Anàlisi, disseny i implementació d'un sistema de gestió d'incidències i sol·licituds informàtiques en J2EE

Albert Puigvert Bosch
ETIG

Salvador Campo Mazarico

14 de gener de 2009

A la meva dona i als meus tres fills

Per la seva complicitat i paciència i per totes les hores que no els he pogut dedicar en els últims cinc anys

Al meu pare

Es sentiria orgullòs de ternir un fill Enginyer.

Agraïments:

A tota la comunitat de la UOC, gracies per haver fet possible aquest somni.

Resum del Treball Final Carrera.

Aquest treball respon al repte de construir, des de zero i sense cap coneixement previ de la tecnologia J2EE, una aplicació professional i d'àmbit empresarial que té com a finalitat la gestió d'incidències i sol·licituds informàtiques referents a l'àmbit tecnològic, per part d'un departament de Tecnologies de la Informació i de la Comunicació.

El projecte està basat totalment en el món real i pretén, per tant, donar solució a una bona part de les necessitats existents en els departaments tecnològics que donen suport als usuaris finals.

La combinació dels coneixements adquirits de manera acadèmica en aquest últims cinc anys d'estudi, amb la necessitat de satisfer uns requeriments 100% objectius, fan que aquest projecte sigui la millor manera de sincronitzar el mapa conceptual teòric amb la tecnologia existent i les necessitats empresarials reals.

El projecte ha girat sobre els pilars de l'Enginyeria de Programari Orientat a Objectes i seguint i ampliant les Tècniques de Desenvolupament apreses a la UOC. En aquest sentit s'han establert de manera força exhaustiva les etapes de anàlisi, disseny i implementació així com s'han utilitzat de manera justificada molts dels patrons de desenvolupament que podem considerar estàndards.

L'elecció de la plataforma J2EE com a base del desenvolupament del projecte ha estat motivada simplement per les ganes de conèixer-la i aprendre-la. En aquest sentit s'ha realitzat un esforç extramadament gran, dins d'aquest desconeixement previ, per escollir de manera justificada tots els elements idonis per portar a terme la implementació del projecte, tenint en compte la la gran diversistat tant d'APIs com de diferents implementacions que ofereix l'estandard J2EE.

També s'ha tingut una cura molt especial en totes les decissions arquitectòniques per tal de fer prevaldre els principis d'alta coherència i baix acoblament, proporcionant d'questa manera un alt nivell de reutilització de cadascuna de les seves parts i obtenint un producte final totalment consistent, robust, flexible i escalable.

1. Índex.

2. Índex de Figures

3. INTRODUCCIÓ.

3.1 Justificació del TFC i context: punt de partida i aportació.

Durant més de 20 anys la meva vida laboral ha estat girant en l'entorn del desenvolupament de programari, inicialment utilitzant programació procedimental i en els últims quatre anys utilitzant programació orientada a objectes. Durant aquest temps he viscut al marge de les aplicacions Web, és a dir, he desenvolupat única i exclusivament aplicacions d'escriptori, de la mateixa manera que he viscut també a esquenes del mon Java.

Crec que, avui en dia, una persona que es posi el títol de desenvolupador ha de tenir uns certs coneixements de la plataforma java, i concretament de J2EE i de la mateixa manera ha de tenir un cert domini de tots els conceptes que fan del món Web un mon tant especial.

Per això, aquest TFC ha estat per mi la oportunitat clau. Ha estat la manera de posar-me al dia, d'enfrontar-me a les meves mancances i d'assolir molts dels requeriments del mercat laboral. Ha estat sens dubte el repte més gran des de que vaig començar la carrera.

L'elecció d'una aplicació real, plantejada en el meu propi entorn de treball, li dona al projecte un caire més professional, un objectiu menys ambigu i sobre tot uns ànims addicionals a l'hora de treballar en el projecte pel fet de pensar que aquest programari possiblement serà utilitzat per usuaris de veritat.

Des que vaig començar a estudiar he intentat sempre donar un valor afegit a tot el que faig, i aquest cas no serà una excepció. Per una banda, des del començament del projecte, he ampliat els meus coneixements d'enginyeria realitzant varies lectures prèvies a la presa de decisions arquitectòniques, concretament he llegit "Patterns of Enterprise Arquitechture" de Martin Fowler, "POJOs in Action" i "Java Server Faces in Action" de l'editorial Manning, així com bona part dels manuals de referència de Spring, Struts i Hibernate, i , per una altra banda he estat investigant mitjançant articles i projectes d'Internet quines eren les possibilitats a l'hora d'utilitzar APIs, implementacions i Frameworks de treball.

Per tant, crec que la meva aportació vindrà donada per unes decisions arquitectòniques molt sòlides i per una combinació d'elements en el projecte del tot justificades. M'agradaria remarcar en aquest aspecte, la decisió d'utilitzar la tecnologia JSF i Facelets en la part de presentació, que podríem titllar de jove, o fins i tot de verda, que ens aportarà elements visuals i de funcionalitat operativa que ens direccionen cap al que s'anomena **RIA** (Rich Internet Applications) i que milloren en molt l'experiència d'usuari.

També m'ha semblat que una aportació interessant podria ser la de integrar tant el **Google Maps** com un element de Business Intelligence, concretament un element d'anàlisi **OLAP** (On Line Analitycal Processing) dins del projecte, en un moment que es parla molt d'aquesta tecnologia, però que ningú incorpora de manera integrada en les seves aplicacions.

3.2 Objectius del TFC.

Com es desprèn dels darrers punts, l'objectiu d'aquest TFC és doble:

- L'objectiu més important és totalment de caire personal i el podem descriure com "L'assoliment dels coneixements necessaris per portar a terme projectes en J2EE" .

- Desenvolupar una eina basada en Web de Gestió d'Incidències i Sol·licituds Informàtiques que compleixi els principis de coherència, consistència, robustesa i usabilitat i que permeti als usuaris finals dels diferents departaments de l'empresa adreçar totes les incidències i peticions referents a l'àmbit tecnològic al departament de les TIC, dotant a aquest últim dels elements necessaris per a la seva gestió, seguiment i resolució.

3.3 Enfocament i mètode seguit.

El projecte es portarà a terme seguint el **cicle de vida clàssic** del programari, també anomenat cicle de vida en cascada. En cadascuna de les etapes d'aquest mètode obtindrem uns documents que serviran de base per l'etapa següent.

L'elecció dels tipus de document es farà en cadascuna de les etapes segons es cregui convenient amb la premissa d'utilitzar sempre la riquesa dels diagrames de l'estendard UML.

El projecte que ens ocupa comporta una dificultat addicional sobre la majoria de projectes i resideix en el fet de que ens enfrontem a un tecnologia molt complexa i àmplia amb un nivell de coneixements que s'acosta a zero.

El plantejament consistirà en combinar la fase d'anàlisi i la fase de disseny amb l'aprenentatge de J2EE, per tal de poder arribar a la fase de programació amb uns coneixements suficients por portar-la a terme.

De fet, com es veure més endavant en la planificació, l'aprenentatge de J2EE és una línia constant que abasta tot el projecte.

Remarcar, que, un dels secrets del projecte residirà en la solidesa de l'anàlisi i el disseny i sobre tot en el establiment d'un domini de funcionalitats inicials que no pugui fer perillar la fase de desenvolupament.

3.4 Planificació del projecte.

La planificació de cadascuna de les fases ha estat condicionada a les dates dels diferents lliuraments assenyalades en el Pla Docent de l'assignatura. D'aquesta manera cadascuna de les fites del projecte es correspon amb una data de lliurament:

- **01/10/2008 – Lliurament PAC1**
  Confecció i lliurament del Pla de Treball amb la descripció del projecte, objectius i planificació.

- **05/11/2008 – Lliurament PAC2**
  Lliurament dels documents d'anàlisi i disseny del projecte. El primer Inclou l'anàlisi de requeriments funcionals i no funcionals, diagrama de casos d'us i diagrama d'estats, el segon la definició de l'arquitectura del sistema, prototips de pantalla, diagrama estàtic de classes, E/R de persistència i diagrames de col·laboració.

- **17/12/2008 – Lliurament PAC3**
  Aquest lliurament està catalogat com a control de seguiment i es situa en la part avançada de la fase d'implementació.

- **14/01/2009 – Lliurament final**
  Lliurament de la Memòria del projecte.
  Lliurament del programari i manual d'instal·lació.
  Lliurament de la presentació del projecte.

Seguidament acompanyem un diagrama de Gantt amb les tasques principals i la seva descomposició en sub tasques:

Diagrana de Gantt

Nombre de tarea	Duración	Comienzo	Fin
Anàlisi previ	**9 días**	**22/09/08**	**01/10/08**
Justificació del projecte	5 días	22/09/08	26/09/08
Definició d'objectius	5 días	22/09/08	26/09/08
Reunió consultor	1 día	27/09/08	27/09/08
Planificació	3 días	29/09/08	01/10/08
Elaboració document Pla de Treba	3 días	29/09/08	01/10/08
Lliurament PAC1	1 día	01/10/08	01/10/08
Anàlisi i especificació del sistem	**10 días**	**02/10/08**	**15/10/08**
Requeriments funcionals	2 días	02/10/08	03/10/08
Requeriments no funcionals	1 día	06/10/08	06/10/08
Definició d'actors i casos d'us	2 días	07/10/08	08/10/08
Definició de subsistemes i paquets	1 día	09/10/08	09/10/08
Diagrama d'estats	1 día	10/10/08	10/10/08
Elaboració document anàlisi i gloss	3 días	13/10/08	15/10/08
Disseny	**15 días**	**16/10/08**	**05/11/08**
Arquitectura del sistema	1 día	16/10/08	16/10/08
Interfície d'usuari	3 días	17/10/08	21/10/08
Diagrama de classes	3 días	22/10/08	24/10/08
Diagrames de seqüència	3 días	27/10/08	29/10/08
Diagrama de persistència	3 días	30/10/08	03/11/08
Diagrama E/R	2 días	04/11/08	05/11/08
Lliurament PAC2	1 día	05/11/08	05/11/08
Implementació	**40 días**	**06/11/08**	**31/12/08**
Desenvolupament Business logic	10 días	06/11/08	19/11/08
Desenvolupament Persistència	6 días	20/11/08	27/11/08
Desenvolupament Interfície gràfica	24 días	28/11/08	31/12/08
Lliurament PAC3	1 día	17/12/08	17/12/08
Testing i control de qualitat	3 días	01/01/09	05/01/09
Documentació	7 días	06/01/09	14/01/09
Lliurament Final Projecte	1 día	14/01/09	14/01/09
Aprenentatge J2EE	87 días	17/09/08	14/01/09

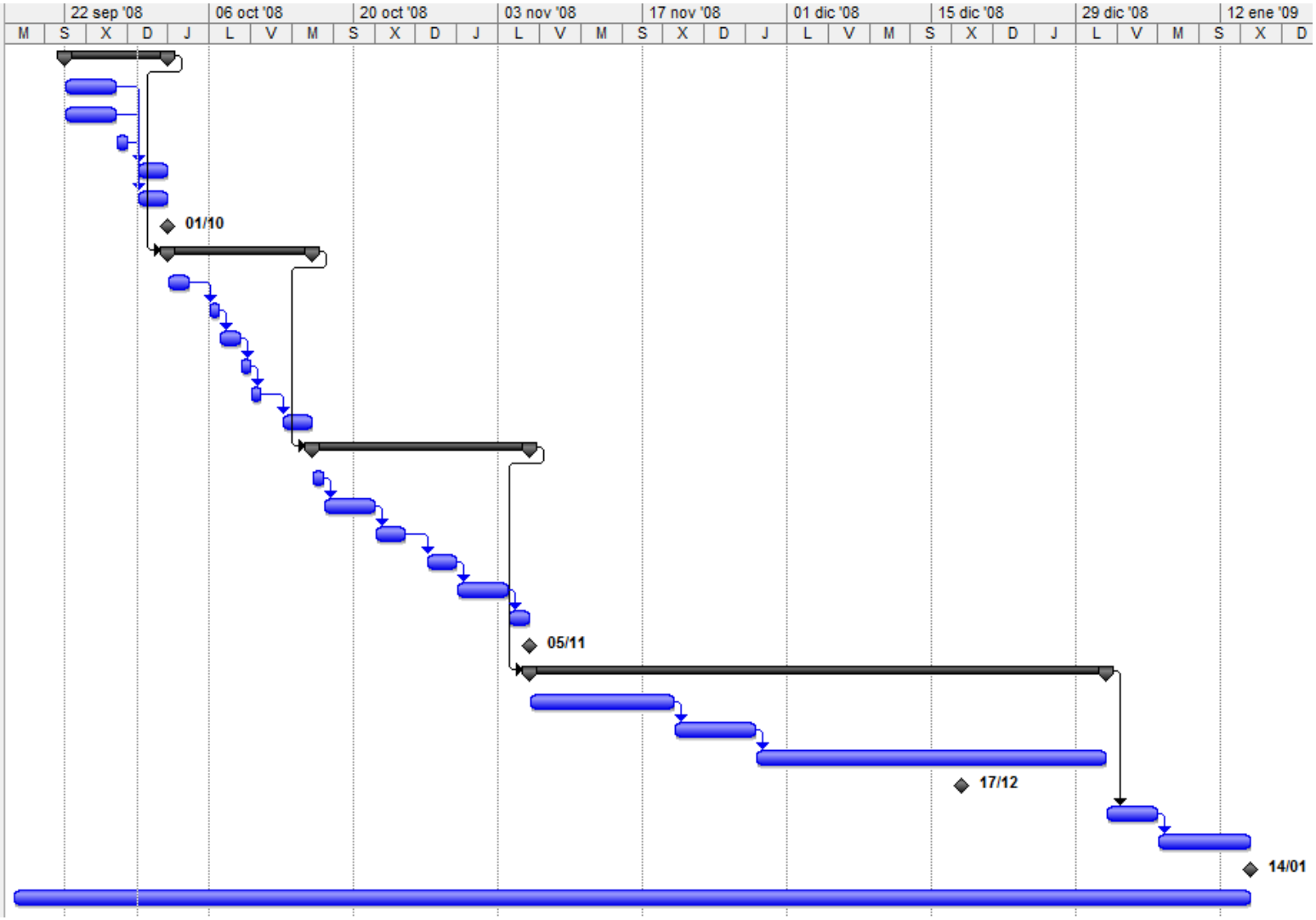


Figura 1. Diagrama de Gantt

3.5 Productes obtinguts.

Com a producte final tangible hem obtingut una aplicació Web de Gestió d'Incidències i Sol·licituds informàtiques desenvolupada en J2EE i, per tant, multiplataforma tant en la part de servidor com a la part de client. Aquesta aplicació és lightweight (poc pesada) pel fet de que no necessita d'un servidor d'aplicacions sinó que tant sols utilitza un simple servidor Web com pot ser l'Apache Tomcat.

