt l'alumne podrà revisar la trajectòria de totes les assignatures que té associades. Podrà veure les qualificacions i els temps emprats per acabar tant els tests de pràctiques com els reals.

Administrador

L'administrador té les següents funcionalitats disponibles:

- **Gestió dels usuaris**. Amb aquesta opció l'administrador podrà donar d'alta un usuari del sistema de qualsevol dels tipus que el sistema admet. En cas de ser un alumne, podrà associar-li les matèries de les que estigui matriculat.
- **Dades estadístiques**. Des d'aquesta opció, l'administrador podrà extreure dades estadístiques d'un alumne o globalment de tots els alumnes que estan matriculats d'una determinada matèria.

Requeriments no funcionals

Tota aplicació a desenvolupar conté elements que no formen pròpiament part de la funcionalitat del sistema, si no que més aviat són requeriments fins a cert punt abstractes, però que tot i això, s'han de mirar d'incorporar en l'anàlisi i, és clar, s'han de veure reflexats en el producte final. Acostumen a especificar propietats del sistema, tals com restriccions de l'entorn o de la implementació, rendiment, dependències de la plataforma, facilitat de manteniment, extensibilitat, fiabilitat —en general, totes les "-ats".

Part de la definició extreta de "El Proceso Unificado de Desarrollo de Software", de Jacobson, Booch i Rumbaugh

Els requeriments que no formen part directa de la funcionalitat de l'aplicació **CreaTests**, però que són importants per a un bon desenvolupament de la tasca que ha de dur a terme, són els següents:

- **rapidesa**: l'aplicació ha de ser fluïda en el seu funcionament. Aquest punt és important ja que el client on es desenvoluparà l'activitat de l'aplicació serà un client web.
- **fàcil d'utilitzar i orientada a usuari**: s'ha de tenir una cura específica a l'hora de crear la interfície d'usuari per que sigui fàcilment utilitzable.
- **fiable**: aquest punt també és força important. Per fiable entenem el fet que la informació que doni l'aplicació sigui la correcta. És a dir, que no demanem la informació d'un usuari i ens doni la d'un altre.
- **segura**: no hauria de ser possible per cap usuari poder entrar a l'aplicació com si en fos un altre si no sap les seves dades d'accés. Això implica l'ús d'una entrada segura via login i password, validació d'operacions dintre de la web, evitar atacs d'injecció SQL, etc...
- **extensible**: l'aplicació ha de tenir la possibilitat de ser ampliable de forma senzilla a noves funcionalitats que puguin arribar.

En un producte 'real' es realitzarien proves d'usabilitat en forma d'observació directa de la interacció dels usuaris amb l'aplicatiu així com passant diferents enquestes on els propis usuaris avaluarien el seu disseny i usabilitat. Així mateix també se seguirien les pautes marcades per la Web Accessibility Iniciative per crear un lloc web el màxim d'accessible http://www.w3.org/WAI.

Glossari de termes

És important definir un glossari de termes que estigui contextualitzat dintre de l'àmbit de l'aplicació que s'està creant.
Amb aquest glossari, la comunicació entre el client i l'equip de desenvolupament i, fins i tot, entre els propis membres de l'equip, és més fluïda i duu a menys equívocs.

Descrivim a continuació alguns dels termes emprats en l'aplicació.

Terme	Descripció
Administrador	Usuari amb atribucions per poder gestionar els usuaris de l'aplicació i veure dades estadístiques globals o d'alumnes concrets.
Alumne	Usuari que pot executar un test, ja sigui real o de pràctiques per comprovar la seva evolució amb les matèries de les que està matriculat.
Assignatura	Veure matèria.
Bloc	Subdivisió dels continguts d'una matèria. Per exemple, la matèria 'Anàlisi Matemàtica' es pot dividir en els blocs 'Funcions', 'Derivació', 'Integració', etc...
Gestor	Usuari amb atribucions per poder crear tests, matèries, blocs i temes. En el context de l'aplicació, es pot considerar sinònim de professor.
Matèria	Conjunt de coneixements aplicats a un mateix àmbit. Els alumnes es matricules de matèries. En el context de l'aplicació es pot considerar sinònim d'assignatura.
Pregunta	Element del test consistent en un enunciat que planteja un problema i que l'alumne ha de resoldre seleccionant la resposta correcta.
Professor	Veure gestor
Resposta	Per a cada pregunta del test existeixen diferents opcions per contestar on només una de les quals és la correcta. Cada opció és una resposta.
Tema	Subdivisions dels blocs que formen les matèries. Per exemple, del bloc 'Derivació' de la matèria 'Anàlisi Matemàtica', podem tenir els temes 'Concepte de límit', 'Resolució de derivades', 'La regla de l'Hôpital', etc...
Test	Conjunt de preguntes amb diferents respostes disponibles de les que només una és correcta. Al finalitzar el test, a l'alumne se li dóna la nota resultant de la seva execució i se li permet veure quines eren les respostes correctes d'aquelles que ha fallat o no ha contestat.
Test (de pràctiques)	Test el resultat del qual no compta per a la nota final de la matèria. Durant l'execució del test, l'alumne té la possibilitat de rebre ajuda sobre la pregunta que estigui contestant.
Test (real)	Test el resultat del qual compta per a la nota final de la matèria.

Definició d'actors i casos d'ús

Actors

Hi ha tres actors que interactuen amb el sistema:

- **Administrador**: L'administrador és l'usuari que durà un control dels usuaris que poden accedir al sistema i de les estadístiques.
- **Alumne**: L'alumne és el tipus d'usuari que contestarà les preguntes del test. Podrà seguir la seva evolució amb les diferents matèries gràcies a la funcionalitat d'històric de qualificacions.
- **Gestor de tests (professor)**: El gestor de tests és l'usuari que s'encarrega de la creació i manteniment dels tests i de tot allò que hi està relacionat.

Els actors s'identifiquen amb una d'aquestes tipologies en funció de la seva creació a partir d'una categorització que els identifica com a pertanyents a un o diversos d'aquests grups.

Els diferents actors tindran accés a diferents funcionalitats dintre del sistema. Això sí, dintre del seu rol d'actors tots tindran accés a tots els elements que els hi siguin permesos (no hi ha propietat dels elements creats ni jerarquia entre els actors que són d'un mateix tipus).

Un usuari del sistema pot ser de més d'un tipus d'actor a la vegada podent fer totes les operacions dintre del sistema que s'adeqüin al tipus d'actor que és.

Casos d'ús

Aquests són els casos d'ús del sistema

1. Gestió de matèries
   1.1. Alta de matèria
   1.2. Llistat de matèries
   1.3. Consulta/Modificació de matèria
   1.4. Baixa de matèries
2. Gestió de blocs
3. Gestió de temes
4. **Gestió de tests**
   4.1. Publicar un test real
5. **Gestió de preguntes**
6. **Executar test de pràctiques**
7. Executar test real
8. **Veure històric de qualificacions**
9. **Gestionar usuaris**
10. Consultar dades estadístiques
11. Generar informe d'aprovats/ suspesos per assignatura
12. **Entrar al sistema**

Els casos d'ús que estan en **negreta** són els que s'han implementat en el TFC. La càrrega de la implementació dels casos d'ús assenyalats d'aquesta manera representa aproximadament el 80% de la càrrega total que seria implementar-los tots.

Així mateix, els casos d'ús que són del tipus 'Gestió de XXX' tenen associats els casos d'ús d'alta, llistat, consulta/modificació i baixa de l'element que s'estigui gestionant.

Casos d'ús dels actors

No tots els actors poden executar els mateixos casos d'ús. Com s'ha dit anteriorment, els actors tenen unes atribucions molt concretes dintre del sistema.

Així doncs, els casos d'ús que cada actor pot executar són els següents:

S'ha de tenir en compte que per gestió entenem les operacions d'afegir, llistar, consultar, modificar i eliminar.

A no ser que s'indiqui el contrari, donar de baixa significa eliminar físicament de la base de dades i en cascada tota la informació afectada per la baixa d'un registre.

Actor gestor de tests

1. Gestió de matèries
   1.1. Alta de matèria
   1.2. Llistat de matèries
   1.3. Consulta/Modificació de matèria
   1.4. Baixa de matèries
2. Gestió de blocs
3. Gestió de temes
4. **Gestió de tests**
   4.1.Publicar un test real
5. **Gestió de preguntes**

Actor alumne

6. **Executar test de pràctiques**
7. Executar test real
8. **Veure històric de qualificacions**

Actor administrador

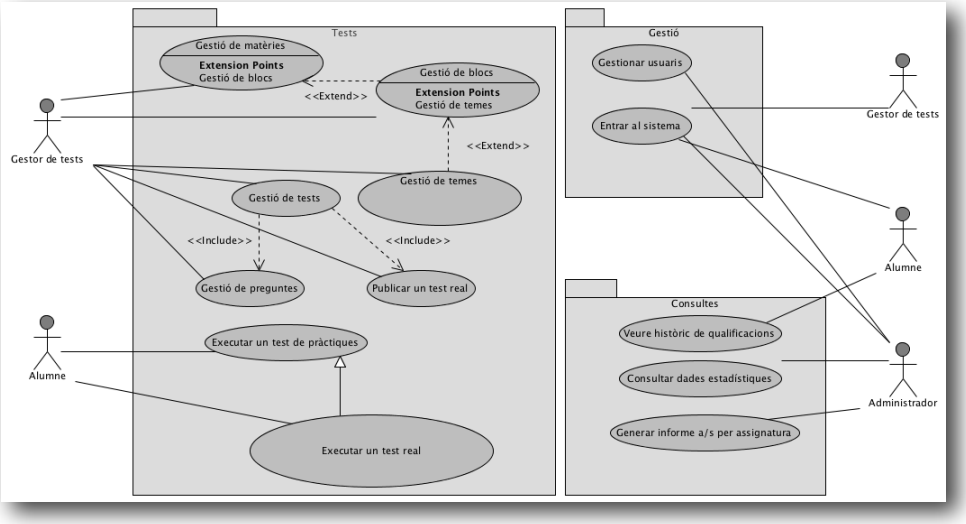
9. **Gestionar usuaris**
10. Consultar dades estadístiques
11. Generar informe d'aprovats/ suspesos per assignatura

Tots els actors

12. **Entrar al sistema**

Els casos d'ús que estan en **negreta** són els que s'han implementat en el TFC. La càrrega de la implementació dels casos d'ús assenyalats d'aquesta manera representa aproximadament el 80% de la càrrega total que seria implementar-los tots.

En aquesta imatge veurem aquesta distribució de casos d'ús en actors de forma gràfica.



Il·lustració Separació de casos d'ús en funció dels actors

Descripció dels casos d'ús

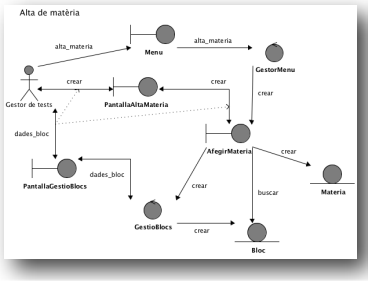
En aquest punt farem una descripció des del punt de vista de l'anàlisi dels casos d'ús que s'han marcat en **negreta** en la secció anterior. El format emprat per documentar els casos d'ús és el d'una taula amb diferents apartats on s'indica la descripció del cas d'ús, els actors que participen, pre i post condicions, etc...

Al llarg de la realització del TFC s'ha intentat evitar donar informació que fos redundant.
Això significa que en la PAC de l'anàlisi, només es van definir completament els mètodes que feien referència a la gestió de l'apartat de matèries.
És per aquest motiu que, tot i no estar desenvolupats, s'afegeixen en aquest apartat, ja que la resta de casos d'ús del tipus 'Destió de XXX' fan referència als casos d'ús esmentats en la gestió de matèries.

Alta de matèria

CAS D'ÚS: Alta de matèria	CODI: 1.1
Resum	dóna d'alta una matèria a la base de dades
Paper en el treball de l'usuari	esporàdicament usat pel gestor de tests quan s'ha de donar d'alta una matèria.
Actors	Gestor de tests
Casos d'ús relacionats	Gestió de blocs
Precondició	La matèria no existeix a la base de dades
Postcondició	La matèria està guardada a base de dades i pot ser usada
Passos d'execució	El gestor de tests fa clic al menú per donar d'alta una matèria. A continuació se li obre una pantalla des de la que por introduir les dades de la matèria (el nom de la matèria per tants idiomes com estigui configurada l'aplicació). Si fa clic al botó de guardar s'emmagatzema a la base de dades. Si fa clic a cancel·lar, se surt de la pantalla sense haver guardat res. Si falta per donar d'alta la matèria en algun dels idiomes configurats per l'aplicació, s'avisa de la circumstància però es deixa guardar igualment. Si ha de sortir el nom de la matèria ho farà amb l'idioma per defecte (que aquest sí que s'haurà d'emplenar obligatòriament).
Excepcions	Si no s'ha emplenat el nom de la matèria en l'idioma per defecte, el sistema obliga a introduir-lo.

Diagrama del cas d'ús

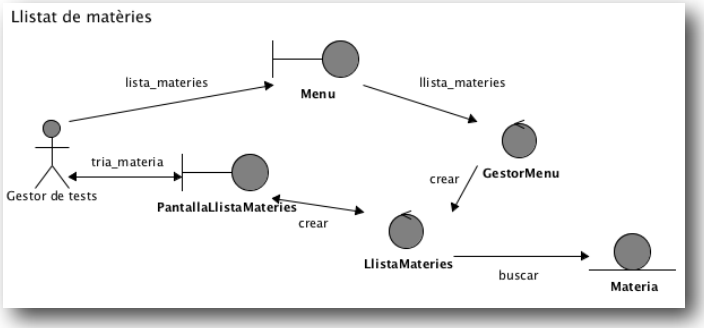


Il·lustració Diagrama de col·laboració d'alta de matèria

Llistat de matèries

CAS D'ÚS: Llistat de matèries	CODI: 1.2
Resum	treu un llistat per pantalla de les matèries donades d'alta al sistema
Paper en el treball de l'usuari	esporàdicament usat pel gestor de tests com a pas previ per consultar, modificar o eliminar alguna de les matèries donades d'alta.
Actors	Gestor de tests
Casos d'ús relacionats	cap
Precondició	cap
Postcondició	la pantalla mostra un llistat amb les matèries donades d'alta o res si no n'hi ha cap que s'hagi donat d'alta
Passos d'execució	L'usuari clica al menú per veure les matèries donades d'alta.

Diagrama del cas d'ús

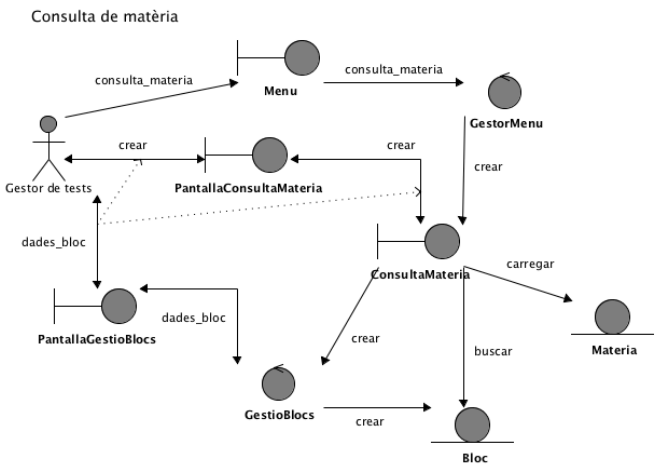


Il·lustració Diagrama de col·laboració de llistat de matèries

Consulta/Modificació de matèria

CAS D'ÚS: Consulta/Modificació de matèria	CODI: 1.3
Resum	veu les dades d'una matèria
Paper en el treball de l'usuari	esporàdicament usat pel gestor de tests per comprovar les dades complertes d'una matèria.
Actors	Gestor de tests
Casos d'ús relacionats	"Gestió de blocs - Alta"
Precondició	la matèria ha d'existir a la base de dades
Postcondició	cap
Passos d'execució	l'usuari selecciona la matèria que vol consultar i fa clic al botó d'editar. Aleshores se li presenta una pantalla amb tota la informació pertanyent a la matèria que, a més del nom, també comprèn els blocs que composen la matèria (si és que n'hi ha de donats d'alta associats a la matèria). En aquest punt es poden afegir o eliminar els blocs que composen una matèria.

Diagrama del cas d'ús

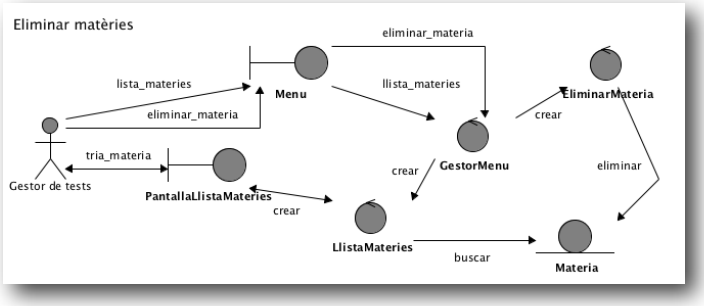


Il·lustració Diagrama de col·laboració consulta de matèria

Baixa de matèria

CAS D'ÚS: Baixa de matèria	CODI: 1.4
Resum	elimina una matèria de la base de dades
Paper en el treball de l'usuari	molt esporàdicament usat pel gestor de tests per donar de baixa una matèria. Quan una matèria es dóna de baixa, s'elimina de la base de dades tota la informació que pogués estar relacionada amb aquesta: blocs, temes, tests que tenen per assignatura l'eliminada, etc…
Actors	Gestor de tests
Casos d'ús relacionats	cap
Precondició	la matèria està donada d'alta
Postcondició	la matèria ha estat eliminada de la base de dades
Passos d'execució	l'usuari selecciona una matèria i prem el botó d'eliminar. El fet d'eliminar una matèria és un pas molt crític, pel que abans d'executar-se l'operació se li preguntarà a l'usuari si realment vol eliminar la matèria. Si contesta que sí, la matèria i tot el que està relacionada amb ella serà eliminat. Si contesta que no, es cancel·larà l'operació.

Diagrama del cas d'ús



Il·lustració Diagrama de col·laboració d'eliminar matèria

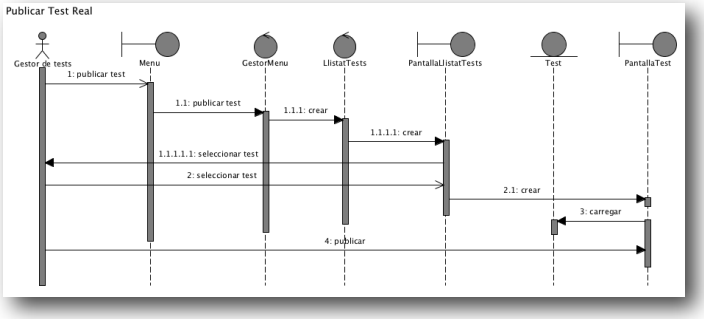
Gestió de tests

CAS D'ÚS: Gestió de tests	CODI: 4
Resum	gestiona els tests
Paper en el treball de l'usuari	habitual per al gestor de tests
Actors	Gestor de tests
Casos d'ús relacionats	Gestió de preguntes
Precondició	en funció de l'operació de manteniment de test a realitzar. Es pot prendre de referència les precondicions de les operacions de gestió de matèries
Postcondició	igual que la precondició
Passos d'execució	el test és l'objecte central de l'aplicació CreaTest. Un test constarà d'un nom, un nombre de preguntes, tindrà associada una assignatura a partir de la que se li podran associar blocs. Un test podrà ser classificat com a real o de pràctiques, així un alumne només podrà veure els que són de pràctiques. També se li podran donar regles de puntuació. Per exemple, quant descompten les preguntes fallades, si descompten les preguntes no contestades, etc…

Publicar un test real

CAS D'ÚS: Publicar un test real	CODI: 4.1
Resum	fa que un test marcat com a real aparegui com a executable per als alumnes
Paper en el treball de l'usuari	esporàdicament quan un gestor de tests vol que un test sigui avaluable per a les notes dels alumnes
Actors	Gestor de tests
Casos d'ús relacionats	"Gestió de tests - Modificar"
Precondició	el test existeix a la base de dades i està donat d'alta com a test 'real'
Postcondició	el test és publicat per a la seva execució
Passos d'execució	l'usuari selecciona un test marcat com a real de la llista total de tests. Quan el modifica, té l'opció de poder seleccionar un dia concret, una hora d'inici, una de fi i una duració. Això significa que al dia i hora seleccionat i fins l'hora de fi, els alumnes que cursin aquella matèria tindran disponible per executar el test publicat.

Diagrama del cas d'ús

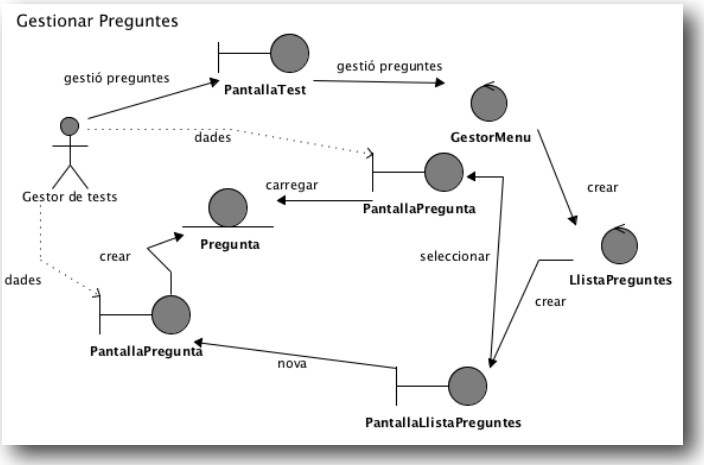


Il·lustració Diagrama de seqüència de publicar test real

Gestió de preguntes

CAS D'ÚS: Gestió de preguntes	CODI: 5
Resum	Quan es crea/modifica un test s'han de poder generar les preguntes que formaran part del mateix
Paper en el treball de l'usuari	habitual al llarg de la creació d'un test
Actors	Gestor de tests
Casos d'ús relacionats	"Gestió de tests - Alta", "Gestió de Tests - Modificació"
Precondició	en funció de la tasca de manteniment aplicada a la pregunta. Es poden prendre com a referència les precondicions de les operacions de gestió de matèries
Postcondició	igual que la precondició
Passos d'execució	Quan s'ha guardat un test s'obre la possibilitat de donar d'alta les preguntes que formaran part del test. En aquesta situació es podran donar d'alta, llistar, modificar o esborrar les preguntes que hi formen part. Una pregunta constarà de la pregunta pròpiament en forma de text, un apartat d'ajuda textual en forma de pista i un llistat de possibles respostes, només una de les quals és la certa.

Diagrama del cas d'ús

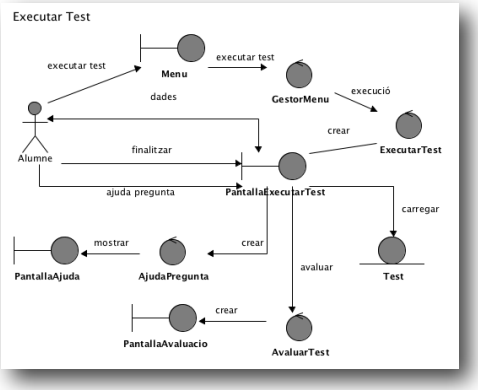


Il·lustració Diagrama de col·laboració de gestionar preguntes

Executar test de pràctiques

CAS D'ÚS: Executar test de pràctiques	CODI: 6
Resum	l'alumne executa un test classificat com 'de pràctiques'
Paper en el treball de l'usuari	habitual per controlar l'avançament del coneixement de la matèria
Actors	Alumne
Casos d'ús relacionats	cap
Precondició	l'alumne està associat amb la matèria de la que executarà el test
Postcondició	l'alumne ha executat el test de pràctiques i s'ha emmagatzemat la seva nota i el temps que ha trigat en fer-lo
Passos d'execució	l'usuari entra en el menú d'executar test de pràctiques. Allí té disponibles els tests que pot executar en funció de les assignatures que tingui associades. Els tests es poden filtrar mer matèries. un cop seleccionat un test fa clic al botó de començar. A la pantalla d'execució de test té la pregunta i un llistat de respostes. Al costat de cada una d'elles hi ha un botó de radi per poder-ne seleccionar només una. Hi ha un botó que serveix per demanar ajuda de la pregunta actual. També té tres botons més, avant, endarrere i acabar test, que provoquen que es vagi a la pregunta anterior, a la següent o que es doni el test per acabat. Un cop acabat el test, apareix una pantalla resum indicant la nota, el temps trigat i una relació de les preguntes fallades. Amb aquesta relació, l'alumne podrà veure quines són les preguntes que ha fallat o no contestat i també quina era la resposta correcta. Un test de pràctiques es pot executar tots els cops que es vulgui.

Diagrama del cas d'ús

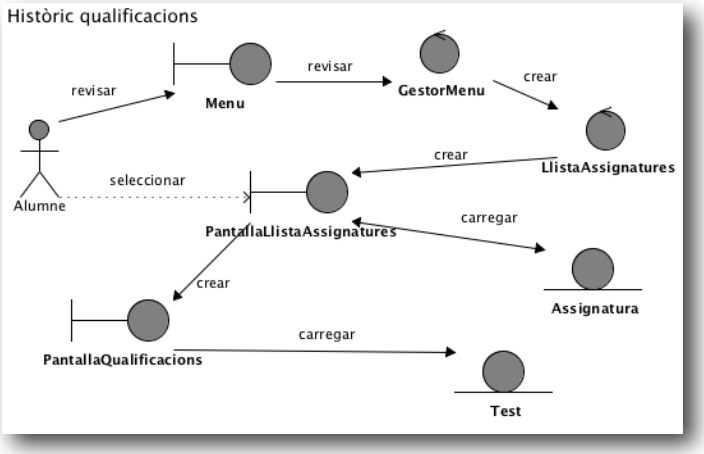


Il·lustració Diagrama de col·laboració d'executar test

Veure històric de qualificacions

CAS D'ÚS: Veure històric de qualificacions	CODI: 8
Resum	l'alumne pot revisar les diferents notes que ha tret i els temps que ha trigat en realitzar els tests
Paper en el treball de l'usuari	habitualment per fer el seguiment d'una assignatura o del conjunt d'assignatures
Actors	Alumne
Casos d'ús relacionats	cap
Precondició	cap
Postcondició	l'alumne ha revisat les qualificacions d'una matèria
Passos d'execució	des del menú, l'alumne escull l'opció de revisar les notes de les matèries que té associades. Li apareix una pantalla des de la que pot escollir una matèria i, fent clic a la icona de veure qualificacions accedeix a una altra pantalla on hi ha una taula ordenada per data de tots els tests que hagi executat d'aquella assignatura. A la taula apareixerà la data, el nom del test, el temps que ha trigat en executar-lo, si era un test de pràctiques o no i la nota.

Diagrama del cas d'ús

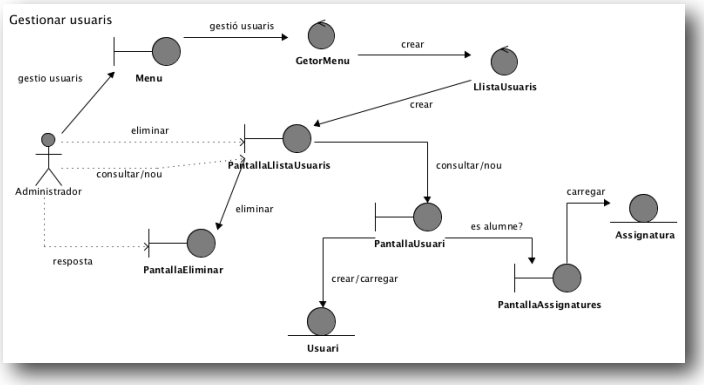


Il·lustració Diagrama de col·laboració d'històric de qualificacions

Gestionar usuaris

CAS D'ÚS: Gestionar usuaris	CODI: 9
Resum	permet a l'usuari administrador realitzar el manteniment dels usuaris de l'aplicació
Paper en el treball de l'usuari	esporàdicament usat per realitzar manteniment dels usuaris. Hi poden haver pics d'ús alts (per exemple, a l'inici d'un curs si hi ha molts alumnes nous)
Actors	Administrador
Casos d'ús relacionats	cap
Precondició	en funció de la tasca de manteniment aplicada a l'usuari. Es poden prendre com a referència les precondicions de les operacions de gestió de matèries
Postcondició	igual que la precondició
Passos d'execució	l'usuari clica sobre l'opció de manteniment dels usuaris i li apareix una pantalla amb els usuaris donats d'alta. Aquests es poden filtrar per tipus. Si són alumnes, es poden filtrar també per matèria que tinguin associada. Als usuaris se'ls hi assigna un o més tipus quan se'ls dóna d'alta. En el cas que es doni d'alta un alumne, també s'haurà d'indicar quines són les matèries que té associades. Si s'elimina un usuari, s'elimina també tota la informació que tingui associada.

Diagrama del cas d'ús

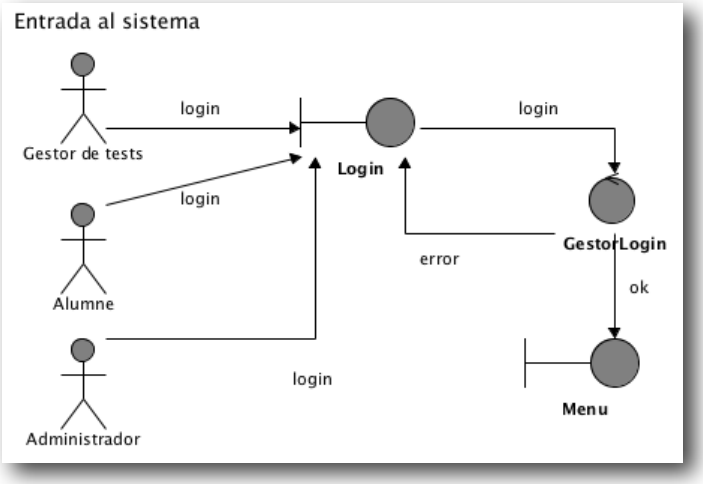


Il·lustració Diagrama de col·laboració de gestionar usuaris

Entrar al sistema

CAS D'ÚS: Entrar al sistema	CODI: 11
Resum	l'usuari es logina per poder entrar al sistema
Paper en el treball de l'usuari	molt habitual per poder usar el sistema
Actors	administrador, alumne, gestor de tests
Casos d'ús relacionats	cap
Precondició	l'usuari existeix en el sistema
Postcondició	l'usuari ha entrat al sistema
Passos d'execució	A la pantalla d'entrada l'usuari introdueix el seu login i password. Si l'usuari existeix al sistema, entra a l'aplicació. En cas contrari, no pot entrar.