Aquest producte està compost per:

- Un fitxer .war de desplegament que conté totes les classes bytecode de java, els recursos web i les llibreries necessàries per realitzar la instal·lació en un servidor web Apache Tomcat.
- El codi font de totes les classes implementades en el projecte.
- Els fitxers de sentencies SQL per a la creació tant de la base de dades, com per la inserció de les dades necessàries per realitzar proves.
- Documentació javadoc de les classes implementades.
- Manual d'instal·lació.

Com a producte intangible i potser més important hem obtingut un coneixement de la tecnologia que va més enllà de les previsions inicials

El producte obtingut en la implementació d'aquest projecte ha estat una aplicació web realitzada amb tecnologia J2EE.

.

3.6 Descripció de la resta de capítols de la memòria.

D'acord amb les fases d'Enginyeria de Programari seguides en el projecte trobarem la següent descomposició en capítols:

- Especificació i Anàlisi de Requeriments amb una introducció a les necessitats, identificació d'actors i casos d'us i un diagrama d'estats
- Disseny amb els diagrames estàtics de classes, diagrama E/R de la base de dades, diagrama de paquets, diagrames de seqüència i totes les decisions d'arquitectura presses amb la seva corresponent justificació.
- Implementació amb el resum de programari utilitzat en el desenvolupament.
- Valoració econòmica de tot el projecte.
- Conclusions finals sobre el TFC.

4. ESPECIFICACIÓ I ANÀLISI DELS REQUERIMENTS

4.1 Introducció.

Aquest capítol recull tots els documents formals necessaris per poder realitzar de manera consistent la següent fase del projecte, la de disseny.

Com es natural, tots els diagrames estaran en notació UML i serà la nostra intenció que aquests siguin totalment intel·ligibles tant per part del client que ens ha encarregat el projecte com pels enginyers que es responsabilitzaran de la següent fase.

Començarem amb una primera aproximació al sistema.

4.2 Descripció del projecte.

L'evolució constant de la tecnologia i l' impacte positiu que ha tingut envers a la gestió i organització de les empreses, ha fet que el nombre d'elements i la complexitat en les instal·lacions informàtiques, tant de programari, com de maquinari, presents dins les companyies hagi crescut de manera quasi exponencial. Com a conseqüència d'això s'ha produït també un augment tant de les incidències provocades pels elements tecnològics com de les provocades pel seu us incorrecte, afectant ambdues i de manera directa al departament de TIC.

Paral·lelament i degut a que la informàtica s'ha convertit en la pedra angular d'algunes organitzacions, s'ha produït un creixement, per no dir un allau, de les sol·licituds o peticions de canvi o millora que els diferents usuaris traslladen al departament de tecnologies de la informació.

És d'aquesta manera com, els departaments de TIC reben un impacte molt dur en el seu dia a dia i es veuen obligats a establir i organitzar un servei que pugui atendre, canalitzar i donar resposta a totes les incidències i peticions rebudes, i és en aquest sentit on el nostre projecte, de la manera més simple possible, intentarà proveir d'una solució tecnològica que satisfaci a tots els actors involucrats.

L'objectiu principal del projecte és, doncs, el de desenvolupar una eina de Gestió d'Incidències i Sol·licituds Informàtiques que permeti als usuaris finals dels diferents departaments de l'empresa adreçar totes les incidències i peticions referents a l'àmbit tecnològic al departament de les TIC, dotant a aquest últim i als tècnics que el composen dels elements necessaris per a la seva gestió, seguiment i resolució.

4.3 Definició d'actors

En el cicle de vida del que seria una incidència o una sol·licitud informàtica intervenen bàsicament dues parts, la primera que és la part peticionaria o consumidora del servei i que es correspon en el mon real a qualsevol empresa usuària de serveis informàtics , i la

segona que és la que proveeix dels elements necessaris per a portar a terme la solució més adient i que es correspon amb una empresa d'outsourcing tecnològic o be amb un departament TIC intern de la pròpia empresa.

Basant-nos en les funcions realitzades en cadascuna de les dues bandes poden identificar de manera clara els següents actors:

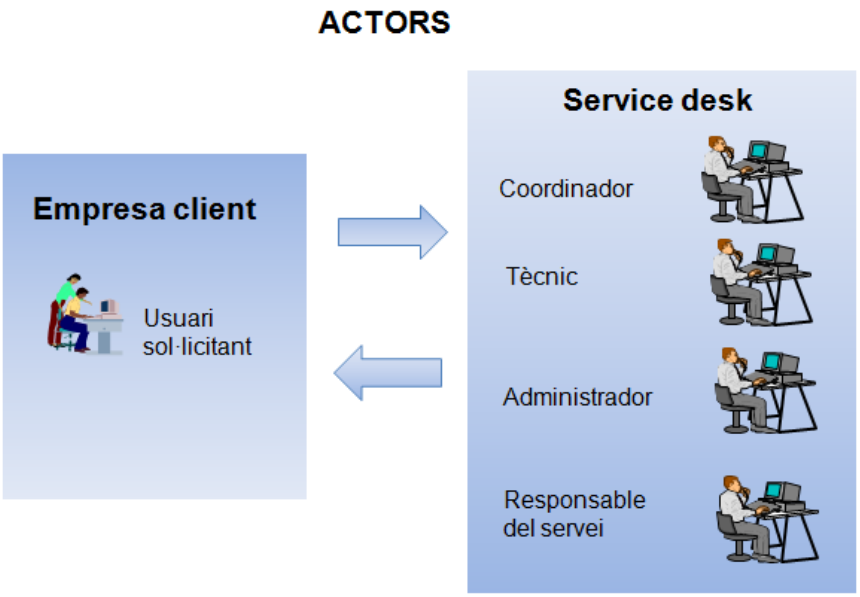


Figura 2. Identificació d'Actors

- **Usuari sol·licitant**: És l'únic actor que pertany a la part consumidora del servei i bàsicament es pot identificar com a la persona que, a partir d'una necessitat informàtica, cursa una petició al Service desk.

- **Coordinador**: Es responsabilitza de l'enregistrament i classificació de les incidències i peticions rebudes en el Service desk i de l'assignació del tècnic responsable. Realitza també l'estimació de la data prevista de finalització.

- **Tècnic**: És aquella persona que realitzarà totes les tasques necessàries per aconseguir resoldre la incidència i en deixa constància. Realitza una valoració del temps emprat.

- **Administrador**: La seva tasca principal és la de proveir de tots els elements necessaris per què el programari de Service Desk s'executi de la manera prevista.

- **Responsable del servei**: Des del punt de vista d'aquest projecte, és la persona que disposarà d'un sistema d'anàlisi de dades per ajudar-lo a prendre decisions enfocades a la millora del servei.

4.4 Diagrames de casos d'ús per actor

Usuari base

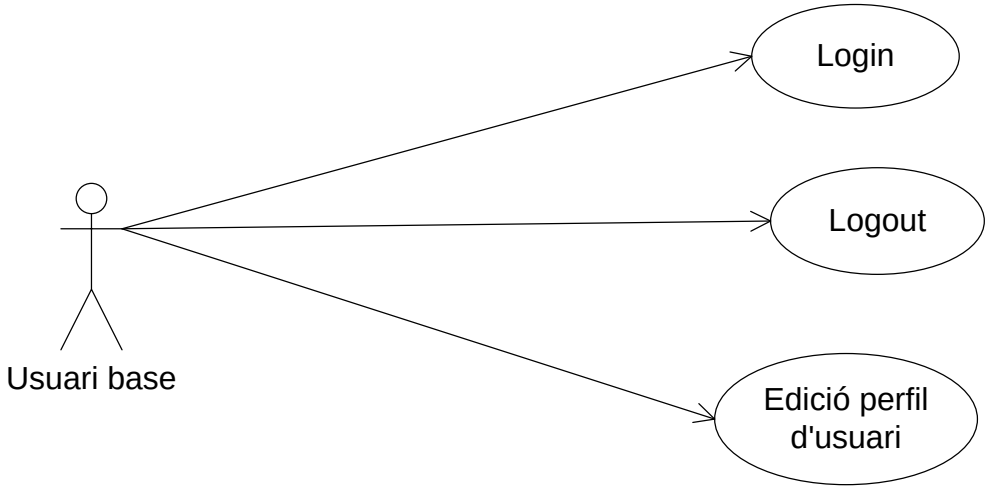


Figura 3. Casos d'us de l'usuari base

Usuari sol·licitant

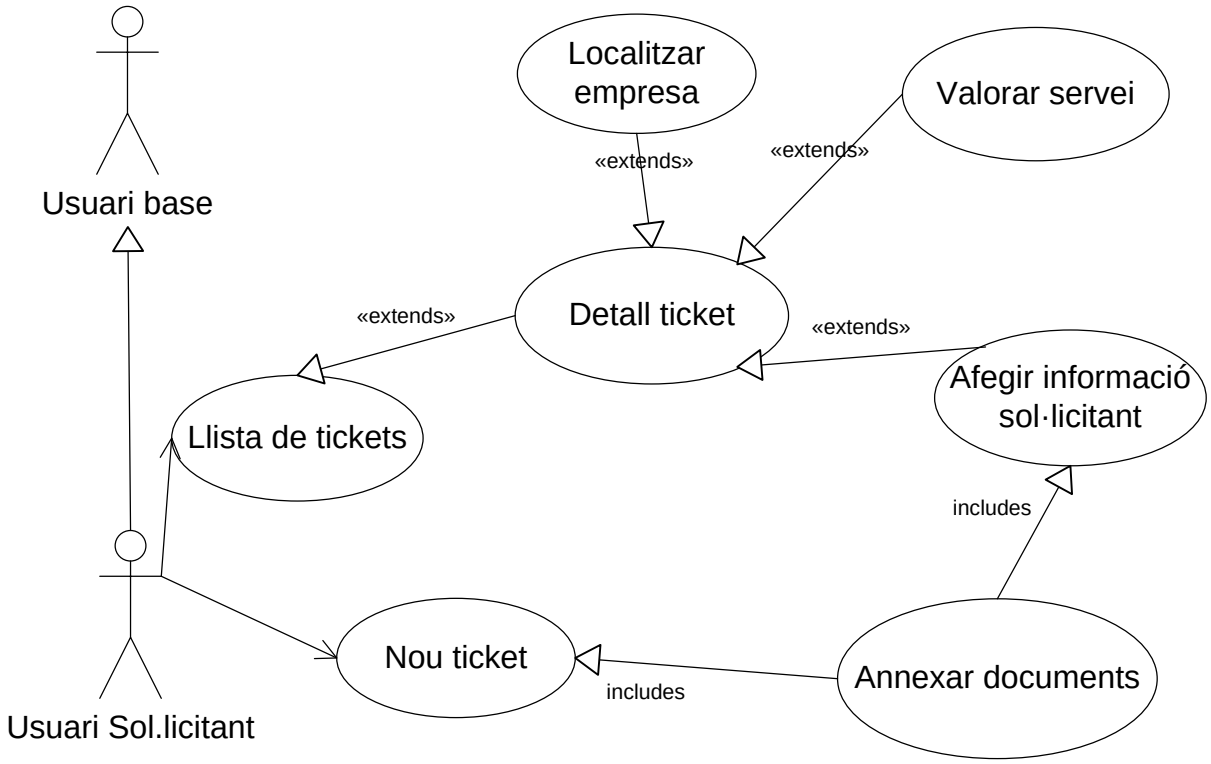


Figura 4. Casos d'us de l'usuari sol·licitant

Coordinador del Service Desk

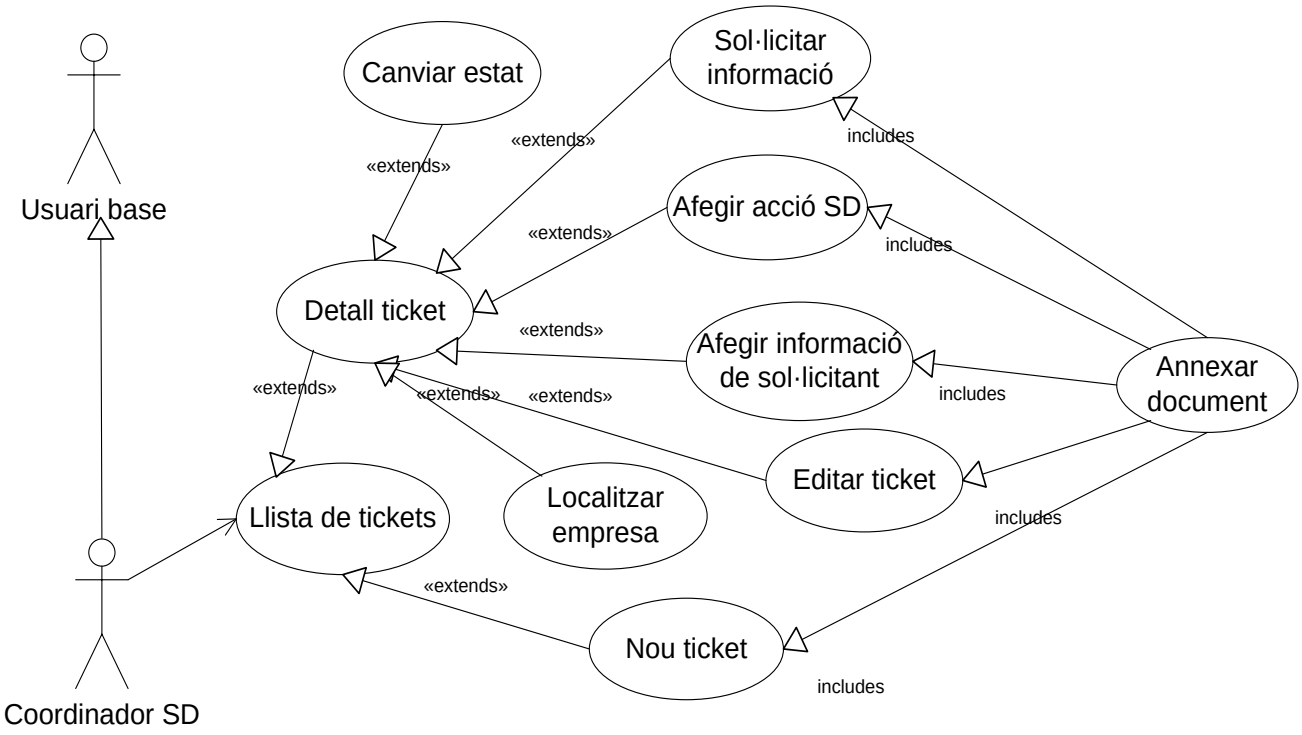


Figura 5. Casos d'us de del coordinador

Tècnic del Service Desk

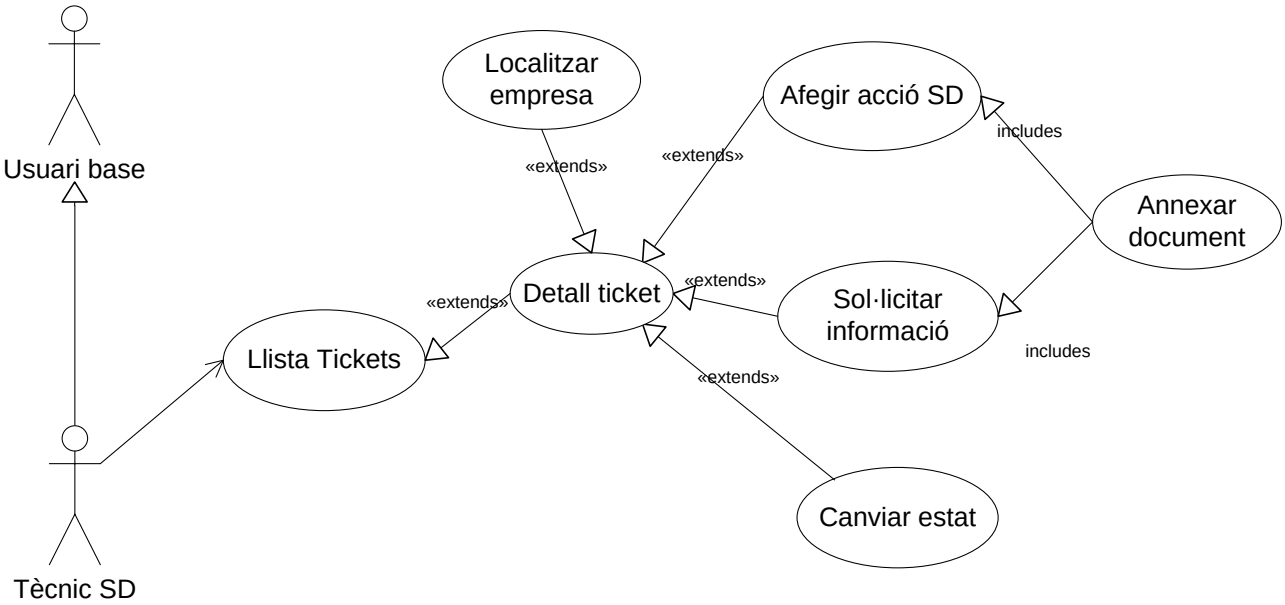


Figura 6. Casos d'us de del tècnic

Administrador del Service Desk

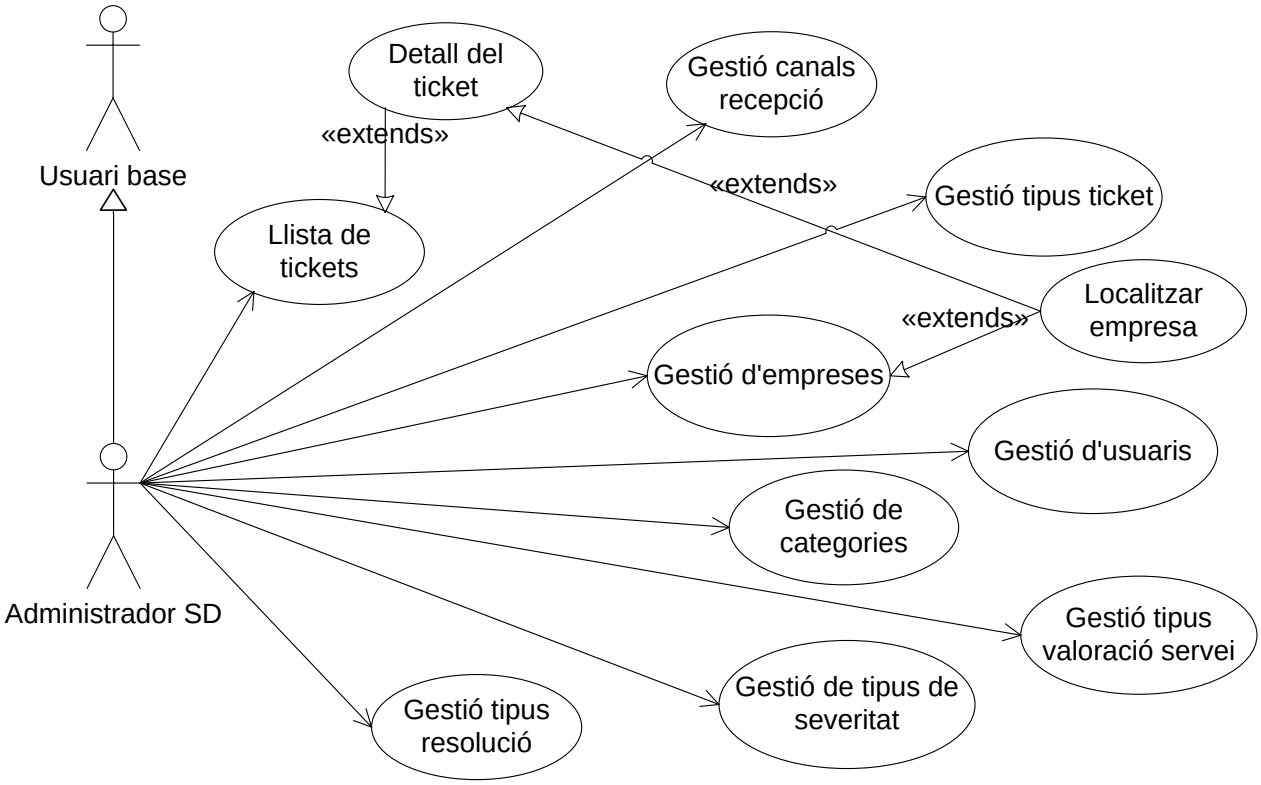


Figura 7. Casos d'us del administrador

Responsable del servei

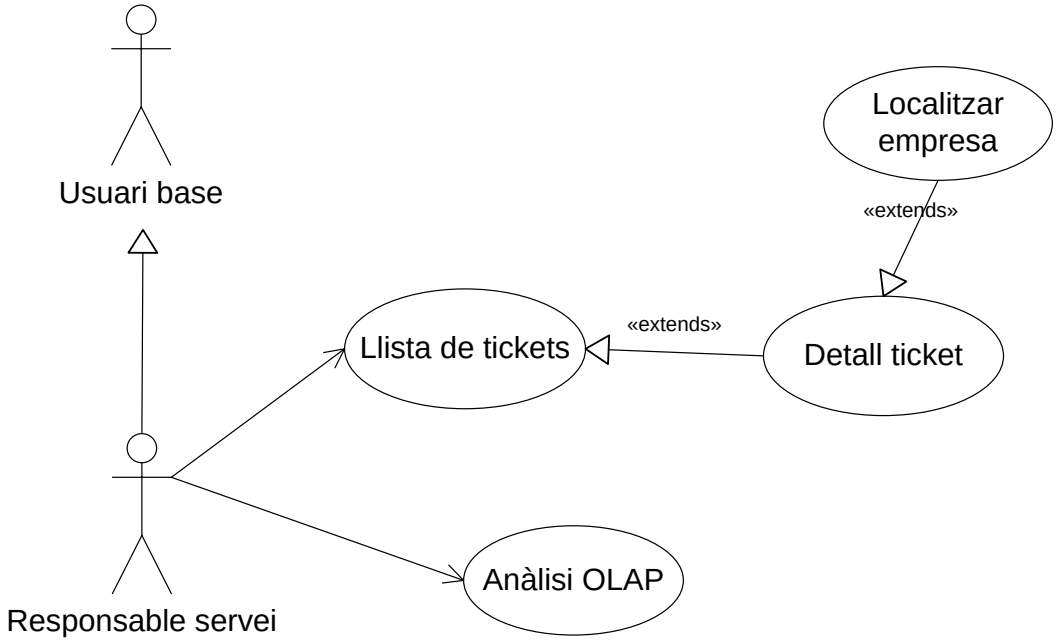


Figura 8. Casos d'us del responsable del servei

4.5 Definició textual dels casos d'us.

En l'elaboració dels casos d'us s'han substituït tots els prototips de pantalla que inicialment s'havien dissenyat en la fase de requeriments per les pantalles definitives per tal d'aconseguir un document més explícit i òbviament més proper a la realitat.

Login

Actors: Usuari base (usuari sol·licitant, coordinador, tècnic, administrador, responsable de servei).

Descripció: Permet la introducció del nom d'usuari i la paraula de pas per tal d'identificar i registrar l'usuari en el context de sessió.

Pre-condició: No estar registrat com a usuari de la sessió. Que l'usuari que s'identifica estigui donat d'alta en l'aplicació de Service Desk i no estigui en estat de baixa.

Post-condició: L'usuari s'ha identificat i registrat com a usuari de la sessió i per tant té accés a la resta de casos d'us que li han estat assignats.

Fluxe: Aquesta funció es podrà executar de manera explicita tot introduint la URL de la pàgina de Login (...\Login.jsf) o bé sempre que es detecti un intent d'utilització del programari sense que l'usuari hagi estat identificat prèviament. Pasos:

1. L'usuari introdueix el nom i la paraula de pas i prem la tecla "Login".
2. El sistema avalua la pre-condició i registra a l'usuari, altrament mostra un missatge d'error per pantalla.

Apareix una nova pantalla amb el menú superior adaptat a la funcionalitat assignada a l'usuari. Si l'usuari havia demanat un recurs i ha estat enviat a la pantalla de login, en aquest moment se'l redireccionarà cap al recurs sol·licitat originalment.

Pantalla:

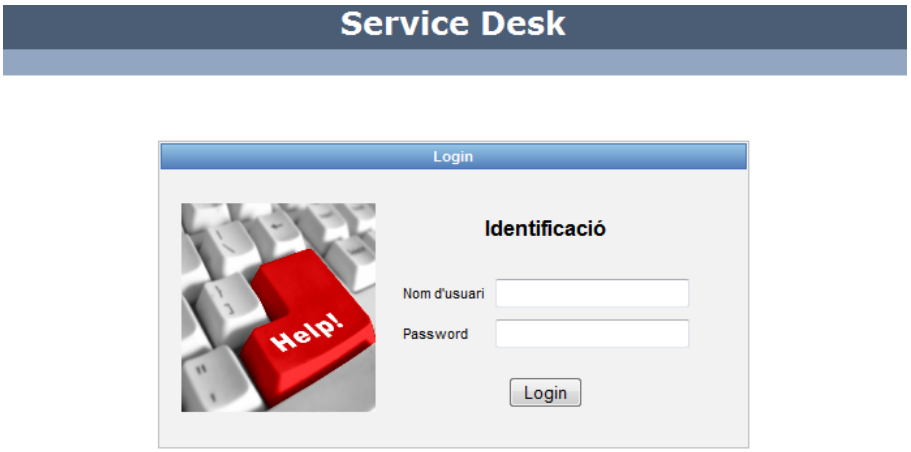


Figura 9. Pantalla de Login

Logout

Actors: Usuari base (usuari sol·licitant, coordinador, tècnic, administrador, responsable).

Descripció: Permet tancar una sessió prèviament establerta amb la funció login.

Pre-condició: Estar registrat en la sessió actual.

Post-condició: L'usuari deixa d'estar registrat en la sessió i per tant la sessió es tanca.

Fluxe: Tots els usuaris, en el menú de la part superior de la pantalla, disposen d'una opció de "Logout". Al prémer aquesta opció, el programari procedirà a tancar la sessió amb l'usuari i li mostrarà la pantalla de login.

Prototip de pantalla: No disposa de pantalla.

El meu perfil

Actors: Usuari base (usuari sol·licitant, coordinador, tècnic, administrador, responsable).

Descripció: Permet a l'usuari la modificació del seu nom i la seva paraula de pas i de les dades que el programari utilitzarà per a contactar-lo.

Pre-condició: Estar registrat en la sessió actual.

Post-condició: El programari registrarà les noves dades.

Fluxe: Tots els usuaris, en el menú de la part superior de la pantalla, disposen d'una opció d' "Editar el meu perfil". Els passos per a realitzar l'acció seran els següents:

1. L'usuari selecciona l'opció corresponent del menú.
2. L'usuari canvia les dades que creu oportunes i prem el botó "Desar perfil".
3. El programari comprova la validesa de les dades i en cas satisfactori les enregistra, altrament mostra un missatge d'error perquè l'usuari les rectifiqui.