Diagrama del cas d'ús



Il·lustració Diagrama de col·laboració d'entrada al sistema

Especificació dels casos d'ús

Els casos d'ús comentats en el punt anterior són descripcions que ens diuen què és el que han de fer, però no ens expliquen pas com fer-ho, més que amb diagrames de col·laboració entre diferents classes de frontera, entitat i control. Al llarg de la fase de disseny s'estudia, tenint en compte quina serà la tecnologia emprada per desenvolupar l'aplicació, com s'han de desenvolupar les especificacions donades a la fase de l'anàlisi.

En aquest punt veurem l'especificació des del punt de vista del disseny dels casos d'ús vistos prèviament. Constarà d'una especificació textual, tabulada per tal d'indicar condicions subordinades a d'altres, del procés que s'estigui duent a terme.

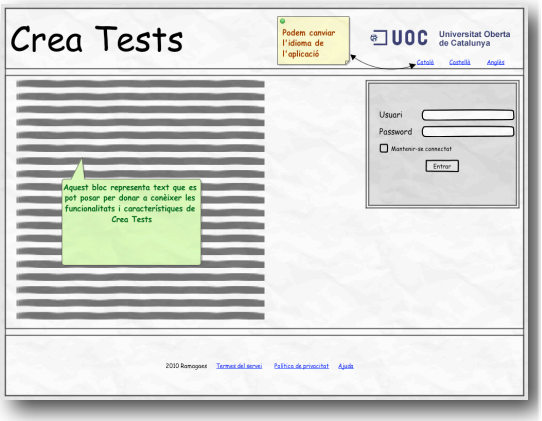
Entrada al sistema

Quan l'usuari intenta entrar a l'aplicació es crea una instància de la classe `FormLogin` que fa aparèixer a la pantalla un formulari amb dos camps per emplenar (usuari i password) i un botó per executar l'operació `checkLogin`.

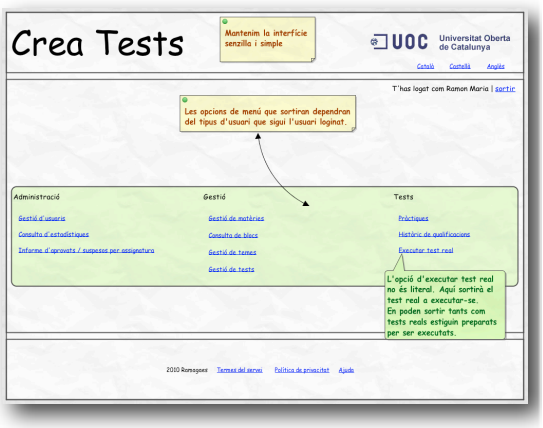
1. L'usuari entra el seu nom d'usuari i el seu password i clica el botó d'entrar. Quan es prem el botó d'entrar, s'executa l'operació `checkLogin` del bean `UserBean`.

    a) Si l'usuari no s'ha trobat a partir de la combinació d'usuari i password, es crea una instància de la la classe `Failure` i es mostra un missatge indicant el problema. FI

    b) Si l'usuari s'ha trobat, es guarda la seva informació en un bean de sessió per que estigui disponible al llarg de la sessió de l'usuari. Es crea una instància de la la classe `FormPrincipal` i es mostra una pantalla amb les opcions de menú en funció del rol que tingui l'usuari loginat.



En aquest projecte de pantalla es poden veure tots els elements comentats en la descripció del cas d'ús d'entrada al sistema.



El disseny de la interfície s'ha procurat que fos net i simple per què les seves opcions fossin fàcilment localitzables.

Il·lustració prototip entrada al sistema i pantalla principal

Sortida del sistema

En qualsevol moment l'usuari pot sortir de l'aplicació.

1. L'usuari fa clic sobre l'opció sortir que hi ha a totes les pàgines de l'aplicació.
2. S'elimina de la sessió les dades que hi poguessin haver.
3. Es crea una instància de la la classe `FormLogin` i es mostra per pantalla. FI

Llistat de tests

Quan l'usuari selecciona l'opció de gestió de tests, es crea una instància de la la classe `FormLlistaTests`. A la pantalla que apareix hi ha un control per poder escollir la matèria i filtrar els tests d'acord amb el valor seleccionat. A més, també hi ha un control que permet veure els tests que són reals, els de pràctiques, o bé ambdós. Els filtres són acumulatius.

1. L'usuari prem l'opció de gestió de tests des del menú principal de l'aplicació. Es crea una instància de la la classe `FormLlistaTests` i es crea un bean request `TestBean`.
2. <u>Si l'usuari té el rol de gestor de tests</u>

   a) El `TestBean` retorna tots els tests que existeixen a l'aplicació. `FormLlistaTests` mostra un botó de nou test, un botó de sortir i els tests retornats per `TestBean` els mostra en forma de taula amb les capçaleres següents:

   - **Matèria**: matèria a la que pertany el test
   - **Test**: el nom del test
   - **Real**: indica si és real o de pràctiques
   - **Editar**: enllaç que ens permetrà editar el test
   - **Eliminar**: enllaç que ens permetrà eliminar el test

   b) Si l'usuari selecciona una matèria del combo que apareix, la llista queda actualitzada amb els tests que són de la matèria seleccionada.

c) Si l'usuari fa ús dels checks que distingeixen entre tests reals i de pràctiques, la llista quedarà modificada per reflectir aquest filtratge.

d) Si l'usuari prem el botó de sortir, `TestBean` es destrueix, es crea una instància de la la classe `FormPrincipal` i es mostra. FI

e) Si l'usuari prem el botó de nou es crea una instància de la un nou `TestBean` que tindrà un àmbit d'aplicació que durarà fins que s'acabi de crear el test. També Es crearà una instància de la la classe `FormGestioTest` que tindrà associat aquest bean creat i es mostrarà. FI

f) Si l'usuari prem l'enllaç d'editar, el test al que fa referència l'enllaç d'editar es carregarà en un nou `TestBean` que tindrà un àmbit d'aplicació que durarà fins que s'acabi d'editar el test. També es crearà una instància de la la classe `FormGestioTest` que tindrà associat aquest bean creat i es mostrarà. FI

g) Si l'usuari fa clic a sobre de l'enllaç d'eliminar, la classe `FormLlistaTests` li preguntarà a l'usuari si realment vol eliminar el test. En cas que contesti que sí, el test serà eliminat i la taula que es mostra actualitzada per reflectir aquesta circumstància. FI

3. Si l'usuari és de tipus alumne

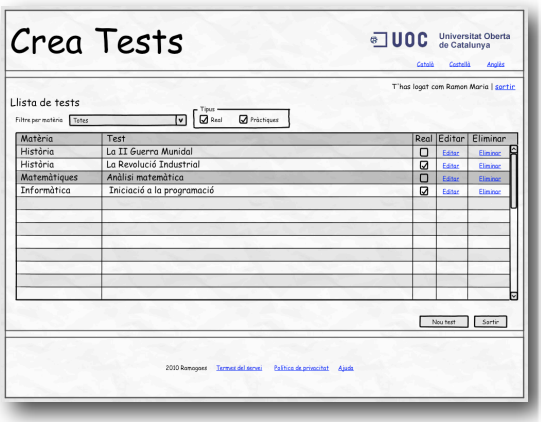
a) El `TestBean` retorna tots els tests de pràctiques que l'alumne té disponibles per a les matèries a les que està associat. `FormLlistaTests` els mostra amb aquesta informació:

- **Matèria**: nom de la matèria a la que pertany el test
- **Test**: nom del test
- **Executar**: enllaç que permet executar el test

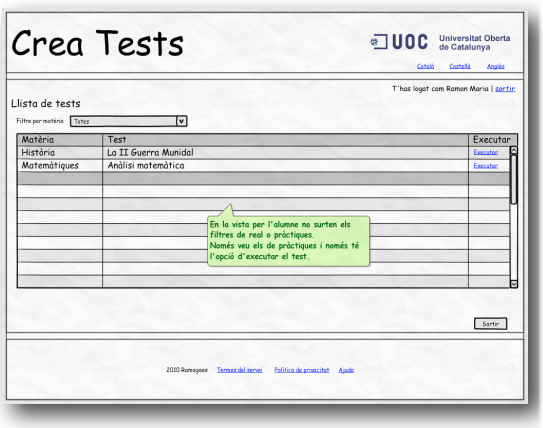
b) Si l'usuari canvia el filtre de les matèries, veurà només els tests corresponents a la matèria seleccionada.

c) Si l'usuari prem el botó de Sortir, torna a la pantalla del menú principal. FI

d) Si l'usuari clica sobre l'enllaç d'executar el test, es crea una instància de la la classe `FormExecutarTest` i es carrega amb les dades del test per poder ser executat. FI



En aquest projecte de pantalla es poden veure tots els elements comentats en la descripció del cas d'ús per a quan l'usuari és de tipus gestor de tests.



Aquesta és la pantalla de llista de tests si l'usuari és un alumne. Correspon amb les opcions esmentades en la descripció del cas d'ús.

Il·lustració prototip llistat de tests

Gestió de tests

Si l'usuari ha premut l'enllaç d'editar o el botó de test nou a la pantalla que mostra els tests, es crea una instància de la la classe `FormGestioTest`. Aquesta classe té associat un `TestBean` que estarà buit si s'ha premut el botó de nou, o estarà carregat amb les dades del test al que feia referència l'enllaç d'editar.

1. L'usuari es troba amb una interfície amb dues pestanyes des de les que pot donar d'alta o modificar les dades del test. Concretament aquestes dades són les següents:

   a) Pestanya **Dades Generals**

      - Nom
      - Nombre de preguntes
      - Matèria
      - Blocs seleccionats
      - Real
      - Data de publicació
      - Hora de publicació
      - Duració (minuts)

   b) Pestanya **Dades de Puntuació**

      - Màxim nombre de preguntes fallades
      - Les preguntes fallades descompten
      - Les preguntes no contestades descompten
      - Punts que descompten les preguntes no contestades

2. Si l'usuari modifica la matèria, la classe `FormGestioTest` li preguntarà si realment la vol canviar, ja que aquesta acció implica eliminar els blocs que conté el test i totes les preguntes que tingui el test.

   a) **L'usuari contesta que sí vol canviar la matèria**: aleshores el bean `BlocsBean` carregarà els blocs pertanyents a la nova matèria seleccionada. S'eliminaran de la base de dades les preguntes del test i

els blocs que estiguin seleccionats deixaran d'estar-ho per poder escollir-ne de nous de la matèria nova.

b) **L'usuari contesta que no**: aleshores tot continua com fins el moment.

3. Si l'usuari marca el test com a real, li apareixeran els controls de **Data de publicació**, **Hora de publicació** i **Duració (minuts)** per què pugui seleccionar quan vol que el test aparegui per ser executat. Amb aquest canvi s'aconsegueix que el test no aparegui al llistat de tests per executar dels alumnes.

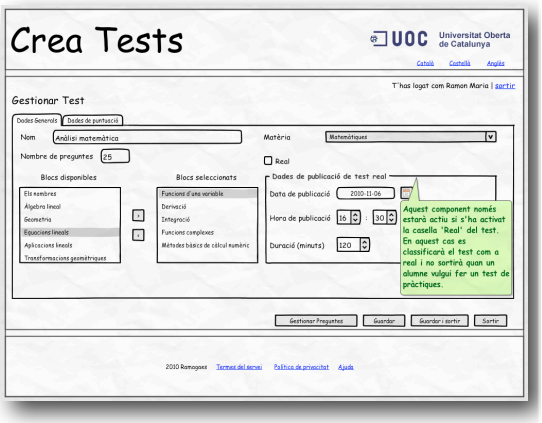
4. Un cop introduïdes les dades, l'usuari té quatre opcions en forma de quatre botons:

I. **Gestionar Preguntes**: si l'usuari prem aquest botó s'emmagatzema el test i es crea una instància de la la classe `FormLlistaPreguntes`. La classe té associat un bean `PreguntaBean` que li retornarà la llista de preguntes per a aquest test. A partir d'allí l'usuari pot seleccionar una pregunta i editar-la, crear-ne de noves (fins al límit de preguntes del test), eliminar-ne o tornar cap a la pantalla de gestió del test.

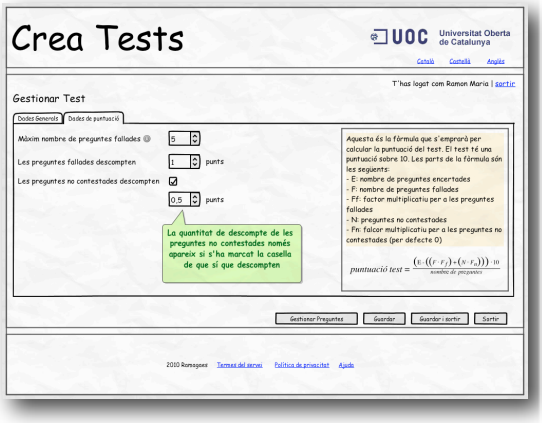
II. **Guardar**: si l'usuari prem aquest botó, el test queda emmagatzemat a la base de dades i es continua a la mateixa pantalla.

III. **Guardar i sortir**: amb aquesta opció, es guarda el test a la base de dades i després se surt a la pantalla `FormLlistaTests`. FI

IV. **Sortir**: si l'usuari decideix sortir i hi ha hagut canvis, se li preguntarà si vol guardar abans de sortir. En cas afirmatiu, es guardarà el test i se sortirà a la pantalla `FormLlistaTests`. En cas negatiu, se sortirà a la pantalla `FormLlistaTests` i es perdran els canvis realitzats. FI



Aquests són els esborranys comentats de les pantalles que representen la classe `FormGestioTest`. Es poden veure les dues pestanyes comentades en el cas d'ús i tots els controls esmentats. En la de puntuació s'aprecia una explicació de com es calcula la qualificació d'un test (que és sempre sobre 10).