Pantalla

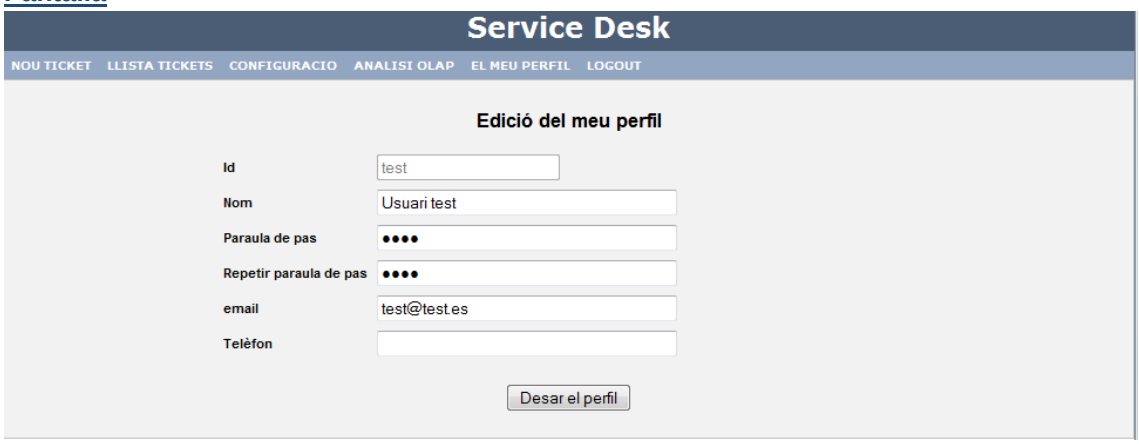


Figura 10. Pantalla d'Edició del perfil d'usuari

Llista de tickets

Actors: Usuari sol·licitant, coordinador, tècnic, responsable de servei i administrador

Descripció: Permet obtenir una llista dels tickets a partir de la introducció dels elements de cerca desitjats. Permet també especificar quina serà l'ordenació de la llista. En el prototip de pantalla podem veure quins són aquests elements.

Pre-condició: Estar registrat en la sessió actual.

Post-condició: El programari mostrarà una llista de tots els tickets que compleixin amb el criteri de cerca i ordenats en funció de l'elecció que hagi fet l'usuari.

Fluxe: Els usuaris afectats per aquest cas d'us disposaran en el seu menú superior d'una opció anomenada "Llista de tickets". Els passos per a realitzar l'acció seran els següents:

1. L'usuari selecciona l'opció corresponent del menú.
2. El programari mostra la pantalla omplint de manera automàtica els criteris de cerca i d'ordenació, atenent a l'última vegada que han estat utilitzats per l'usuari en la sessió actual. Si és la primera vegada que els utilitza, el programari ho farà d'acord amb el tipus d'actor:
   a. **Usuari sol·licitant:** Mostrarà només els seus tickets, per tant, seleccionarà l'usuari en curs dins el desplegable d'usuaris. L'ordenació per defecte serà per data de creació.
   b. **Tècnic**: Mostrarà els tickets que aquest usuari té en curs, és a dir, seleccionarà l'estat "assignat" dins el desplegable d'estats, i l'usuari en curs dins el desplegable d'assignat. L'ordenació per defecte serà per severitat.
   c. **Altres actors**: Mostrarà tots els tickets ordenats per estat.
3. L'usuari podrà canviar els criteris de cerca i ordenació i un cop fet executarà l'acció "Cercar".
4. El programari mostrarà la llista de tickets que compleixin els criteris introduïts. A l'ultima columna de la llista apareixeran les opcions que l'usuari pot executar per cadascuna de les files (consultar i/o editar en funció de l'actor).

Pantalla:

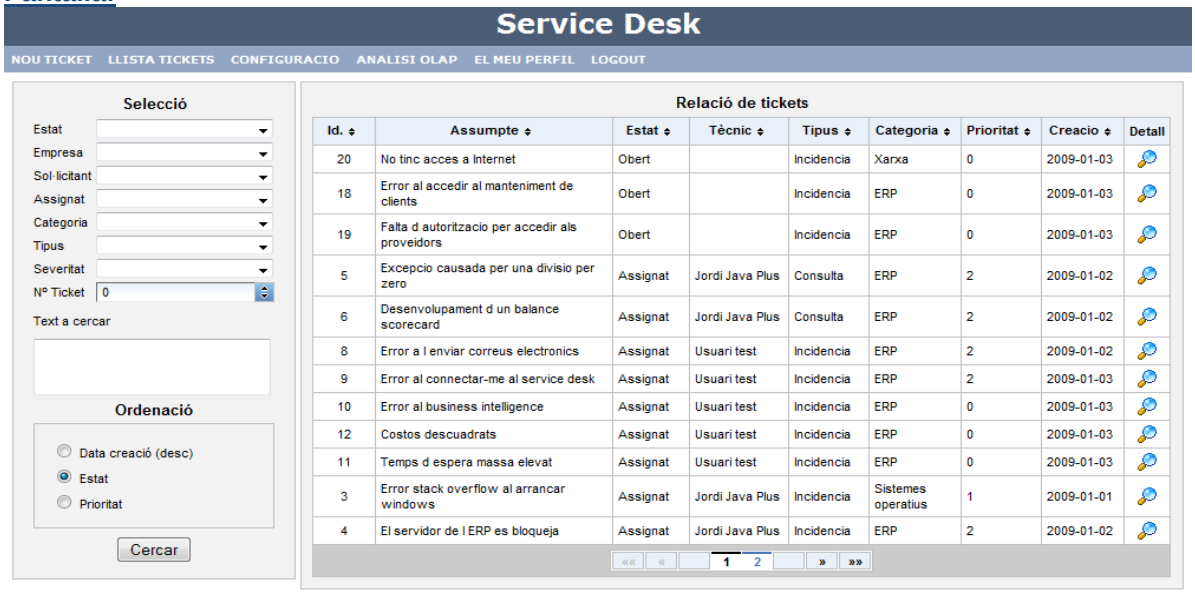


Figura 11. Pantalla Llista de Tickets

Nou ticket

Actors: Usuari sol·licitant, Coordinador.

Descripció: Permet a l'usuari l' introducció ràpida d'un nou ticket.

Pre-condició: L'usuari ha d'estar correctament registrat a la sessió.

Post-condició: El programari ha enregistrat el nou ticket.

Fluxe: Els usuaris afectats per aquest cas d'us disposaran d'una opció anomenada "Nou ticket" en el menú de la part superior de la pantalla. Per registrar un nou ticket s'hauran de seguir els següents passos:

1. Prémer l'opció corresponent al menú superior.
2. El programari mostrarà la pantalla d'introducció d'un nou ticket amb les següents dades per defecte:
    a. El canal d'entrada serà "Formulari web".
    b. La severitat seleccionada serà la que tingui un nivell de prioritat més baix.
    c. L'empresa i el sol·licitant s'ompliran automàticament si l'usuari que executa el cas d'us és un actor sol·licitant.
3. Només l'actor Coordinador tindrà capacitat per registrar nous tickets en nom d'un altre usuari, per tant, en aquest cas apareixeran uns desplegables per poder seleccionar l'empresa i l'usuari sol·licitant.
4. L'usuari introduirà les dades sol·licitades per la pantalla i annexarà els documents que cregui oportuns i executarà l'acció "Desar el ticket".
5. El programari validarà les dades introduïdes i en cas satisfactori registrarà el nou ticket i els documents annexats en el sistema tot assignant-li un número de manera automàtica. Altrament es mostrarà un missatge d'error amb indicació de les dades afectades perquè l'usuari les rectifiqui.

Pantalla

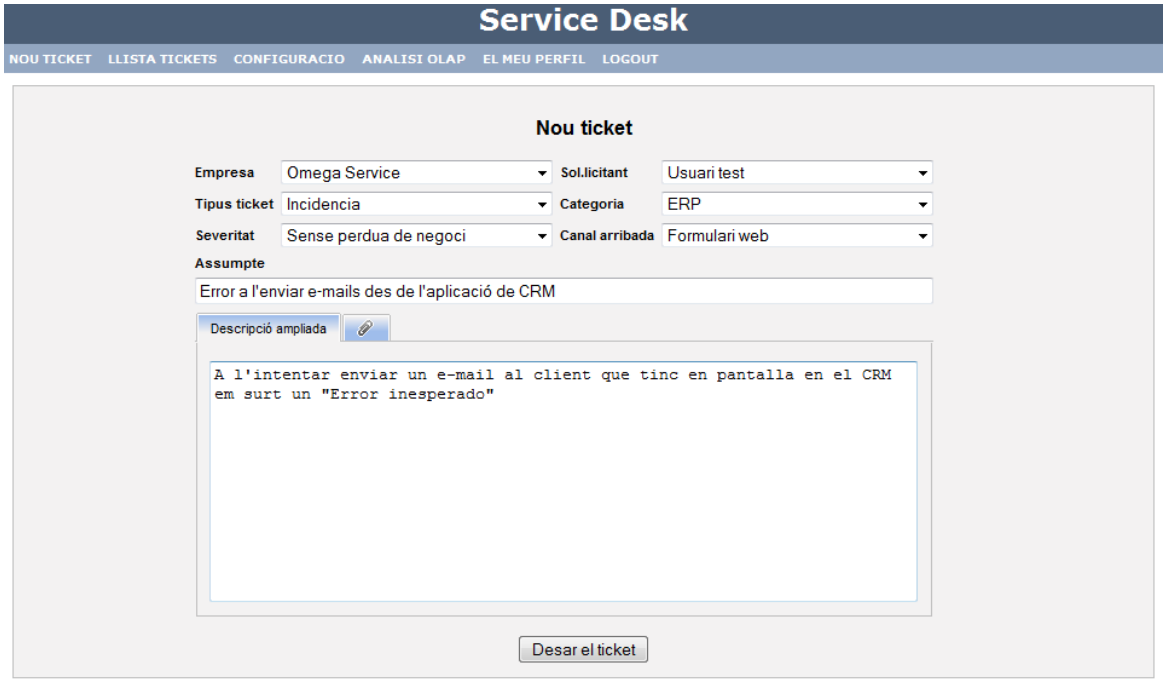


Figura 12. Pantalla Nou Ticket

Detall ticket

Actors: Usuari sol·licitant, Tècnic, Responsable Servei, Coordinador i Administrador

Descripció: Permet veure tota la informació relativa a la incidència en una sola pantalla així com obrir tots els documents annexats tant a nivell de capçalera del ticket com dins de les accions o informacions addicionals aportades.

Pre-condició: L'usuari ha d'estar correctament registrat a la sessió i el número de ticket consultat ha d'existir.

Post-condició: cap.

Fluxe: Les dades es mostren en la pantalla quan es demana la URL del ticket a consultar. En funció dels rols assignats a l'usuari registrat, aquesta pantalla mostrarà de manera contextual una sèrie d'icones per permetre l'accés a altres casos d'us. Concretament des d'aquesta pantalla podrem: Editar les dades de classificació del ticket, eliminar el ticket, canviar l'estat, canviar la descripció, editar, afegir i eliminar accions i valorar el servei ofert pel Service Desk,

Pantalla

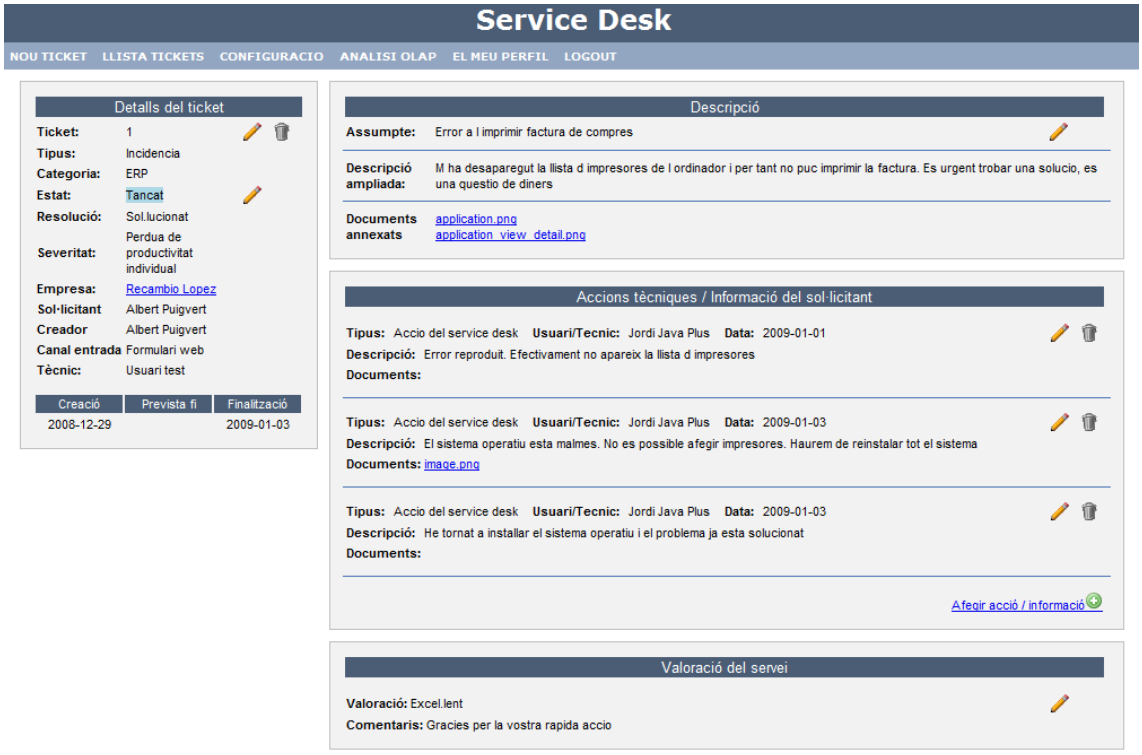


Figura 13. Pantalla Detall Ticket

Valoració del servei

Actors: Usuari sol·licitant i coordinador del Service Desk.

Descripció: Permet a l'Usuari Sol·licitant valorar el servei que el Service Desk li ha proporcionat a l'hora de solucionar un determinat ticket. La valoració es realitzarà seleccionant un item dins de l'escala de valoració introduïda per l'Administrador, possibilitant, alhora, la introducció de comentaris textuals addicionals.