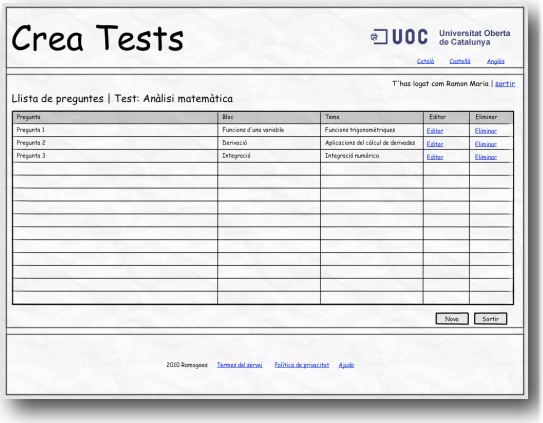


Il·lustració prototip creació / modificació test

Llistat de preguntes

Quan l'usuari prem el botó de gestionar les preguntes a la classe `FormGestioTest`, crea una instància de la classe `FormLlistaPreguntes`, a la que acompanya un bean de tipus `PreguntaBean` i el `TestBean` creat quan hem estrat a `FormGestioBean`, i li mostra una pantalla amb un llistat de les preguntes creades per al test que s'està editant o creant. A partir d'allí l'usuari pot seleccionar una pregunta i editar-la, crear-ne de noves (fins al límit de preguntes del test), eliminar-ne o tornar cap a la pantalla de gestió del test.

1. Si l'usuari edita una pregunta o en crea una de nova, es crea una instància de la la classe `FormGestioPregunta`. La diferència serà que si es nova, el bean `PreguntaBean` estarà buit, mentre que si s'edita una de ja existent, es carregarà el contingut d'aquesta i es mostrarà.
2. Si l'usuari vol eliminar una pregunta, `FormLlistaPreguntes` li demanarà si està segur. En cas afirmatiu eliminarà la pregunta de la base de dades i refrescarà la taula per reflectir el canvi.
3. Si l'usuari vol sortir, reapareixerà a la pantalla `FormGestioTest`. FI



La pantalla que mostra les diferents preguntes que pot tenir un test i les diferents opcions de les que es disposen.

Il·lustració prototip llistat de preguntes

Gestió de pregunta

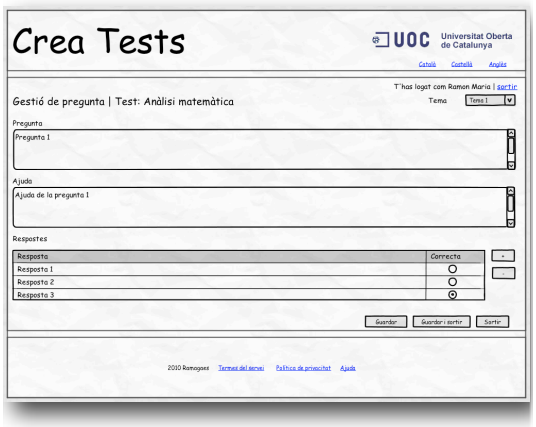
Quan l'usuari ha premut l'enllaç d'edició de pregunta o el botó de nova pregunta, la classe `FormGestioPregunta` es crea una instància de la i es carrega amb les dades de la pregunta seleccionada o es manté buida en cas que aquesta sigui nova. Al mateix temps es crea un bean del tipus `RespostaBean` que emmagatzemarà les respostes de la pregunta.

1. L'usuari apareix a la pantalla de gestió de pregunta i té els següents camps per poder emplenar:
   - **Pregunta**: camp de text on es realitza la pregunta que s'haurà de contestar
   - **Tema**: Tema de la pregunta. Es carrega a partir dels blocs als que pertany el test
   - **Ajuda**: camp de text on es pot escriure una ajuda sobre la pregunta
   - **Respostes**: taula on hi ha les possibles respostes a la pregunta formulada. Les preguntes es creen fent clic al botó de '+' que hi ha al costat de la taula. Si se'n vol eliminar alguna, se selecciona i es fa clic al botó '-'. Cada resposta té un text amb la resposta i un botó de radi que permet escollir quina és la correcta.
2. L'usuari té tres opcions:

   I. **Guardar**: s'emmagatzema la pregunta a la base de dades amb les seves respostes i s'associa al test al que pertany. No se surt de la pantalla de gestió de pregunta.

   II. **Guardar i sortir**: la mateixa funcionalitat que **Guardar** i, a més, se surt a la pantalla de llistat de preguntes. FI

   III. **Sortir**: si hi ha hagut algun canvi, es pregunta a l'usuari si el vol guardar. En cas afirmatiu es procedeix com a **Guardar i sortir**. Si diu que no, els canvis es perden i se surt a la pantalla de llista de preguntes. FI



Aquesta és la pantalla de gestió de pregunta on es poden veure tots els elements mencionats en la descripció del cas d 'ús.

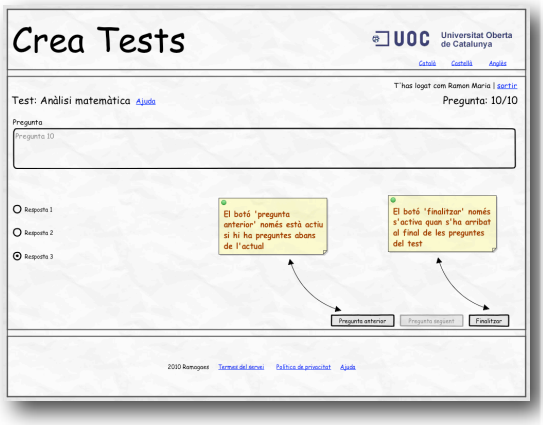
Il·lustració prototip gestió de pregunta

Executar test de pràctiques

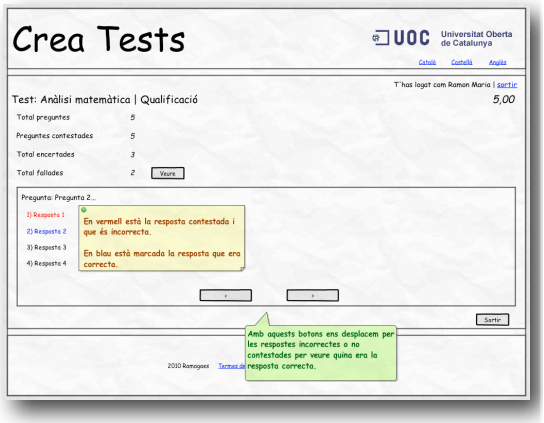
Quan un usuari alumne clica l'enllaç per executar el test en pràctiques, es crea una instància de la i es carrega la classe `FormExecutarTest`. Hi ha un `TestBean` que s'encarregarà de mantenir la traça de les preguntes contestades

per l'alumne fins el moment en què decideixi acabar-lo i veure la seva qualificació.

1. L'usuari entra a la pantalla d'execució de test i allí es troba amb els següents elements:
    - Un enllaç per poder veure l'ajuda de la pregunta
    - Un espai per poder veure la pregunta que se li fa
    - Un espai on s'indica el total de preguntes i quina és la pregunta actual
    - Un espai per veure les respostes i seleccionar la correcta
    - Tres botons per poder navegar per les preguntes del test i acabar-lo
2. L'usuari contesta la pregunta o no en funció de si coneix la resposta. Llavors clica el botó Pregunta següent si encara li queden preguntes, o Pregunta anterior si pot i vol tornar endarrere alguna pregunta. En tot cas, en quan clica un dels dos botons, el bean `TestBean` guarda la resposta i carrega la pregunta seleccionada amb les seves respostes per que l'alumne la pugui contestar.
3. Quan l'usuari clica sobre el botó Finalitzar, es crea una instància de la la classe `FormQualificacioTest` i es passa a la pantalla on es veu la qualificació del test. Al mateix temps, `TestBean` calcula la puntuació final del test per poder-la mostrar.
4. A la pantalla de qualificació del test podem veure el següent:
    - La nota del test
    - El total de preguntes
    - Les preguntes contestades
    - Les no contestades
    - El total d'encertades
    - El total de fallades
    - Un botó per poder comprovar les preguntes fallades. Si l'usuari no ha fallat cap pregunta i les ha contestat totes, no sortirà el botó de veure les preguntes fallades.
    - Dos botons per navegar avant i endarrere de les preguntes fallades
    - Un botó de sortir
5. Si l'usuari fa clic sobre el botó de veure les preguntes fallades, `TestBean` carrega les preguntes fallades i les mostra de forma que la resposta contestada (i fallada) surt en vermell i la correcta en blau. Aquí l'usuari es pot anar desplaçant per les preguntes per veure quines són les respostes correctes.
6. Quan l'usuari fa clic a Sortir, surt a la pantalla de llistat de tests (com a mode alumne).



Aquests són els prototips de pantalles d'execució de test i de mostrar els resultats de l'execució del mateix. Conté les característiques esmentades en la descripció dels casos d'ús.

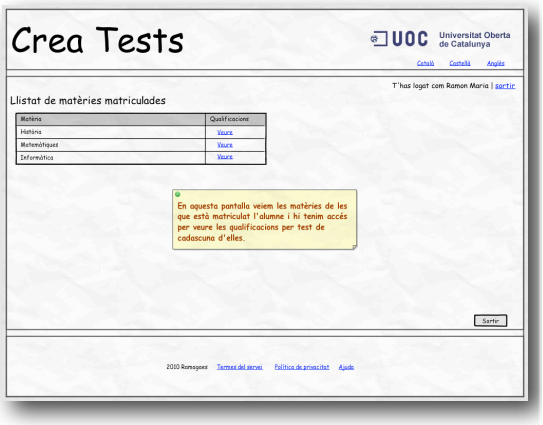


Il·lustració prototip execució de test de pràctiques i resultat del test

Llistat de matèries matriculades

Quan l'usuari clica sobre l'enllaç d'històric de qualificacions des del menú principal crea una instància de la classe `FormLlistaMateriesMatriculades`. El bean `UserBean` que té un rang d'aplicació de sessió retornarà totes les matèries de les que està matriculat l'alumne.

1. L'usuari pot fer clic sobre l'enllaç Veure que surt al costat de cada assignatura que tingui associada. En aquest cas, es crea una instància de la la classe `FormVeureQualificacions`. Amb aquesta classe hi haurà el bean `QualificacionsBean` que detallarà les notes de la matèria. FI
2. L'usuari pot fer clic sobre el botó de sortir. En aquest cas es destruiria la instància de la classe `FormLlistaMateriesMatriculades` i s'aniria a la pantalla del menú principal. FI



En aquest prototip de pantalla podem apreciar els elements comentats en la descripció del cas d'ús.

Il·lustració prototip llista matèries matriculades

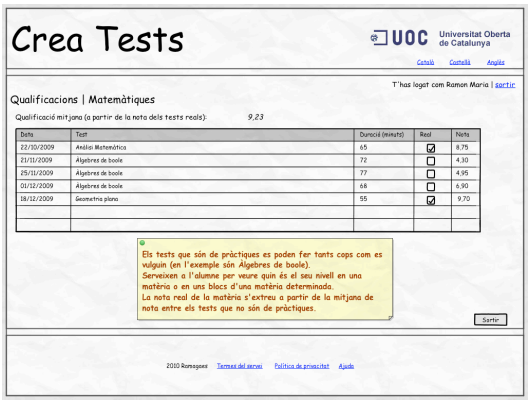
Veure històric de qualificacions

Quan l'usuari ha fet clic sobre l'enllaç de Veure al costat d'una de les matèries, es crea una instància de la la classe `FormVeureQualificacions`. Amb aquesta classe hi haurà el bean `QualificacionsBean` que detallarà les notes de la matèria. Aquesta és una pantalla de només consulta on l'usuari podrà veure la seva nota mitjana en la matèria (calculada a partir de fer la mitjana de les notes dels tests reals) i una llista on hi ha els tests que ha realitzat. Concretament la taula conté els camps següents:

- **Data**: la data de la realització del test
- **Test**: nom del test
- **Duració (minuts)**: temps en minuts que s'ha trigat en fer el test
- **Real**: informa de si era real o de pràctiques
- **Nota**: nota del test

L'usuari només té l'opció de fer clic al botó de sortir que hi ha a la pantalla.

1. L'usuari fa clic al botó de sortir i es destrueix la instància de la classe `FormVeureQualificacions`. Apareix a la pantalla de llistat de matèries matriculades. FI



La pantalla conté tots els controls esmentats al cas d'ús.

Il·lustració prototip llista de qualificacions

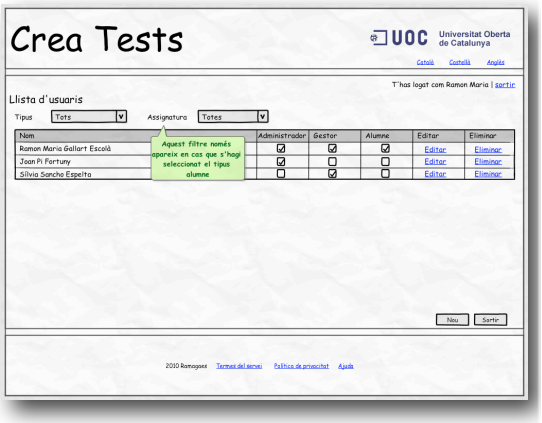
Llistat d'usuaris

Quan l'usuari fa clic a l'enllaç de gestió d'usuaris es crea una instància de la la classe `FormLlistaUsuaris`. Aquesta classe és acompanyada per un bean `UserBean` que retorna tots els usuaris existents al sistema. Aquests usuaris es mostren en una taula que conté la següent capçalera:

- **Nom**: nom de l'usuari
- **Administrador**: indica si és administrador
- **Gestor**: indica si és gestor
- **Alumne**: indica si és alumne
- **Editar**: enllaç per editar l'usuari
- **Eliminar**: enllaç per eliminar l'usuari

A més també disposa d'un botó Nou i d'un de Sortir.

1. Si l'usuari fa clic a sobre del botó Nou o de l'enllaç d'editar, es crea una instància de la la classe `FormGestioUsuari` que té associat un `UserBean`. Aquest bean estarà buit si s'ha fet clic al botó Nou o estarà carregat amb les dades de l'usuari si s'ha clicat a sobre de l'enllaç. FI
2. Si l'usuari fa clic a sobre de l'enllaç d'eliminar, se li preguntarà si realment desitja eliminar l'usuari. En cas afirmatiu s'eliminarà de la base de dades i es refrescarà la taula per reflectir el canvi.
3. Si l'usuari fa clic a sobre del botó de Sortir, la instància de `FormLlistaUsuaris` es destruirà i es passarà a la pantalla principal de l'aplicació. FI



La pantalla conté tots els controls esmentats al cas d'ús.

Il·lustració prototip llistat d'usuaris

Gestió d'usuari

Quan l'usuari clica el botó Nou o l'enllaç d'editar de `FormLlistaUsuaris` es crea una instància de la classe `FormGestioUsuari` que té associat un `UserBean`. Aquest bean estarà buit si s'ha fet clic al botó Nou o estarà carregat amb les dades de l'usuari si s'ha clicat a sobre de l'enllaç i emplenarà els controls de la pàgina. Tant si està editant com si està creant un usuari nou, l'usuari tindrà els següents controls de pantalla disponibles per poder incorporar informació de l'usuari:

- **Nom**: nom de l'usuari
- **Cognoms**: els seus cognoms
- **Usuari**: login de l'usuari
- **Password**: paraula clau
- **Repetir** password: sistema de seguretat per comprovar que el password entrat abans és correcte
- **DNI**: espai per guardar un número d'identificació
- **Observacions**: notes sobre l'usuari
- **Administrador**: indica si és administrador
- **Gestor**: indica si és gestor
- **Alumne**: indica si és de tipus alumne
- **Assignatures**: si és alumne, té l'opció de seleccionar les matèries de les que està matriculat

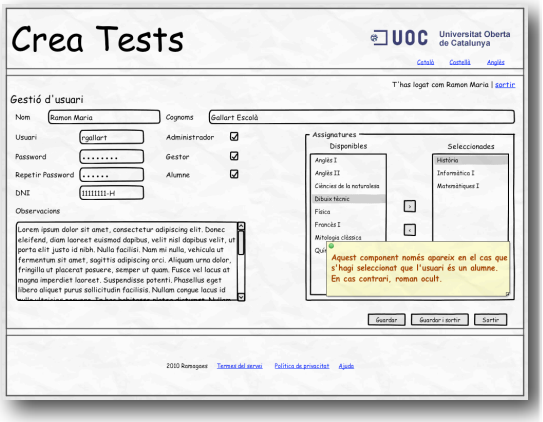
A més també disposa de tres botons, Guardar, Guardar i sortir i Sortir.

1. L'usuari entra la informació de l'usuari nou o modifica la d'un existent. Si selecciona que és de tipus alumne, també pot seleccionar les assignatures de les que està seleccionat. Després pot fer clic a un dels tres botons esmentats:

   I. **Guardar**: emmagatzema les dades de l'usuari i es queda a la mateixa pantalla.

   II. **Guardar i sortir**: emmagatzema les dades de l'usuari i surt a la pantalla de llista d'usuaris. FI

   III. **Sortir**: si hi ha hagut canvis, pregunta si es volen guardar. En cas afirmatiu es comporta com l'opció de guardar i sortir. En cas negatiu, es perden els canvis i se surt a la pantalla de llista d'usuaris.



La pantalla conté tots els controls esmentats al cas d'ús.

Il·lustració prototip consulta / modificació d'usuari

Disseny tècnic del sistema

En aquest apartat mostraré les diferents classes que han anat sorgint a l'aplicació; primer com a resultat de la fase de l'anàlisi i, després, degut al refinament fet a la fase del disseny.

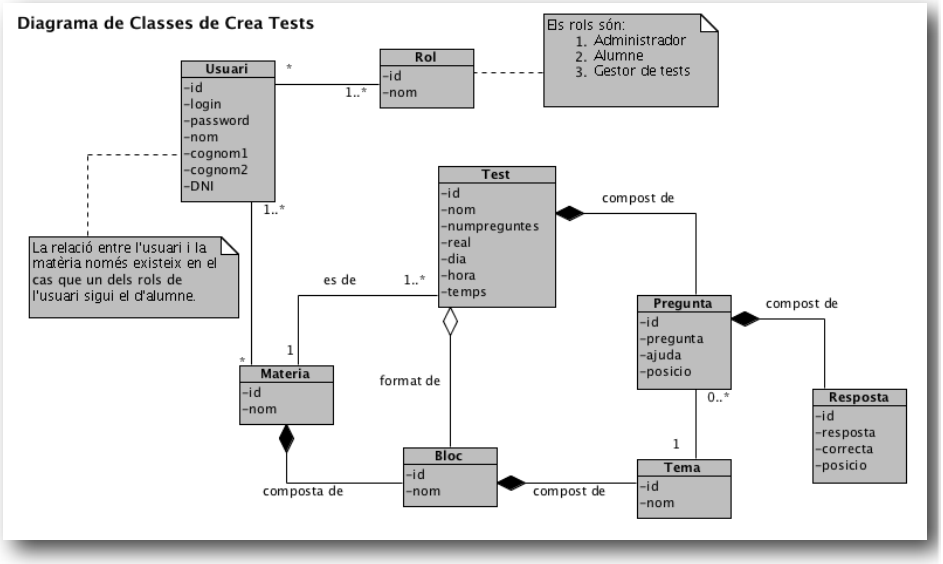
Durant la fase de l'anàlisi s'identifiquen principalment les classes del model del domini, això és, les classes entitat que defineixen el domini del problema que ha de resoldre l'aplicació.

Típicament es considera que el cost entre l'anàlisi i el disseny és d'1 a 5, cosa que ens indica que en el disseny hi haurà moltes més classes que les que s'han identificat en l'anàlisi.
Vegeu el mòdul 6 punt 1.2 de l'assignatura Enginyeria del Programari.

Al llarg de la fase de disseny, degut al refinament abans esmentat, la quantitat de classes s'incrementa, ja que no només s'especifiquen les classes del domini, si no que es creen classes de frontera i de control i, possiblement, classes associades a la pròpia tecnologia amb la que es desenvoluparà l'aplicació.

Entitats derivades de l'anàlisi

Aquestes són les classes que representen el model del domini de l'aplicació **CreaTests**.



Il·lustració diagrama estàtic d'anàlisi

La interpretació del diagrama és que una matèria està composta de blocs i aquests a la vegada estan compostos de temes. Un test és d'una matèria i està format de blocs. Al mateix temps està compost de preguntes que tenen respostes. Les preguntes estan associades a un tema concret.

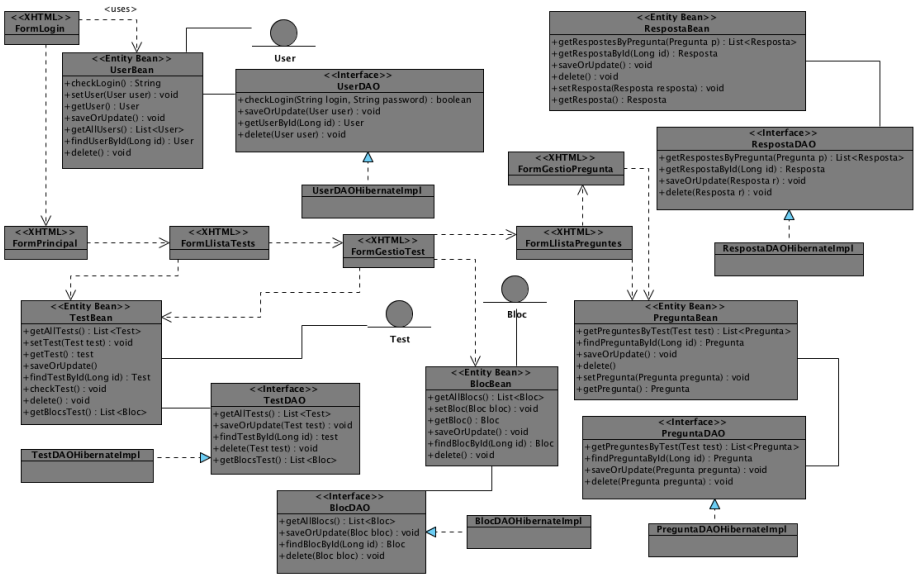
Un usuari pot tenir rols. Aquests rols poden ser els especificat al diagrama. Només si té el rol d'alumne, però, podrà estar associat a les matèries de les que està matriculat.

Diagrama de classes del disseny

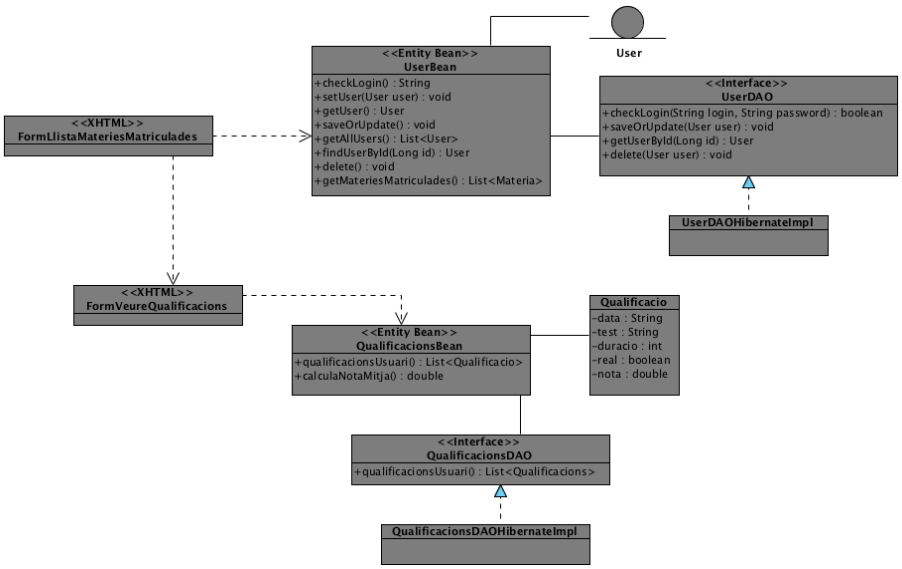
El diagrama de disseny proporciona més informació a l'hora de fer el desenvolupament de l'aplicació. Les classes definides a la fase d'anàlisi s'han incrementat i ara ja no són únicament les classes d'entitat, si no que també hi intervenen les classes de frontera, de control i classes que existeixen degut al bastiment o tecnologia concreta emprada per al desenvolupament de l'aplicació.

En el cas del conjunt de tecnologies emprades per desenvolupar l'aplicació, serien les classes amb estereotip de XHTML (que serien classes frontera) i d'Entity Bean, que serien classes de control juntament amb un fitxer de configuració XML que indicaria la navegabilitat de les pàgines dintre de l'aplicació.

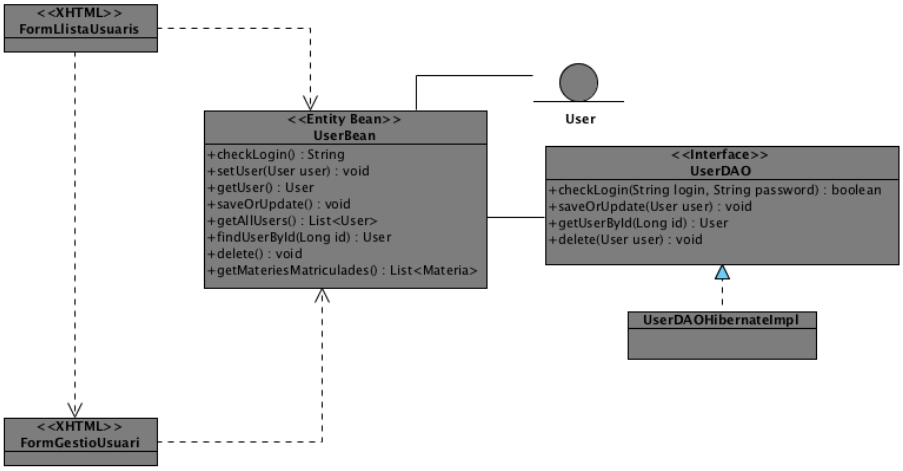
Els diagrames de classe estàtics de disseny, degut als motius exposats, acostumen a ser més grans i complexos. És per aquest motiu pel que s'han dividit en diferents diagrames per poder mostrar totes les classes identificades en una mida que sigui llegible en aquest document.



Il·lustració Classes de disseny de test i de login



Il·lustració Classes de disseny de qualificacions



Il·lustració Classes de disseny d'usuaris

Disseny de l'arquitectura de l'aplicació

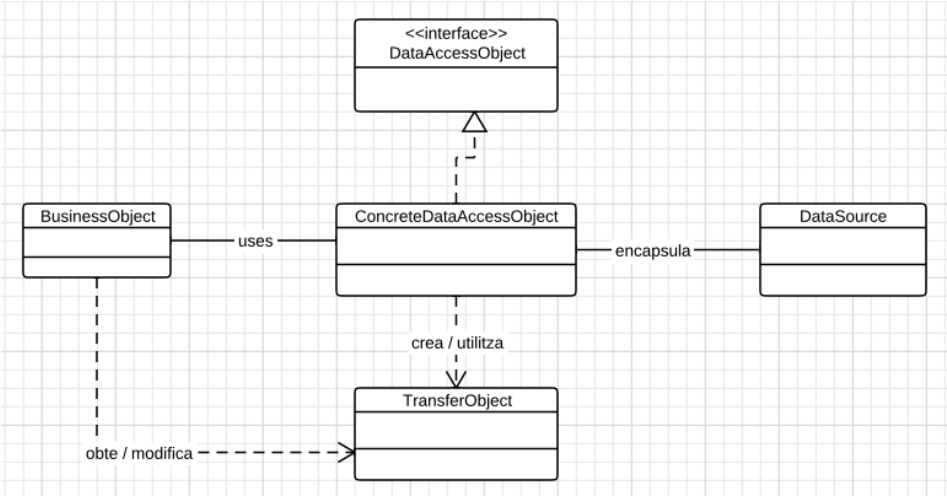
Tota aplicació es recolza en unes bases tècniques i tecnològiques que fan que funcioni i doni valor a l'àmbit per la qual ha estat creada. En aquest punt veurem quins són els fonaments arquitectònics de **CreaTests**.

Podeu una explicació del patró DAO a: http://en.wikipedia.org/wiki/Data_access_object

Arquitectura de la persistència amb Hibernate

La persistència en una aplicació permet que les dades que genera quedin disponibles per usos i consultes futures. Per poder persistir aquestes dades, és necessari que existeixi un mecanisme que permeti l'emmagatzemament de les mateixes de forma perdurable. Habitualment el sistema escollit per dur a terme aquesta persistència és mitjançant l'ús d'un sistema gestor de bases de dades (SGBD). Així doncs, el motor de persistència de l'aplicació és un MySQL versió 5.1. L'accés a dades per part de l'aplicació es realitza mitjançant el patró de disseny DAO (Data Access Object). Aquest patró permet que l'accés a dades per part de l'aplicació sigui transparent respecte al sistema gestor de dades que emmagatzemi les dades de l'aplicació. Amb el patró DAO es defineix una interfície que conté els mètodes que seran implementats per una classe concreta que atacarà a la base de dades.

Podeu trobar informació sobre com crear un DAO genèric, amb Hibernate 3 i Spring 3, en forma de tutorial a la meva web: http://www.ramagaes.info/rmgs/blog/2010/12/01/com-crear-un-dao-generic-amb-hibernate-i-spring/



Il·lustració Diagrama clàssic del patró DAO

L'aplicació usa el bastiment Hibernate 3. Aquest bastiment permet atacar bases de dades també de forma transparent, únicament canviant el que s'anomenen dialectes en el fitxer de configuració de Hibernate. Així, canviar la base de dades de MySQL a PostgreSQL seria teòricament tant senzill com canviar els paràmetres del fitxer de configuració de Hibernate, sense necessitat de tocar codi de l'aplicació.

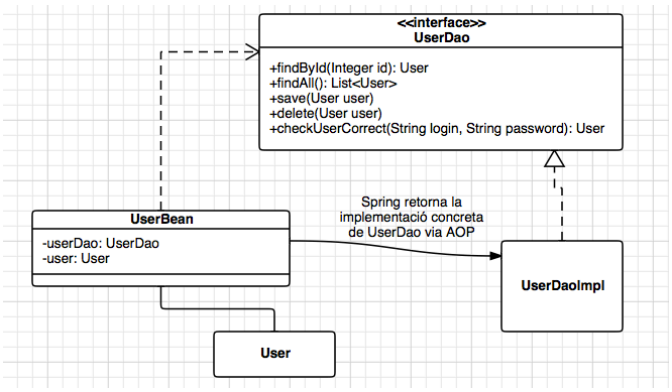
Una de les coses que m'ha permès realitzar aquest TFC ha estat fer recerca sobre la creació d'un DAO genèric que realitzés les operacions CRUD per a totes les entitats. D'aquesta manera evitem repetir codi que seria igual per a tots els diferents DAOs de les entitats persistents de l'aplicació.