Pre-condició: L'usuari ha d'estar correctament registrat a la sessió i el ticket ha d'existir.

Post-condició: El programari ha enregistrat la valoració feta per l'usuari

Fluxe: L'usuari selecciona l'element del desplegable de valoració que més encaixi amb la seva percepció del servei rebut i, si ho desitja, introdueix comentaris addicionals al respecte. Finalment executa l'acció "Desar" perquè el programari validi la informació introduïda i en cas positiu enregistri la valoració o altrament mostri un missatge d'error.

Pantalla

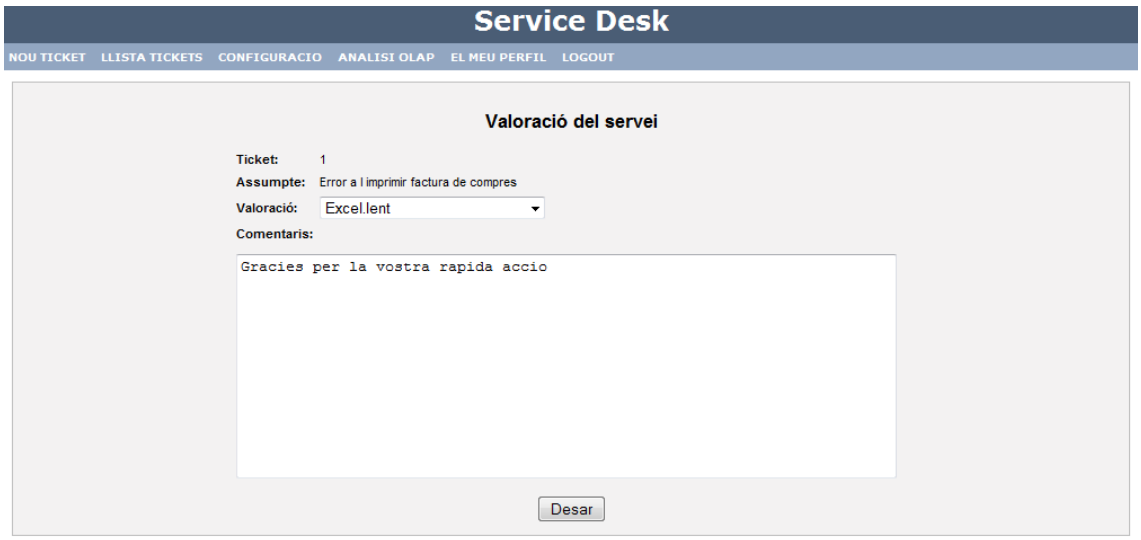


Figura 14. Pantalla Valoració del Servei

Afegir informació sol·licitant

Actors: Usuari sol·licitant, Coordinador del Service Desk.

Descripció: Permet introduir informació textual en qualsevol de les etapes del cicle de vida d'un ticket així com annexar documents mitjançant una crida al cas d'us Annexar Document.

Pre-condició: L'usuari ha d'estar correctament registrat a la sessió i el ticket ha d'existir. L'usuari ha de ser el sol·licitant del ticket, excepte en el cas de que l'usuari sigui Coordinador SD, doncs aquest podrà entrar informació en nom d'altres usuaris.

Post-condició: El programari ha enregistrat amb la data actual la informació de l'usuari i ha creat així com els documents annexats.

Fluxe: Per enregistrar una nova informació només l'hem d'introduir en el quadre de text i prémer el botó "Desar". Si desitgem annexar documents ho farem mitjançant el botó "Annexar document" (veure cas d'us "Annexar Document").

Prototip de pantalla

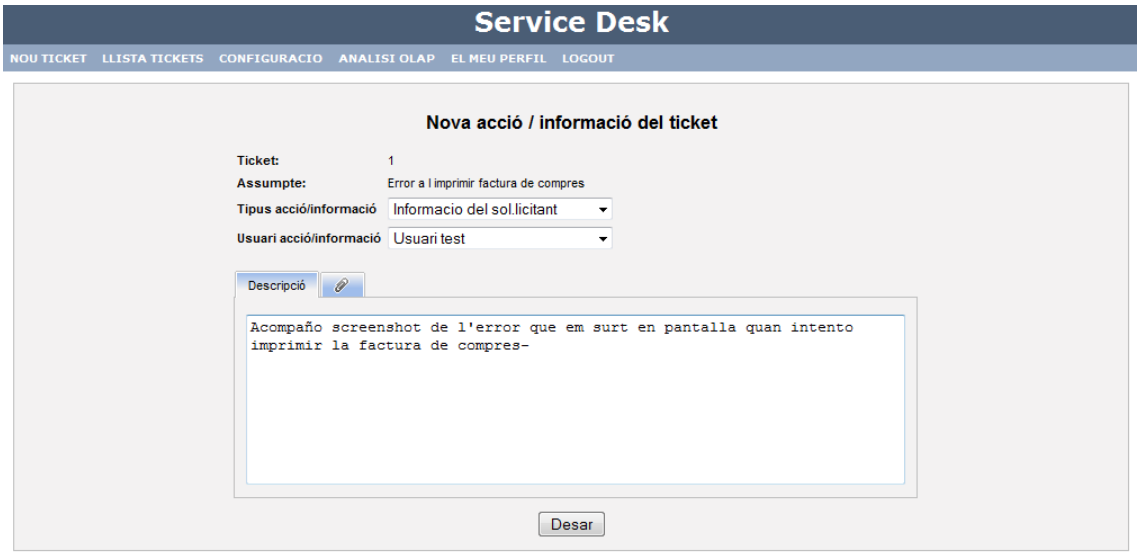


Figura 15. Pantalla Afegir Informació Sol·licitant

Afegir acció SD

Actors: Coordinador SD, Tècnic.

Descripció: Introducció de la descripció textual d'una acció realitzada pel Service Desk envers a la solució del ticket amb especificació del tècnic del SD, la data d'inici i fi de l'acció i les hores invertides. Opcionalment es poden annexar documents relatius a l'acció.

Pre-condició: L'usuari ha d'estar correctament registrat a la sessió i el ticket ha d'existir.

Post-condició: El programari ha desat les dades introduïdes per l'usuari així com el preu de tarifa vigent del tècnic en el registre de l'acció.

Fluxe: L'usuari introdueix les dades desitjades en el formulari i prem el botó desar. El programari valida la informació i en cas satisfactori la persisteix en el sistema, altrament mostra un missatge d'error. Es possible fer una crida al cas d'us "Annexar document" mitjançant el botó que explícitament ho indica.

Pantalla:

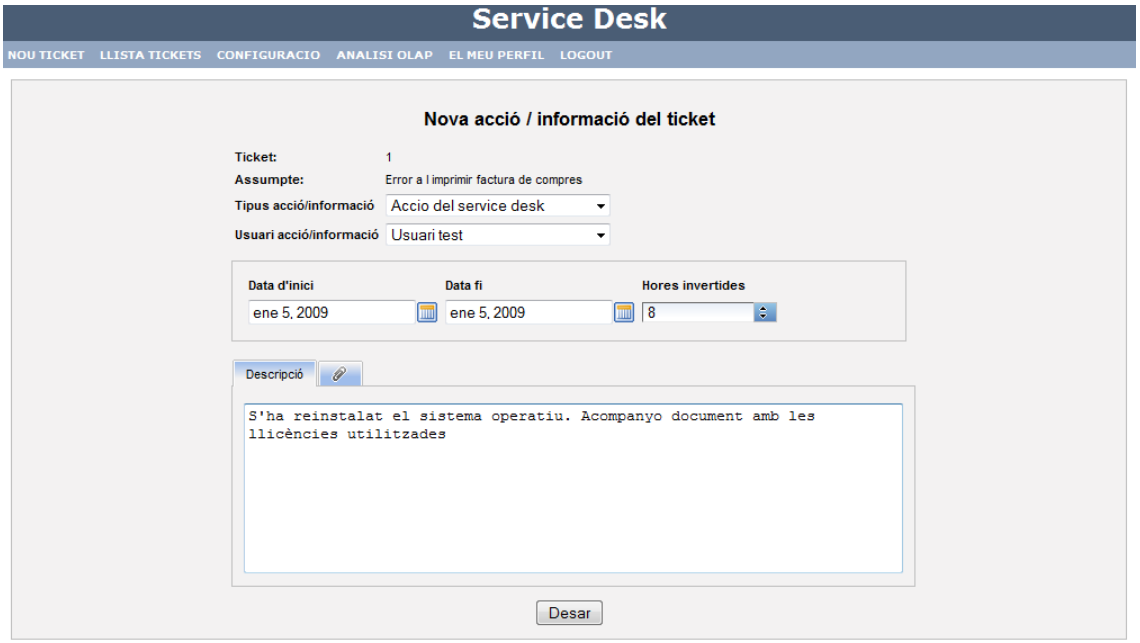


Figura 16. Pantalla Afegir Informació Service Desk

Annexar document

Actors: Usuari sol·licitant, Coordinador SD, Tècnic.

Descripció: Permet a l'usuari incrustar qualsevol tipus de document a l'aplicació de Service Desk.

Pre-condició: L'usuari ha d'estar correctament registrat a la sessió i la mida del document no pot superar la mida màxima permesa per la configuració del programari.

Post-condició: El programari ha desat el document annexat en el lloc corresponent.

Fluxe: Quan l'usuari sol·licita annexar un nou document mitjançant el botó "Afegir document" apareix un quadre de diàleg que li permetrà navegar per les seves unitats de disc i directoris locals fins a localitzar el o els arxius corresponents.

Un cop localitzats, els seleccionarà i executarà l'acció "Obrir". El programari s'encarregarà de carregar els arxius des de la unitat de l'usuari fins al web site del Service Desk.

Un cop carregat cadascun dels arxius, aquests apareixeran en una taula situada a la part dreta, juntament amb un símbol d'eliminar, per tal de poder descartar-los.

Pantalla:

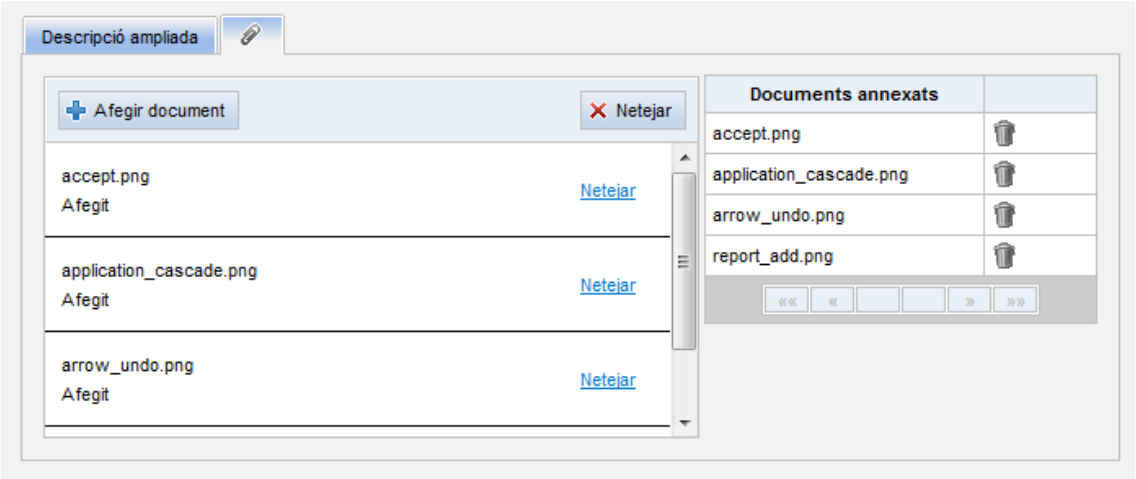


Figura 17. Pantalla Annexar Document

Editar ticket

Actors: Coordinador del Service Desk.

Descripció: El Coordinador del Service Desk és l'únic actor que té la possibilitat d'editar absolutament totes les parts de les que es compon un ticket així com d'eliminar-lo completament. Aquest cas d'us és el que li proporciona aquesta funcionalitat.

Pre-condició: L'usuari ha d'estar correctament registrat a la sessió i el ticket ha d'existir.

Post-condició: El programari ha desat tots els canvis realitzats per l'usuari .

Fluxe: Per eliminar el ticket hem de seguir els següents passos:

- Des de la pantalla o cas d'us "Detall del ticket" hem de prémer la icona en forma de cubell de la brossa que es troba a l'alçada del nombre de ticket.
- Hem de contestar afirmativament al missatge ""Vol eliminar el ticket?"

Per editar el ticket:

- Prémer la icona en forma de llapis que es troba en la pantalla de "Detall del ticket" al costat del número de ticket.
- Apareix una pantalla amb els detalls d'estat i classificació del ticket, amb l'assumpte, la descripció ampliada i la llista de documents annexats.
- Realitzem els canvis que considerem oportuns i annexarem/eliminarem documents.
- Farem un clic al botó "Desar el ticket".
- El programari validarà que les dades siguin correctes i en cas afirmatiu procedirà a persistir-les a la base de dades, altrament mostrarà un missatge d'error.

Com es pot veure en el prototip de pantalla, cadascun dels grups d'informació ofereix un sistema d'edició apropiat al tipus d'informació editable:

- Per eliminar de manera permanent el ticket hem de prémer el botó "Eliminar ticket" i contestar afirmativament a la pregunta de confirmació.
- Per editar les dades generales hem de fer les modificacions desitjades a cadascun dels corresponents controls de pantalla i posteriorment prémer el botó "Desar ticket". El programari realitzarà la validació de les dades i en cas satisfactori procedirà a persistir-les, altrament notificarà els errors a l'usuari.