La idea és la següent: si tenim 10 entitats, hem de fer 10 DAOs, això no ens ho treu ningú. El que sí que podem fer és que en aquests 10 DAOs no es repeteixin, segons el principi DRY (*Don't Repeat Yourself*), les operacions que són més comunes (les CRUD):

Vegeu el principi DRY a: http://en.wikipedia.org/wiki/Don't_repeat_yourself

- Crear (**C**reate)
- Obtenir (**R**etrieve)
- Actualitzar (**U**pdate)
- Eliminar (**D**elete)

Per comprovar-ho, veiem un diagrama de classes que mostraria el disseny DAO típic per una classe entitat:



Il·lustració Diagrama de classes típic per un DAO

Veiem com hi ha una interfície, `UserDao`, que és implementada per `UserDaoImpl`. Aquesta interfície defineix els mètodes CRUD:

El mètode save realment el que fa és un saveOrUpdate. És a dir, en funció de si l'objecte existeix a la base de dades, el crea o l'actualitza.

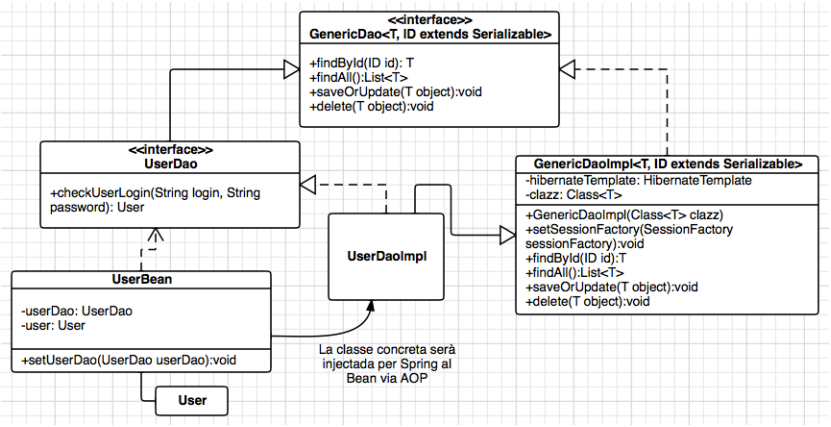
- **C**reate – `save(User)`
- **R**etrieve – `findById(Integer)`, `findAll()`
- **U**pdate – `save(User)`
- **D**elete – `delete(User)`

I un mètode que és específic de la classe `User`:

- `checkUserLogin(String, String)`

Aquest mateix diagrama es repetiria per totes i cada una de les entitats que tinguéssim dintre del nostre domini i que haguessin de ser persistents en base de dades.

Amb l'aproximació de crear un DAO genèric, estalviem feina perquè no repetim codi i, a més, assegurem que el codi serà correcte per totes les classes, ja que si ho és per a una, per inducció ho serà per a la resta. El diagrama de classes amb aquesta solució és el següent:



Il·lustració Diagrama de classes amb un DAO genèric

El diagrama és més complex perquè té un nombre d'elements més gran, però el manteniment d'aquest model és més senzill. Amb aquesta aproximació tenim una interfície `GenericDAO` que, aprofitant les avantatges que ens dóna Java a partir de la seva versió 5, la generalitza com de tipus T (genèrica) i serialitzable. Aquesta interfície és implementada per la classe `GenericDaoImpl` que implementa els mètodes de `GenericDAO`. Aquest és el parell interfície–classe que defineix i implementa les operacions CRUD.

`UserDao` i `UserDaoImpl` continuen existint, però amb aquesta solució només han d'implementar aquells mètodes que tenen a veure amb l'objecte `User`, cosa que redunda en un codi més prim i més senzill de mantenir.

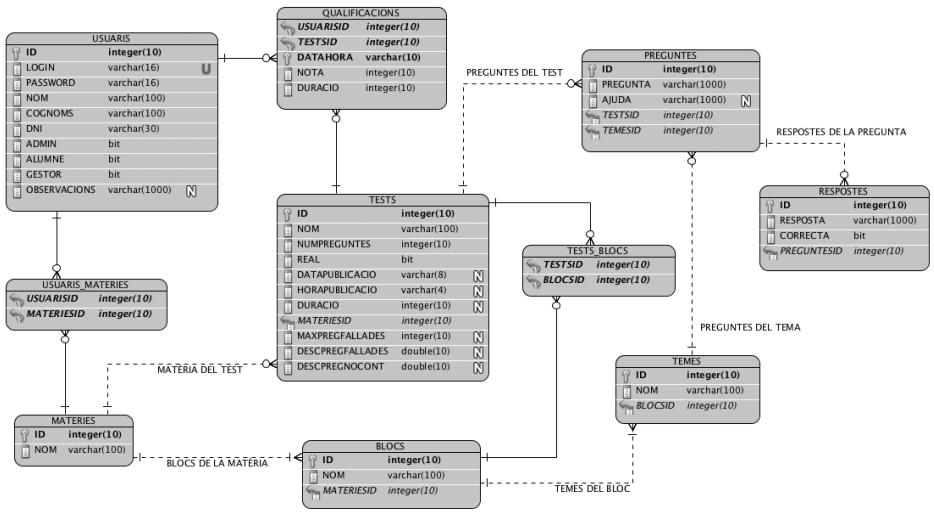
L'arquitectura de la persistència no estaria completa si no esmentéssim l'ús del bastiment Spring 3. Aquest bastiment és força interessant pel que respecta a la persistència. Això és així degut a què ens permet oblidar-nos de la gestió de les transaccions. Mitjançant el fitxer applicationContext.xml podem fer que les transaccions de la base de dades que utilitza Hibernate quan fa les seves operacions, siguin obertes i tancades per Spring, alliberant al desenvolupador de la tasca d'haver de controlar-les.

Spring, al igual que Hibernate i JSF 2, és un bastiment madur i àmpliament usat per la indústria. És per això que està molt provat i resulta aconsellable el seu ús per a aquest tipus d'aplicacions.

És més, l'ús conjunt de Hibernate i Spring pot posar fins i tot en dubte la necessitat de l'ús d'EJBs en aplicacions empresarials. Les dues tecnologies permeten control automàtic de transaccions i escalabilitat horitzontal, pel que semblaria que en força escenaris seria més adequat l'ús de la tupla Hibernate – Spring per la seva senzillesa més que els components EJB.

Diagrama de l'arquitectura de persistència

Aquest és el diagrama de la persistència de l'aplicació:



Il·lustració Model físic de la base de dades

Com es pot veure, les relacions entre les taules definides i les classes identificades en l'anàlisi són les mateixes. Allà on hi havia una relació M<>N s'ha creat una taula entre les dues per poder mantenir la relació entre les entitats.

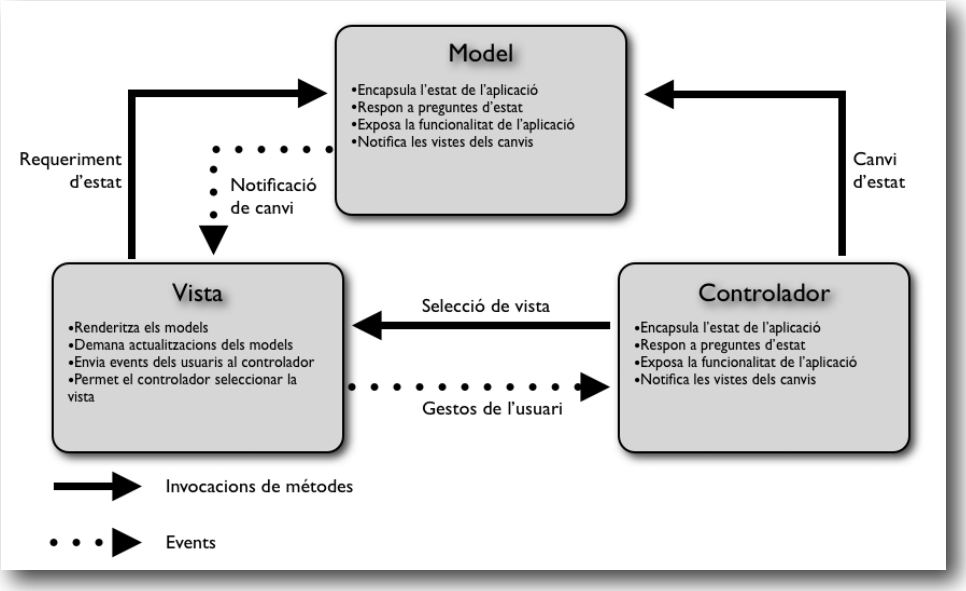
El patró de disseny MVC

Un dels patrons de disseny més emprats per crear aplicacions que disposen d'interfície gràfica és el MVC (acrònim de **M**odel **V**iew **C**ontroller). Aquest patró divideix la responsabilitat de l'aplicació en tres àrees:

- **El model**. És la part de l'aplicació que representa les dades i les normes que defineixen l'accés i les modificacions d'aquestes dades. En la nostra aplicació serien les classes d'entitat i els DAOs.
- **La vista**. Mostra el contingut del model. Defineix com s'han de mostrar les dades. Si el contingut del model es modifica, la vista ho ha de reflectir.
- **El controlador**. Transforma les interaccions de l'usuari a la vista en accions que executarà el model. Aquestes interaccions poden tenir la forma de botons i, en una aplicació web com **CreaTests**, es transformen en peticions `HTTP` de tipus `POST` i `GET`.

Veiem en un diagrama com encaixen aquestes tres peces:

Més informació sobre el patró MVC a: http://en.wikipedia.org/wiki/Mode-view-controller i a: http://www.oracle.com/technetwork/articles/javase/mvc-136693.html Com a curiositat, aquí es pot consultar l'especificació original que es va fer del patró per a l'SmallTalk-80 v2.0: http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html



Il·lustració Diagrama MVC

Les principals avantatges que he trobat amb l'ús d'aquest model són les següents:

- **Separació de conceptes**. La reducció de l'acoblament entre mostrar els elements de la vista, les accions que executa l'usuari i la lògica de negoci fa que l'aplicació sigui amigable al canvi. Per exemple, en el cas que es volgués retirar el suport de Hibernate 3 a l'aplicació, només hauríem de canviar els DAO que usen directament Hibernate i els fitxers de configuració d'Spring. La vista i les classes entitat quedarien intactes.
- **Facilitat de manteniment**. És senzill afegir pantalles per mostrar informació o eliminar pantalles que ja no es fan servir sense que el model es vegi afectat per aquest motiu.

- **Separació d'àmbits de treball**. Amb l'ús d'aquest patró es facilita dintre de l'empresa que el dissenyador pugui treballar en el disseny de la pantalla mentre que el desenvolupador ho faci en la lògica de negoci.

En el context de l'aplicació **CreaTests**, els fitxers XHTML formen el que seria la part de la vista.

La part del model són les classes java entitat i els DAO.

La part controlador són les classes XBean que fan crides a les classes del model i dirigeixen cap a vistes concretes i el fitxer faces-config.xml que indica sota quines circumstàncies hem de navegar d'una pàgina a una altra

MVC i Java Server Faces

Seguint el patró MVC, les capes de vista i de controlador s'han desenvolupat amb el bastiment JSF 2. Aquest bastiment és l'oficial per al desenvolupament d'aplicacions web recomanat per Oracle.

Recomanat en tant en quant és l'esmentat al *The Java EE 6 Tutorial* a la web d'Oracle, concretament la part II, *The Web Tier*: http://download.oracle.com/javaee/6/tutorial/doc/

JSF 2 permet comunicar la vista amb el controlador fent transparent al desenvolupador la gestió de les peticions `POST` o `GET` per navegar entre pàgines. El mecanisme de JSF 2 per poder navegar es basa en l'String que retorna una funció dintre d'un Bean, o del nom de l'enllaç que es clica a la vista. Aquest String pot ser el nom d'una pàgina existent o bé un literal que coincideix amb una regla definida al fitxer faces-config.xml. Per exemple, dintre d'un Bean podem tenir una funció que es crida des de la vista i que retorna una cadena. Aquesta cadena pot ser el nom d'una pàgina, amb el que navegaríem cap a la pàgina indicada, o bé pot pertànyer a una regla definida a faces-config.xml.

El fet de poder cridar una pàgina directament és nou de JSF 2. En les versions anteriors tota la navegació anava en funció de les regles de navegació definides a faces-config.xml.

```
Classe UserBean
...
public String checkLogin() {
  String retorn = null;
  if ( usuariDao.checkLogin(login,password) ) {
    retorn = "FormPrincipal";
  }
  return retorn;
}
...
```

Imaginem que estem a la pantalla de login i que aquest codi comprova si l'accés a l'aplicació per part d'un usuari és vàlid. En cas que sigui així, retornarem la cadena `FormPrincipal`, cosa que farà que ens dirigim al menú principal. Si no, retornarem `null`, la qual cosa provocarà que es repinti la pantalla de login. En aquest exemple hem usat la navegació directa.

Ara veiem el mateix exemple però usant regles de navegació. Necessitarem dos fitxers, la classe que hem vist i el fitxer faces-config.xml.

```
Classe UserBean
...
public String checkLogin() {
  String retorn = “error”;
  if ( usuariDao.checkLogin(login,password) ) {
    retorn = “success”;
  }
  return retorn;
}
...

Fitxer faces-config.xml
...
<navigation-rule>
  <display-name>FormLogin.xhtml</display-name>
  <from-view-id>/FormLogin.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/FormPrincipal.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>error</from-outcome>
    <to-view-id>/FormLogin.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
...
```

Com es pot veure en aquest exemple la classe en comptes de retornar els noms de les pàgines o `null`, retorna `error` o `success` en funció de si el login és correcte o no. El fitxer faces-config.xml té una regla que ens diu que si estem a la pàgina `FormLogin.xhtml` i rebem la cadena `success`, ens hem de dirigir a `FormPrincipal.xhtml`. En canvi, si rebem la cadena `error`, hem de tornar a `FormLogin.xhtml`.

Podéu trobar una referència ràpida dels elements de JSF 2 a la web de l'autor del llibre Core JavaServer Faces Third Edition: http://horstmann.com/corejsf/refcard.html

JSF 2 també atorga al desenvolupador tota una sèrie de controls per poder usar a la vista. Aquests controls són en la seva majoria embolcalls dels components HTML usats tant dintre dels formularis com fora d'ells. Per als que s'empren dintre del formulari dóna la possibilitat d'enllaçar el seu valor amb els valors del model que ens interessi.

Integració de Hibernate 3 i JSF 2 amb Spring 3

Spring és un bastiment que hem utilitzat a **CreaTests** per poder treballar amb més facilitat amb Hibernate i JSF 2.

Spring i Hibernate

Spring permet la configuració de Hibernate des dels seus propis fitxers de configuració. Conté classes que s'integren directament amb aquest bastiment, cosa que permet que el desenvolupador se centri en el codi productiu i deixi les tasques automatitzades (control de transaccions, ja comentat anteriorment en aquest document) al propi bastiment de Spring.

La integració amb Hibernate es fa mitjançant anotacions a les classes que implementen els DAOs i el fitxer de configuració de Spring. A continuació mostro un petit resum de les anotacions emprades i de la configuració del fitxer.

Anotacions de les classes:

Anotació	Significat
`@Transactional`	Anotació situada abans de la definició de la classe. Indica que les operacions que atacaran a la base de dades seran transaccionals. Per defecte aquestes operacions són de només lectura. Per a que les transaccions puguin ser d'escriptura (necessari per eliminar, guardar o modificar a base de dades), a sobre dels métodes que volem que puguin escriure a base de dades hem d'afegir-hi l'atribut `@Transactional(readOnly=false)`. Aquesta anotació va en conjunció amb la configuració del fitxer applicationContext.xml `<tx:annotation-driven />`
`@Autowired`	Els objectes que seran gestionats per Spring es poden declarar de forma explícita al fitxer de configuració, o bé es pot indicar a Spring que busqui les classes que volem que incorpori. L'anotació `@Autowired` el que fa és respondre a l'algorisme exploratori de Spring quan busca classes automàticament. Va en conjunció amb `<context:component-scan base-package="uoc.tfc.creatests" />`.
`@Repository`	Aquesta anotació s'empra en les classes que implementen els DAO definits per l'aplicació. Quan la definim abans de declarar una classe, ho hem de fer passant-li un paràmetre de tipus String, per exemple, `@Repository("usuariDao")`. El que aconseguim amb això és que no haguem de definir explícitament les implementacions dels DAOs al fitxer de configuració. Igual que `@Autowired`, el que fa `@Repository` és respondre a l'algorisme exploratori de Spring quan busca classes automàticament. Va en conjunció amb `<context:component-scan base-package="uoc.tfc.creatests" />`.

Configuració del fitxer applicationContext.xml:

Configuració	**Significat**
`<context:component-scan base-package="uoc.tfc.creatests" />`	Aquest component de configuració indica a Spring que ha de revisar totes les classes que hi ha per sota de `package` a la recerca de classes amb anotacions `@Autowire` i `@Repository`.

Configuració	Significat
`<tx:annotation-driven />`	Indica a Spring que els accessos que es facin a base de dades seran transaccionals. A partir d'aquí Spring s'encarrega de la gestió d'aquestes transaccions.
`<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">`	Defineix una font de dades (un accés contra una base de dades). Propietats típiques d'una font de dades són el host, el nom de la base de dades, l'usuari i el password.
`<bean id="sessionFactory" class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">`	Aquesta configuració defineix una sessió que serà usada per Hibernate a l'hora de realitzar els accessos a la base de dades definida com a font de dades.
`<bean id="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">`	Indica a Spring que ha d'usar la sessió definida per controlar les transaccions que hi hagi a l'aplicació.

Spring i JSF 2

Spring permet que els beans creats amb JSF 2 tinguin cicles de vida determinats dintre de l'aplicació web. Per defecte, Spring admet tres cicles de vida (tot i que el desenvolupador pot crear els seus propis):

- **Singleton**. El bean existeix al llarg de la vida de l'aplicació i per a tots els usuaris. Un ús típic podria ser el guardar una llista de països en memòria per no haver de consultar cada cop la base de dades.
- **Session**. El bean existeix al llarg d'aquella sessió per a aquell usuari. Quan l'usuari tanca la sessió o el contenidor web la tanca per timeout, el bean deixa d'existir. Un ús típic és per guardar l'usuari que ha entrat i poder accedir als seus atributs sense haver de consultar-los cada cop a la base de dades.
- **Request**. La vida del bean dura el que triga en executar-se una petició web. S'empra per passar paràmetres a cercadors, per exemple, o filtres sobre resultats obtinguts.

Podéu trobar més informació sobre el patró Dependency injection a: http://en.wikipedia.org/wiki/Dependency_injection

Els beans de JSF 2 són els que tenen associats els DAO i els que finalment criden als mètodes que accedeixen a dades. Aleshores, se'ns planteja la qüestió de com podem associar una implementació concreta d'un DAO a un bean. La resposta ens la dóna Spring, ja que gràcies al patró *Dependency injection* podem associar implementacions concretes dels DAO als beans. Veiem un exemple de com enllaçar-ho:

```
Classe UsuariDaoImpl
...
@Repository(“usuariDao”)
public class UsuariDaoImpl extends...
...

Classe MateriaDaoImpl
...
@Repository(“materiaDao”)
public class MateriaDaoImpl extends...
...

Classe UsuariBean
...
private UsuariDao usuariDao;
private MateriaDao materiaDao;

public void setUsuariDao(UsuariDao usuariDao) {
  this.usuariDao = usuariDao;
}

public void setMateriaDao(MateriaDao materiaDao) {
  this. materiaDao = materiaDao;
}
...

Fitxer applicationContext.xml
...
<bean id="usuari"
    class="uoc.tfc.creatests.view.UsuariBean"
    scope="session">
  <property name="usuariDao">
    <ref bean="usuariDao"/>
  </property>
  <property name="materiaDao">
    <ref bean="materiaDao"/>
  </property>
</bean>
...
```

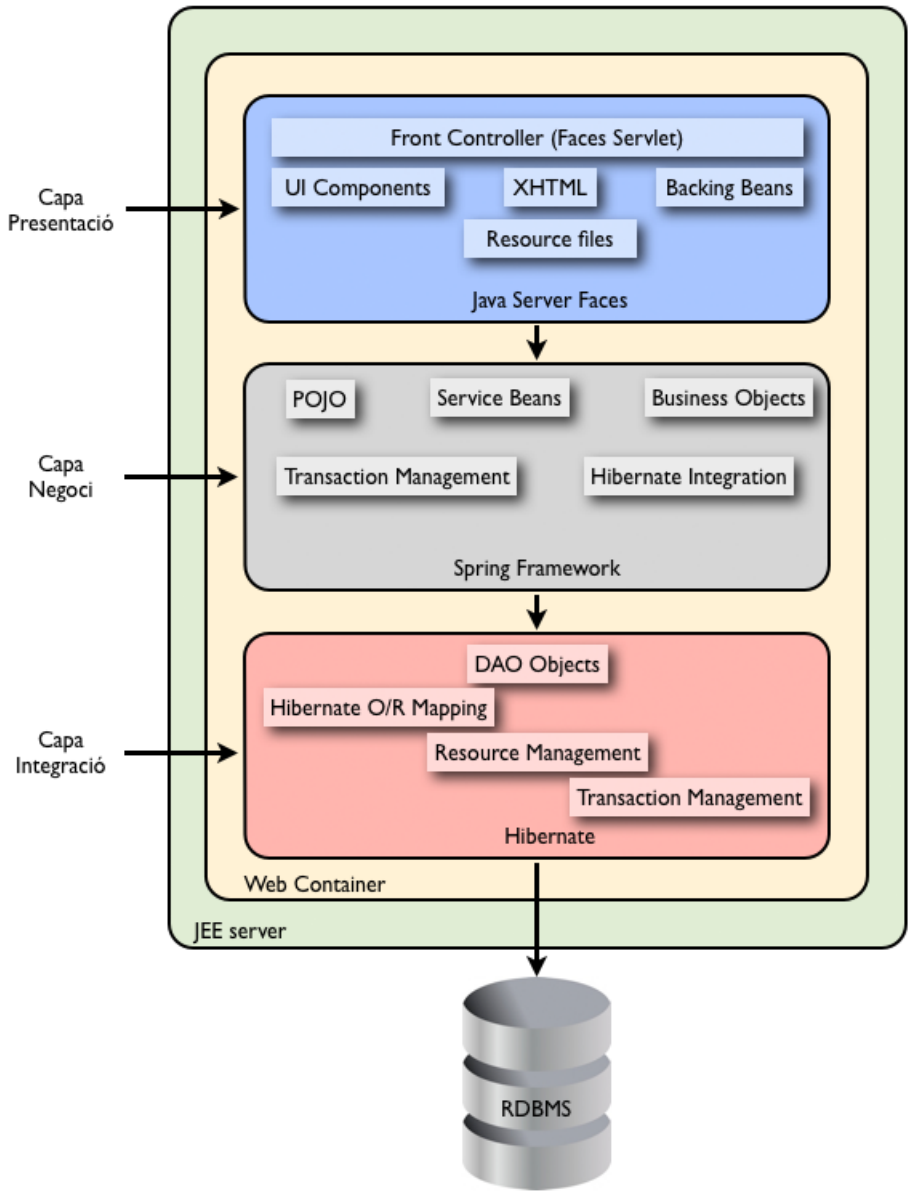
El que hem fet en aquest exemple és veure com, declarant que `UsuariBean` té per atributs els DAOs d’usuari i de matèria i que tenen mètodes d'accés per definir-los, des del fitxer de configuració d’Spring enllacem mitjançant *Dependency injection* les classes concretes que implementen els DAOs.

La integració de tot plegat

Com a punt final d'aquesta secció, mostro un esquema de com interactuen les diferents capes de l'aplicació i quins són els components que s'han utilitzat per dur-la a terme.

Aquests són els components que s'han emprat:

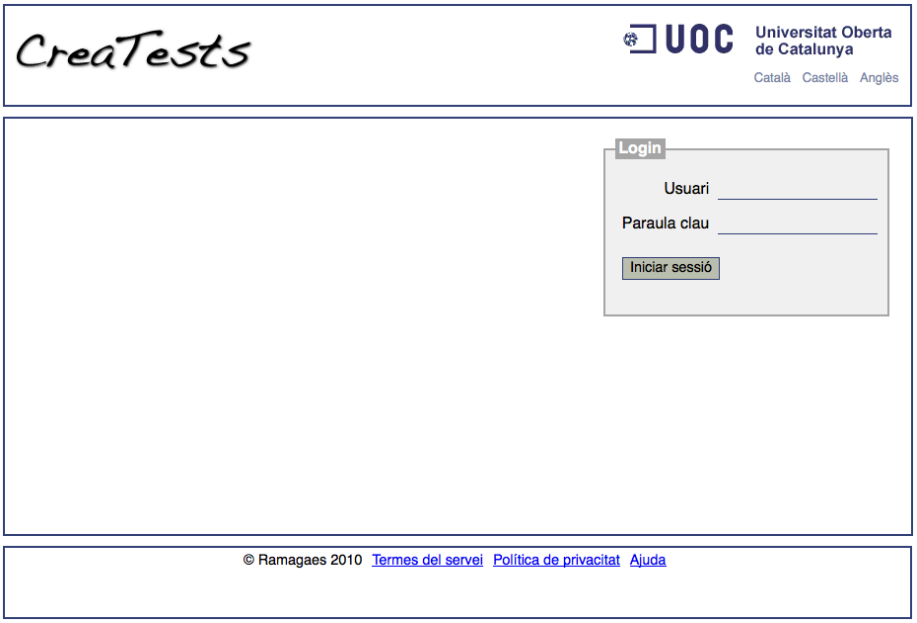
- **Motor SGBD**: MySQL 5.1
- **Contenidor de servlets**: Tomcat 6
- **Capa de presentació**: JSF 2.0
- **Capa de negoci**: Spring 3
- **Capa d'integració**: Hibernate 3.3



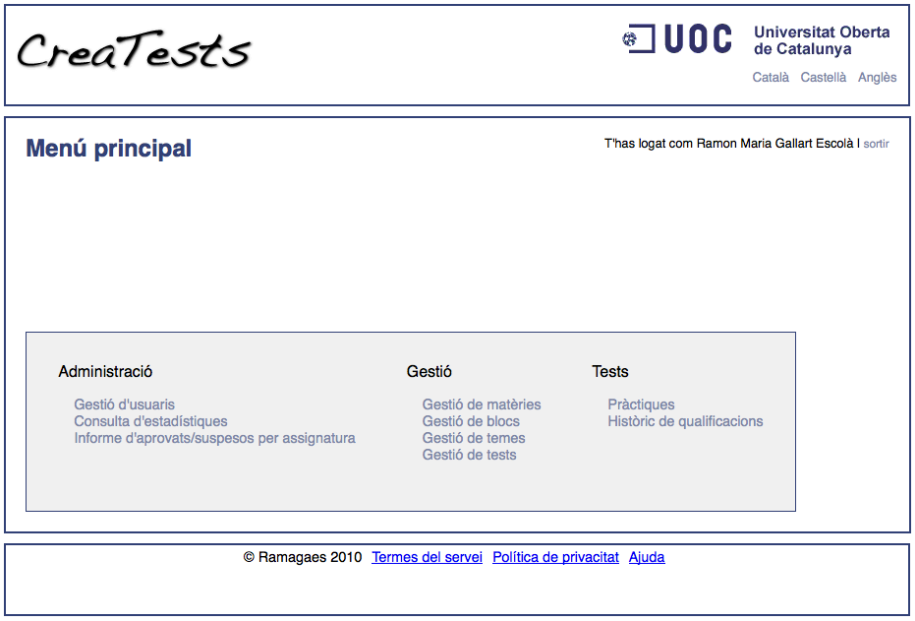
Interfície d'usuari

A l'hora de crear la interfície d'usuari s'ha perseguit l'objectiu que fos neta i accessible per sobre d'un disseny gràfic impactant. El referent han estat bàsicament les planes de Google, que amb un disseny minimalista aconsegueixen fer tot el que la plana ha de fer.