Pantalla:

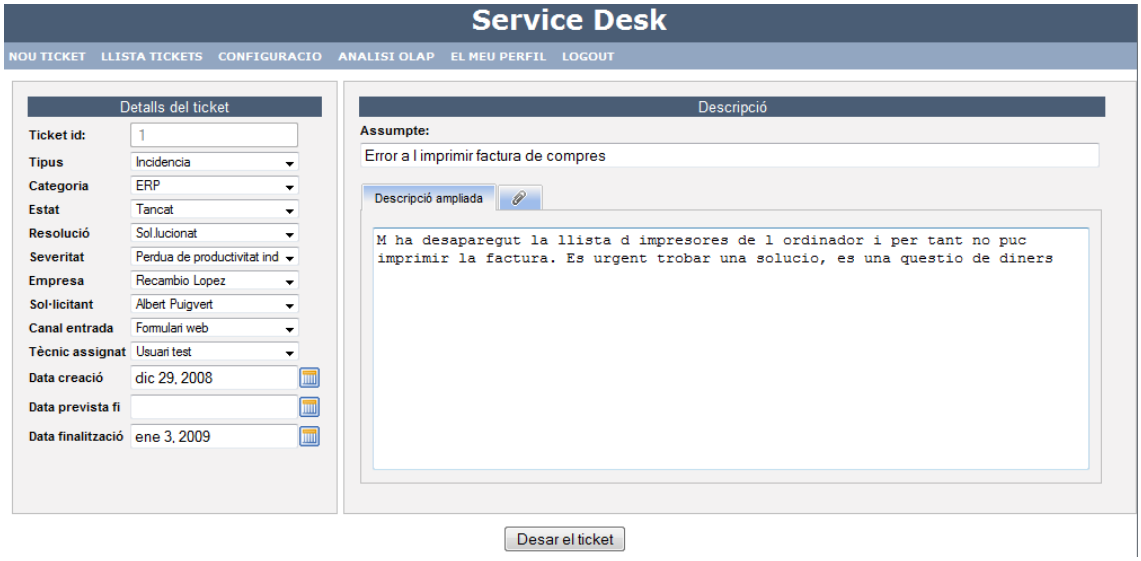


Figura 18. Pantalla Editar Ticket

Sol·licitar informació

Actors: Coordinador SD, Tècnic.

Descripció: Aquest cas d'us dona la possibilitat d'introduir en el ticket un requeriment d'informació destinat a l'usuari sol·licitant.

Pre-condició: L'usuari ha d'estar correctament registrat a la sessió i el ticket ha d'existir.

Post-condició: El programari ha desat les dades de la petició.

Fluxe: Es realitzaran els següents passos:

- Des de la pantalla "Detall del ticket" haurem de prémer la opció "Afegir acció / Informació".
- Apareix una pantalla on hi ha un desplegable de tipus d'acció. Seleccionarem "Sol·licitud d'informació".
- Introduirem les dades desitjades i farem un clic al botó "Desar".
- El programari valida la informació i en cas satisfactori la persisteix en el sistema, altrament mostra un missatge d'error.
- Es possible fer una crida al cas d'us "Annexar document" mitjançant el botó que explícitament ho indica.

Pantalla:

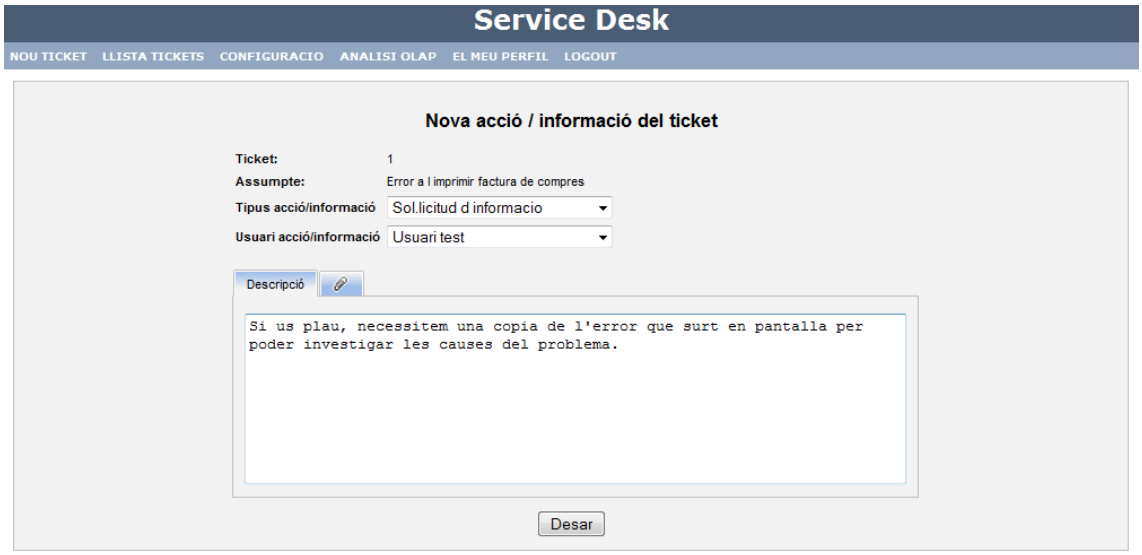


Figura 19. Pantalla Sol·licitar Informació

Canviar estat

Actors: Tècnic, Coordinador del Service Desk.

Descripció: Tant el Coordinador com el Tècnic del Service Desk tenen la potestat de realitzar canvis d'estat en el ticket en funció dels progressos que hi realitzin.

Pre-condició: L'usuari ha d'estar correctament registrat a la sessió i el ticket ha d'existir.

Post-condició: El programari ha desat les noves dades d'estat.

Fluxe: Els pasos a realitzar són els següents:

- Des de la pantalla "Detall ticket", premem la icona en forma de llapis que es troba al costat de l'estat del ticket.
- Apareix una pantalla on podem realitzar els canvis oportuns sobre l'estat, el tècnic assignat, la resolució, la data prevista de fi i la data de finalització.
- Premem el botó desar i el sistema valida les dades introduïdes tot persistint-les si no hi ha error o mostrant un missatge en cas contrari.

Pantalla:

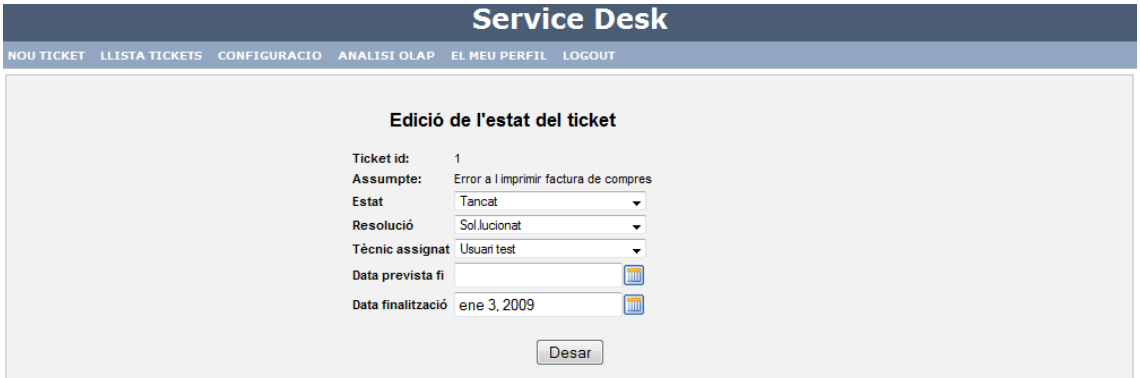


Figura 20. Pantalla Canviar Estat

Anàlisi OLAP

Actors: Responsable de Servei.

Descripció: Aquesta funció proporciona a l'actor la possibilitat de realitzar un anàlisi adhoc de les dades. La seva base és la tecnologia OLAP (On Line Analytical Process), Inicialment s'activaran una sola mètrica anomenada "Nombre de tickets" i les dimensions: "Empresa", "Categoria", "Canal de Entrada", "Tipus de Ticket", "Any" i "Mes".

Pre-condició: L'usuari ha d'estar correctament registrat a la sessió.

Post-condició: Cap.

Fluxe: Des del menú superior existent en qualsevol pantalla del Service Desk seleccionarem l'opció "Anàlisi OLAP" i s'obrirà una nova finestra amb l'eina d'anàlisi JPivot.

Utilitzant les accions corresponents de la botonera situada a la part posterior podrem, juntament amb la possibilitat de plegar i desplegar les files i columnes de la taula, realitzar totes les següents funcions típiques d'un anàlisi OLAP:

- Drill down, drill across i drill over.
- Slice and Dice.
- Filtering
- Pivoting
- Elaboració de charts simples.

Pantalla:

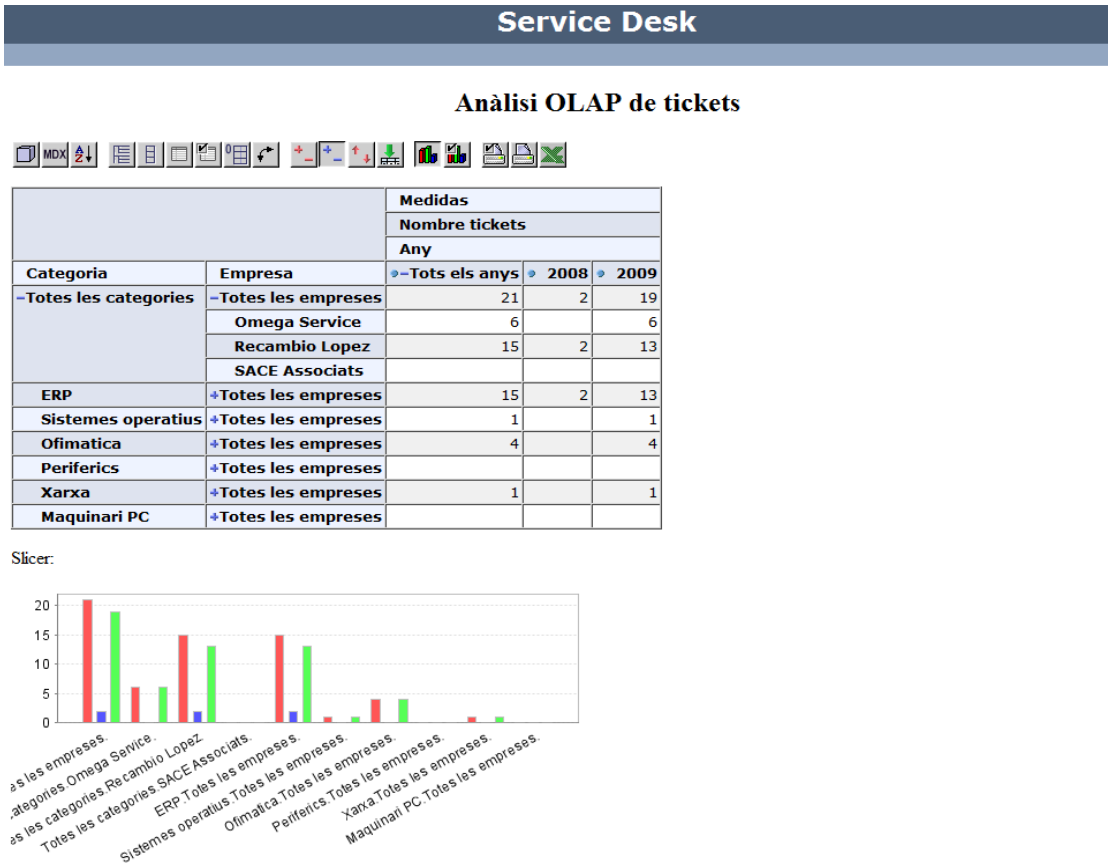


Figura 21. Pantalla Anàlisi OLAP

Localitzar empresa

Actors: Sol·licitant, Coordinador, Tècnic, Responsable, Administrador

Descripció: Aquest cas d'us pretén donar informació de la ubicació de les empreses que utilitzen el servei de Service Desk per tal de facilitar als tècnics la seva localització en cas de desplaçament.

Pre-condició: L'usuari ha d'estar correctament registrat a la sessió i el ticket ha d'existir.

Post-condició: Cap.

Fluxe: Des del cas d'us "Detall del ticket" podem prémer sobre l'empresa sol·licitant del ticket i automàticament apareixerà una nova finestra amb la informació de Google maps referent a la seva ubicació

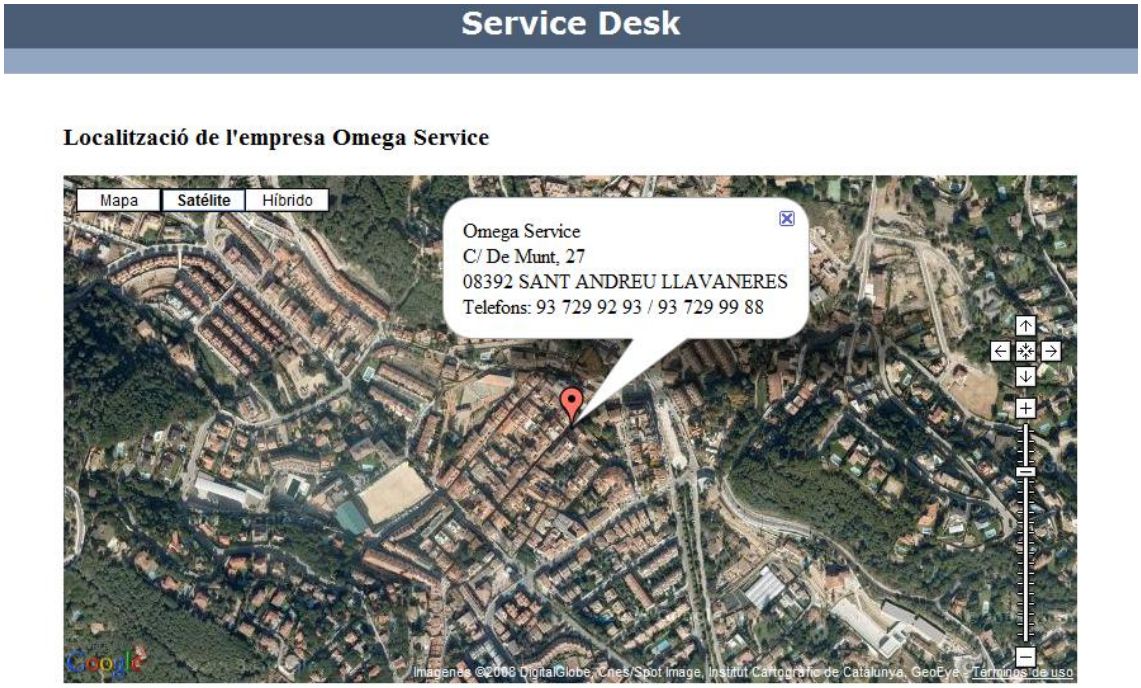


Figura 22. Pantalla Google Maps

Gestió bàsica

Aquest programari de Service Desk, com la majoria de programaris, consta d'una part de casos d'ús, que tenen la funcionalitat simple de donar d'alta, editar i suprimir registres o objectes de característiques molt bàsiques. Donat que la intenció d'aquest document és per una banda pràctica i per l'altre no pretén avorrir amb temes redundants , he preferit tractar-los de manera conjunta en aquest cas d'us i seguidament només presentar el prototip de pantalles per cadascun d'ells per poder veure la informació que s'hi mostra.