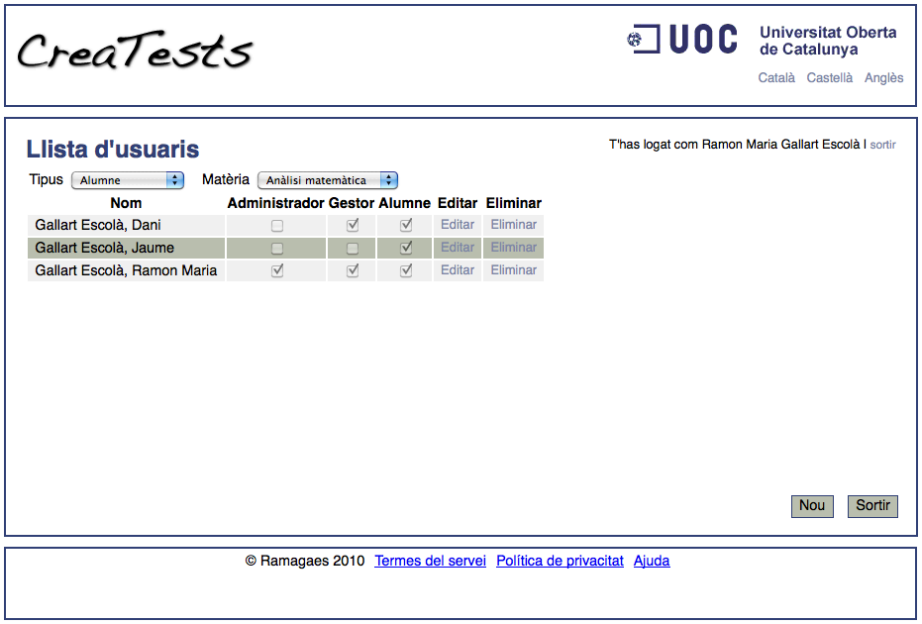
Aquí hi ha una mostra de les planes de l'aplicació:



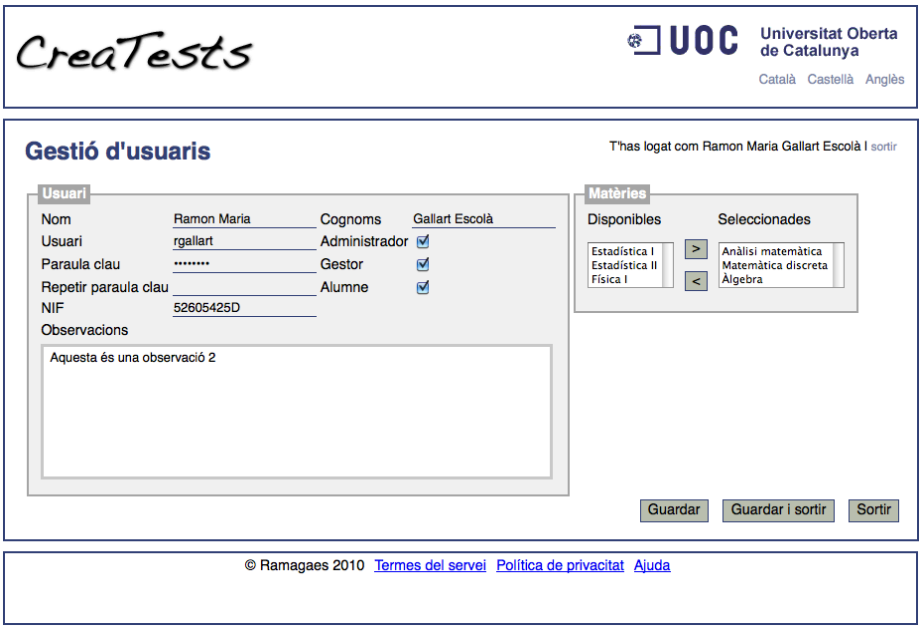
Il·lustració Pantalla d'entrada a l'aplicació



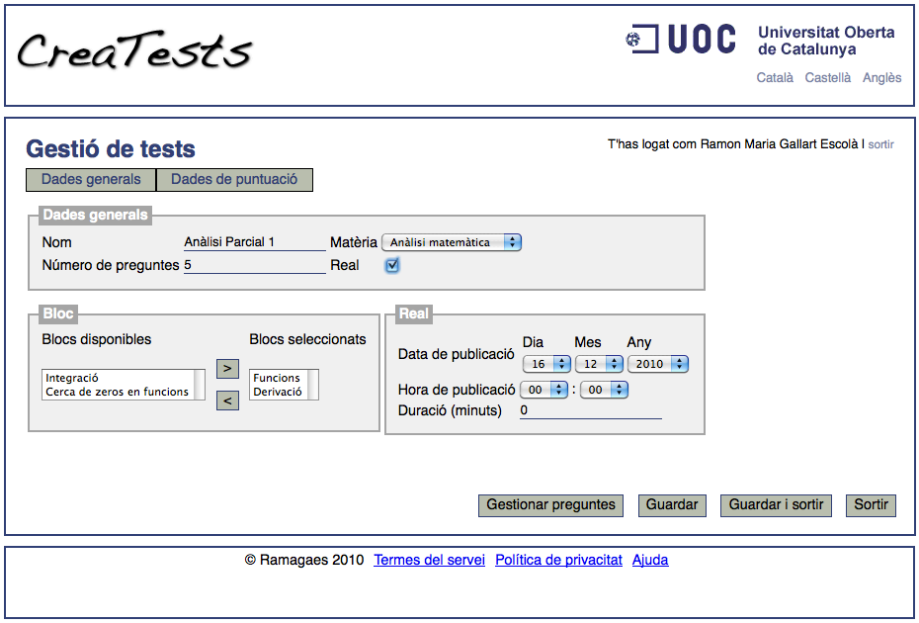
Il·lustració Menú principal



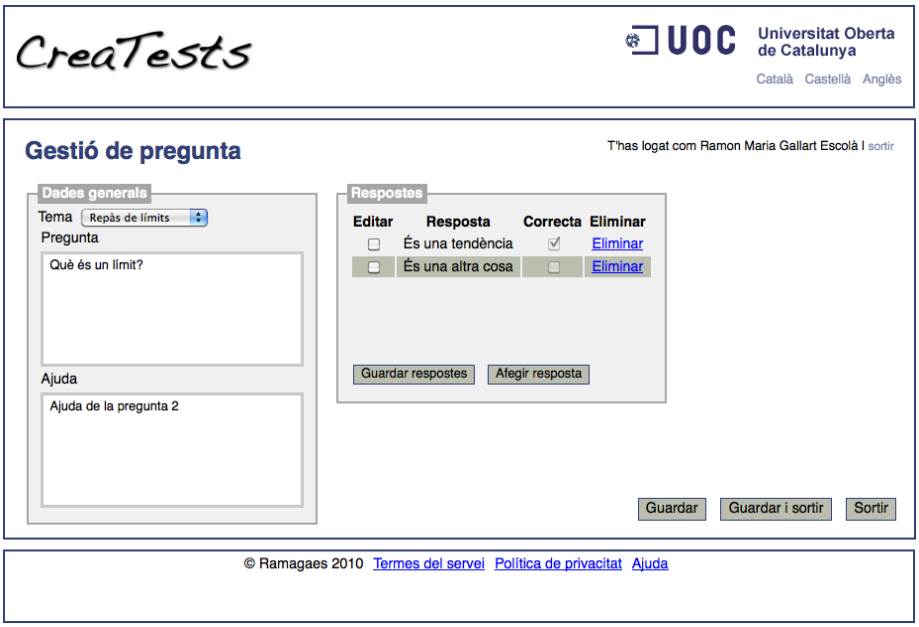
Il·lustració Llista d'usuaris



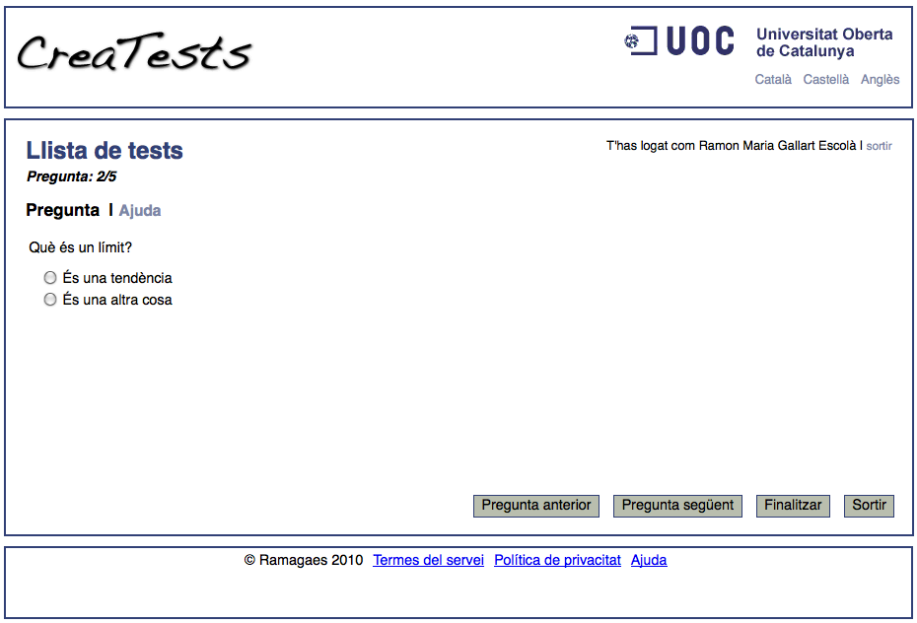
Il·lustració Gestió d'usuaris



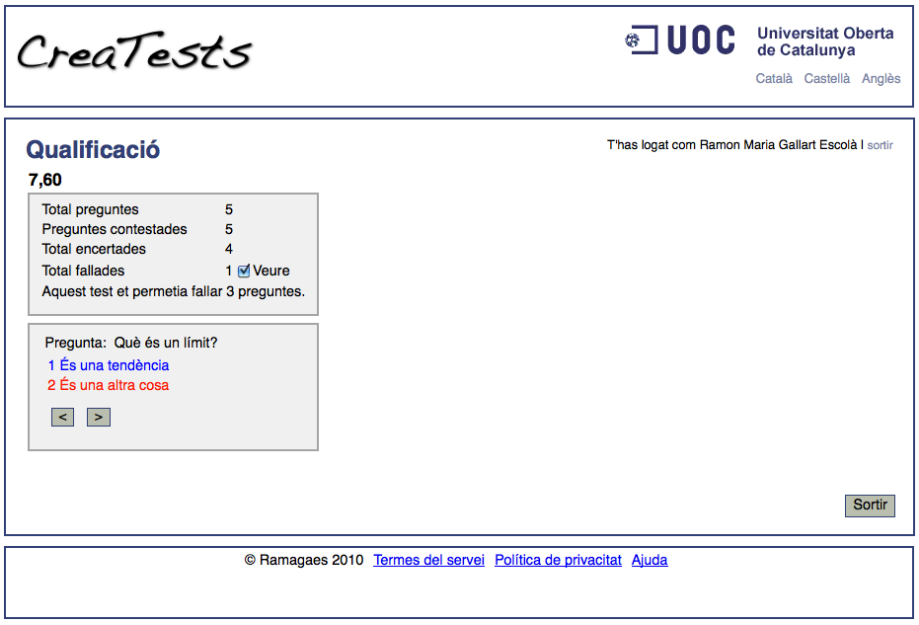
Il·lustració Gestió de tests



Il·lustració Gestió de pregunta



Il·lustració Execució de tests



Il·lustració Qualificació del test

Valoració econòmica

Per fer la valoració econòmica del projecte em basaré en les següents premisses:

1. Hi haurà un cap de projectes que dedicarà el 20% de les hores totals del projecte a la seva gestió. El cap de projectes cobra 60 €/hora.
2. Hi haurà un analista que s'encarregarà de les fases de recollida de requisits, anàlisi i disseny. L'analista cobra 30 €/hora.
3. Finalment hi ha el programador. S'encarrega de la fase de desenvolupament. Cobra 20 €/hora.
4. Hi ha un marge del 30% per a l'empresa que ja està contemplat en el preu del cap de projectes, l'analista i el programador. Amb aquest marge es pot jugar en cas que el projecte s'endarrereixi o hi sorgeixi algun contratemps.
5. La funcionalitat està a un 80% i es calcularà d'aquesta manera. La implementació de la resta de funcionalitats comportaria tornar a calcular la valoració.
6. No es contempla la venda del servidor físic ni de les màquines client a l'usuari final de l'aplicació.
7. No es contempla un altre model d'explotació de l'aplicació més que el de la venda del mateix com un paquet.

Projecte CreaTests

Fases	Rol	Hores	Preu/hora	Preu Final
Requisits	Analista	72	35 €	*2.520 €*
Anàlisi	Analista	88	35 €	*3.080 €*
Disseny	Analista	96	35 €	*3.360 €*
Programació	Programador	192	20 €	*3.840 €*
Gestió	Cap de projectes	90	60 €	*5.376 €*
			Subtotal	**18.176 €**
			IVA	18%
			Total	**21.447,68 €**

Amb aquestes premisses, tenim que hauríem de vendre el producte per 21.447,58 € per tenir el màxim benefici. Si hi ha problemes al llarg del projecte, es pot arribar a vendre'l per 16.500 € –tenint en compte el marge del 30% esmentat en el punt 4. Per sota d'aquest preu, la venda del producte ja és deficitària i comporta pèrdues.

Conclusions

Un Treball de Final de Carrera es fa, entre d'altres coses, per aprofundir en algun tema d'interès particular. En aquest sentit, el TFC ha estat molt gratificant. Com comentava a la introducció de la memòria, personalment tinc molta experiència amb el llenguatge Java i amb la tecnologia web. Tot i això, no havia pogut treballar o investigar amb suficient profunditat certs bastiments que m'interessaven a nivell particular.

Gràcies al TFC ha estat possible aquest aprofundiment i he aprés molt sobre ells en particular, i algunes coses noves sobre la tecnologia web en general.

Com a simulació que ha estat de la creació d'un producte seguint les fases de requeriments, anàlisi, disseny i desenvolupament i proves, també ha estat força engrescador.

Finalment comentar que ha estat una experiència molt positiva. Crec que, a banda de les fases esmentades anteriorment, he passat també per totes les fases d'ànims possibles. Des de l'eufòria de poder fer el TFC de J2EE que era la que volia, passant per la fase 'Déu meu, això no ho acabaré' i la fantàstica sensació de veure que, amb esforç i persistint en la tasca, finalment acaba sortint. Amb raó aquesta assignatura és la mare de totes les pràctiques que hagi pogut fer a la UOC.

Bibliografia

Per poder realitzar aquest TFC s'ha consultat la bibliografia següent:

- Geary, David i Horstmann, Cay (2010). Core JavaServer Faces (3rd ed.). Redwood Shores, CA: Oracle. ISBN-13: 978-0-13-701289-6. http://horstmann.com/corejsf/
- Horstmann, Cay i Cornell, Gary (2008). Core Java, vol. I - Fundamentals (8th. ed.). Santa Clara, CA: Sun Microsystems, Inc. ISBN-13: 978-0-13-235476-9. http://horstmann.com/corejava.
- Horstmann, Cay i Cornell, Gary (2008). Core Java, vol. II - Advanced Features (8th. ed.). Santa Clara, CA: Sun Microsystems, Inc. ISBN-13: 978-0-13-235479-0. http://horstmann.com/corejava.
- Stelting, Stephen i Maasen, Olav (2003). Patrones de diseño aplicados a Java. Madrid: PEARSON EDUCACION. ISBN: 84-205-3839-6. http://bit.ly/hOeoOP
- Jacobson, I. i Booch, G. i Rumbaugh, J. (2000). El Proceso Unificado de Desarrollo de Software. Madrid: PEARSON EDUCACION. ISBN: 84-7829-036-2.
- King, Gavin i Bauer, Christian i Rydhal Andersen, Max i Bernard, Emmanuel i Ebersole, Steve (2010). Hibernate Reference Documentation. http://bit.ly/dVeeCo.
- Sebastian Hennebrueder (2009). Hibernate 3 by Example. Germany: http://www.lalilluna.de
- Johnson, Rod i Co. (2004-2010). Spring Framework. Reference Documentation. 3.0: http://bit.ly/855FFG
- Campderrich Falgueras, Benet i Recerca Informàtica, SL (2004). Enginyeria del Programari. Barcelona: UOC. ISBN: 84-9788-065-X
- Marco Galindo, Ma. Jesús i Anton i Solà, Ma. Dolors (2004). Enginyeria del Programari Estructurada. Barcelona: UOC. ISBN: 84-9788-066-8.
- Barceló García, Miquel i Pastor i Collado, Joan Antoni (2004). Gestió d'organitzacions i projectes informàtics. Barcelona: UOC. ISBN: 84-9788-063-3.