Actors: Administrador.

Descripció: Aquesta funció dona a l'administrador la possibilitat d'afegir, modificar i eliminar registres corresponents a les taules de configuració del Service Desk.

Pre-condició: L'usuari ha d'estar correctament registrat a la sessió.

Post-condició: Les dades editades s'hauran modificat a la base de dades.

Fluxe:

1. L'usuari selecciona la opció corresponent al menú de la part superior de la pantalla.
2. Es presenta una pantalla amb una llista d'elements. Les seves columnes estaran formades per les dades més significatives de cada element. La penúltima columna mostrarà si l'element està o no esta actiu, és a dir, si no té o té introduïda la data de baixa. L'última columna mostrarà les dues accions possibles que l'usuari pot realitzar sobre l'element: Editar i Eliminar. A la part superior de la pantalla apareixerà un botó que permetrà a l'usuari realitzar una nova alta.
3. L'usuari, d'aquesta manera podrà realitzar 3 accions:
    a. Eliminar un element: El programari li demanarà confirmació i en cas satisfactori verificarà que sigui possible l'eliminació de l'element, tot comprovant que no existeixin dependències a la base de dades. Si la validació és satisfactòria es procedirà a eliminar el registre, altrament s'avisarà a l'usuari que hi ha relacions que impedeixen l'acció i se'l aconsellarà que desactivi el registre, és a dir que introdueixi una data de baixa.
    b. Editar un element: Es mostrarà una pantalla amb totes les dades editables perquè les modifiqui l'usuari. Un cop fetes les modificacions, l'usuari podrà prémer el botó "Desar" i el sistema, després de validar que siguin correctes, procedirà a desar-les.
    c. Afegir un nou element: El procediment serà el mateix que l'anterior.
4. Després de qualsevol d'aquestes tres accions, el programari renovarà la llista d'elements d'acord amb les modificacions realitzades.

Prototip de pantalla : En els següents casos d'us veurem els protitips de cadascuna de les pantalles.

Gestió d'usuaris

Pantalla

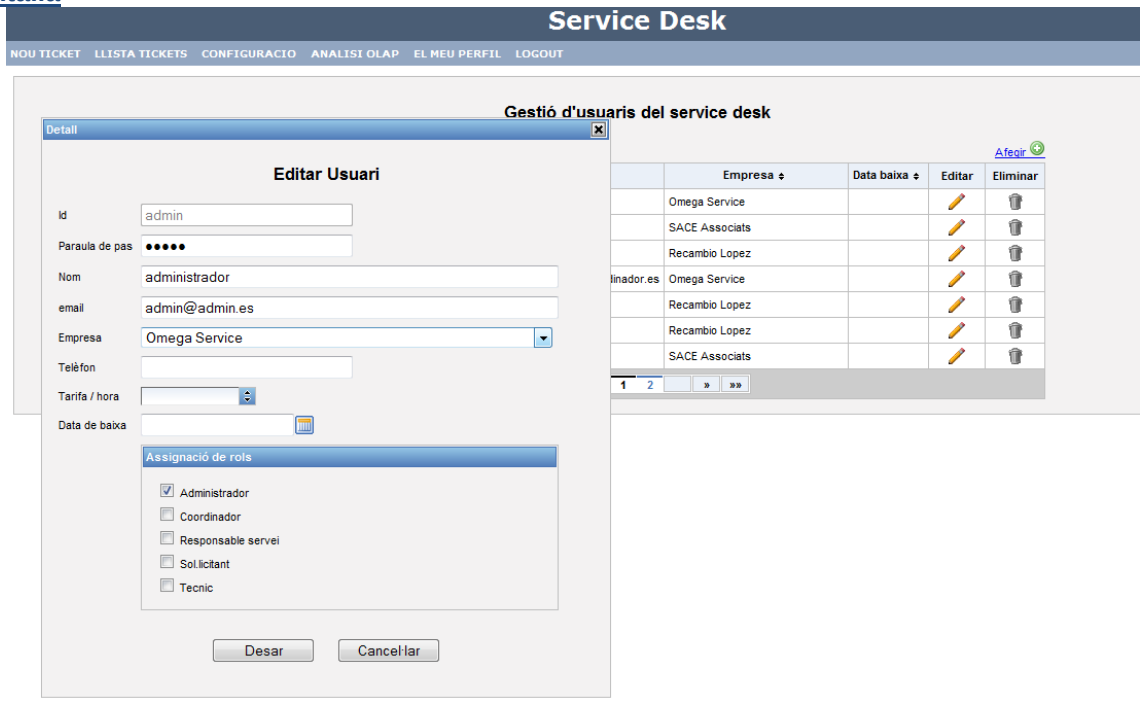


Figura 23. Pantalla Gestió d'usuaris

Gestió de canals d'entrada

Pantalla

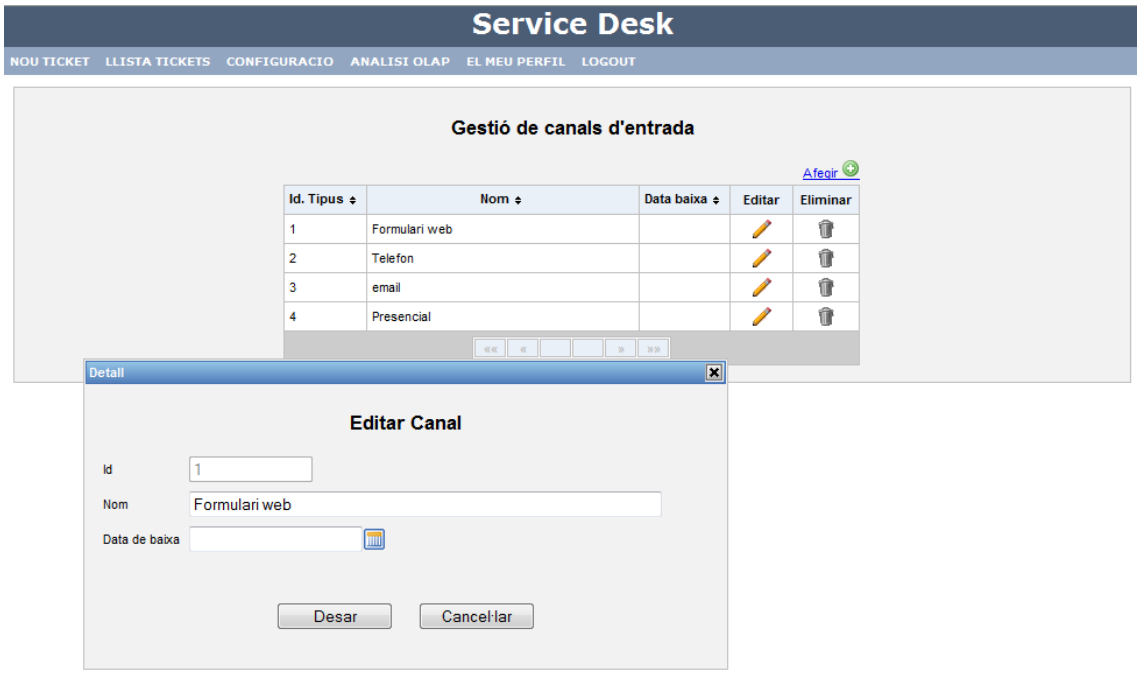


Figura 24. Pantalla Gestió canals

Gestió de categories

Pantalla

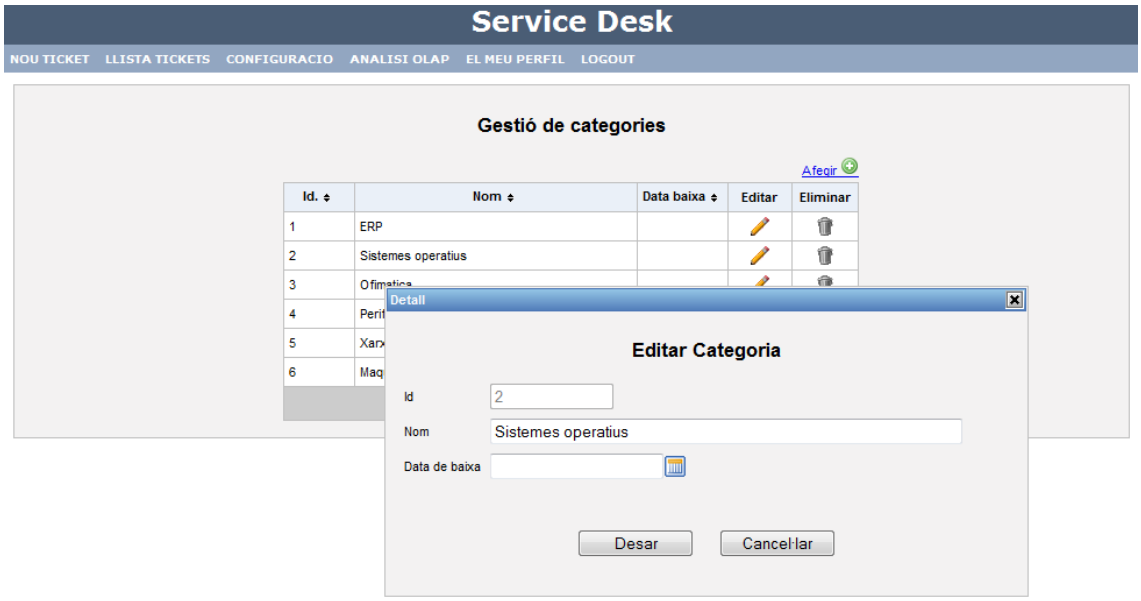


Figura 25. Pantalla Gestió categories

Gestió d'empreses

Pantalla

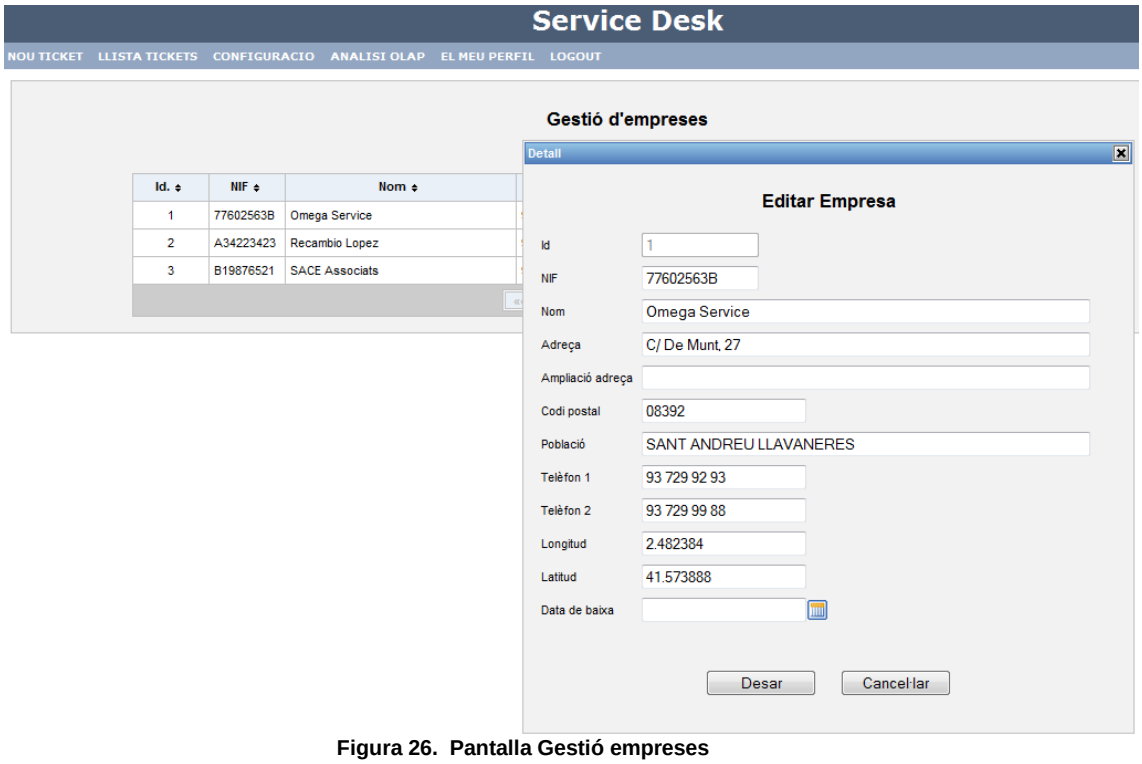


Figura 26. Pantalla Gestió empreses

Gestió de resolucions

Pantalla

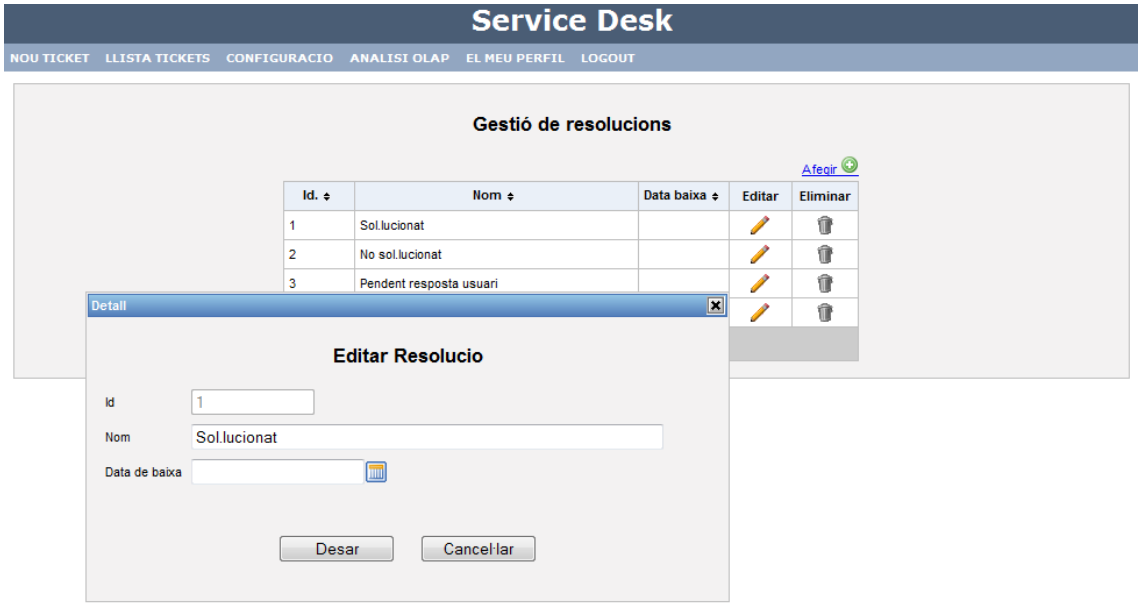


Figura 27. Pantalla Gestió resolucions

Gestió de severitats

Pantalla

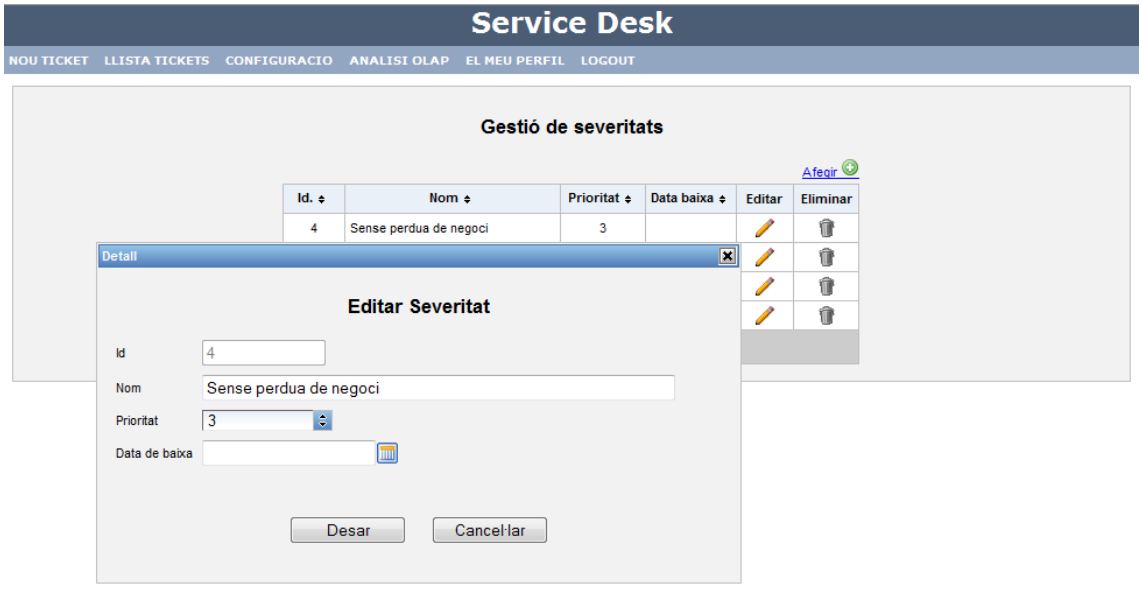


Figura 28. Pantalla Gestió severitats

Gestió de tipus de ticket

Pantalla

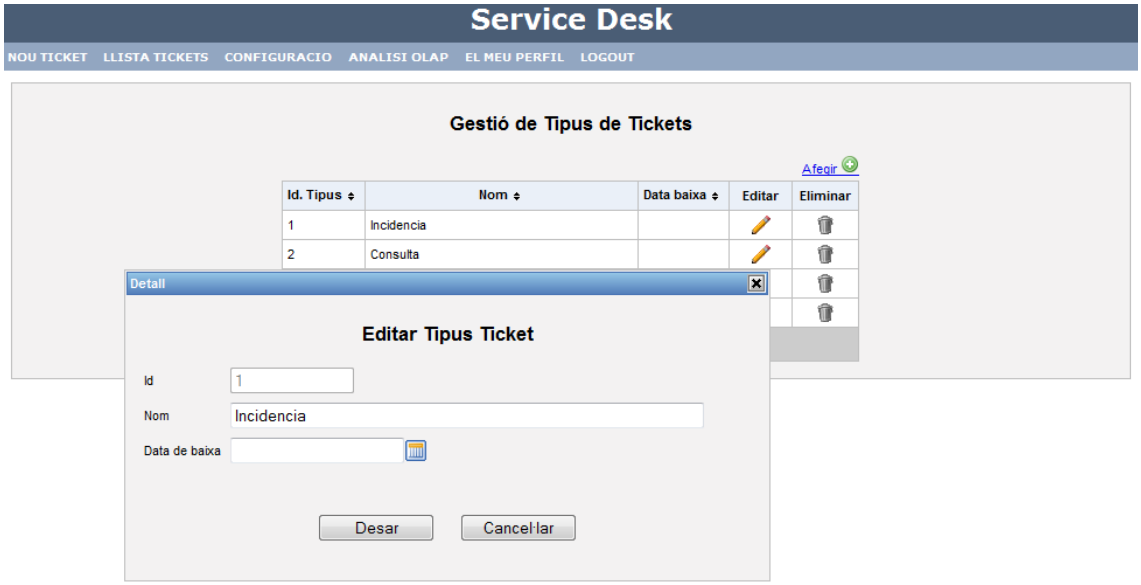


Figura 29. Pantalla Gestió tipus de ticket

Gestió de valoració del servei

Pantalla

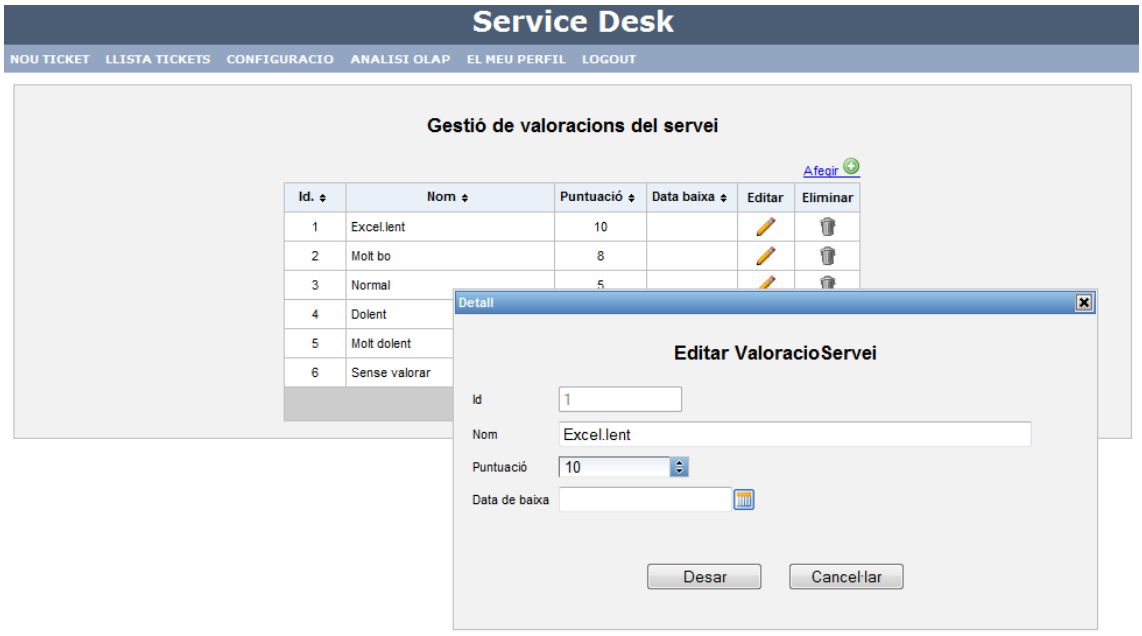


Figura 30. Pantalla Gestió valoració del servei

4.6 Circuit bàsic d'una incidència o sol·licitud

La incidència o sol·licitud informàtica és la pedra angular de l'aplicació del Service Desk, entendre el seu circuit bàsic és primordial per comprendre el funcionament de tot el sistema.

En la següent il·lustració no hem utilitzat la notació UML de diagrama d'estats, sinó que hem preferit un diagrama menys ortodoxa però més intel·ligible per al nostre client:

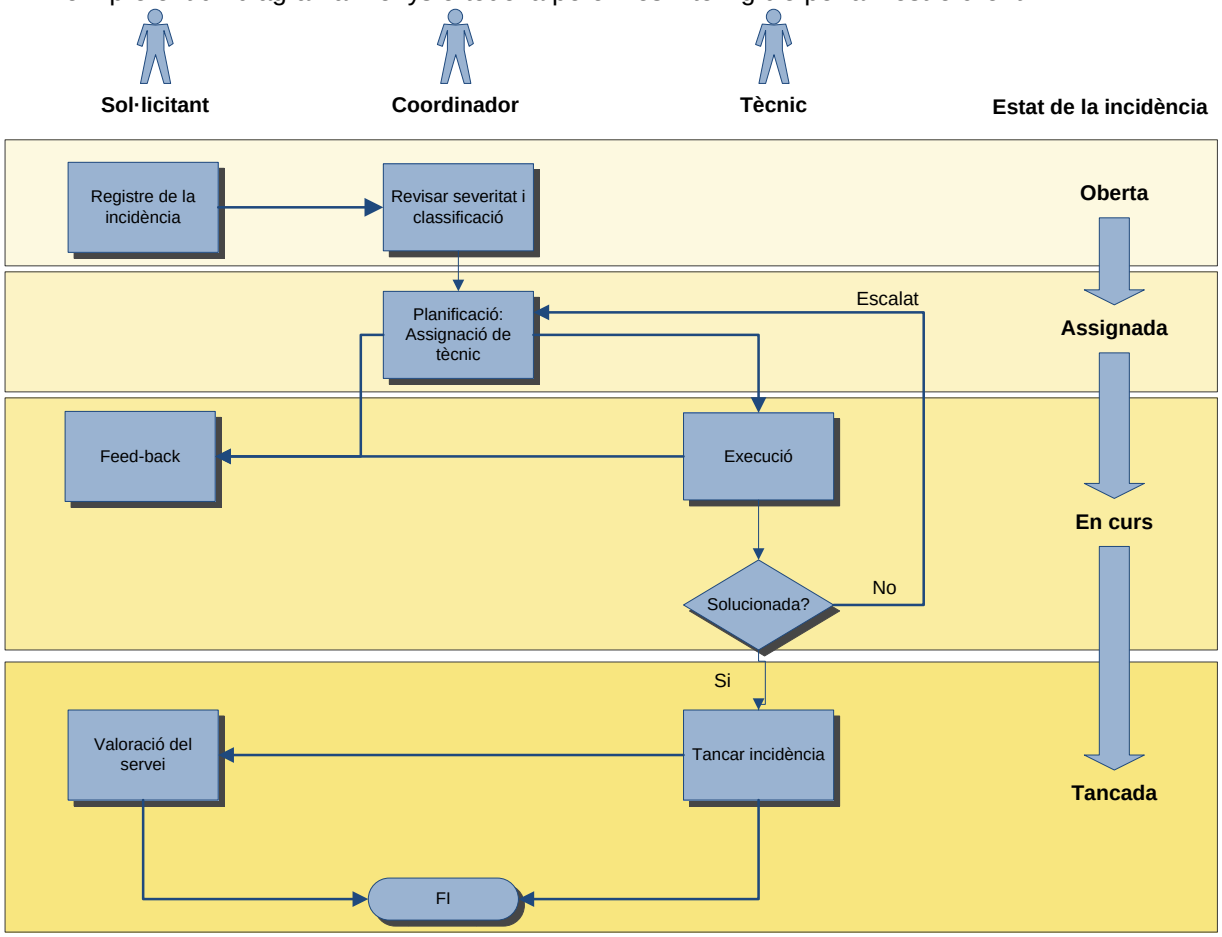


Figura 31. Circuit bàsic d'una incidència

En aquest diagrama podem veure la intervenció de tres dels actors principals: **L'usuari sol·licitant**, que és promotor de la incidència o sol·licitud, el **coordinador** del centre d'atenció que seria la persona que rebria la petició i la planificaria i el **tècnic** del centre d'atenció que seria l'encarregat de proporcionar-li la solució adequada.

El circuit comença en el moment que, l'usuari fa arribar una incidència o petició, ja sigui via email, de manera presencial, per telèfon o per formulari Web, al centre d'atenció. Aquesta acció provocarà que la incidència es consideri "**Oberta**".

El coordinador s'encarregarà de registrar la petició si aquesta no s'ha rebut de manera electrònica, i seguidament de classificar-la i de revisar la seva severitat o prioritat. Posteriorment la planificarà, és a dir, li assignarà un tècnic responsable i una data prevista de fi, provocant que la incidència agafi l'estat "**Assignada**".

En funció de la planificació, el tècnic responsable executarà els passos corresponents per a trobar la solució correcte i ho indicarà així al sistema de gestió. Durant l'execució, la incidència mostrarà l'estat de "**En curs**".

Finalment, si el tècnic ha aconseguit la resolució, la incidència passarà a l'estat de "**Tancada**" i l'usuari serà convidat a valorar el servei obtingut, i, en cas contrari, si la solució no ha estat possible, la incidència passarà una altra vegada al coordinador perquè realitzi un escalat i l'assigni a un tècnic més especialitzat.

4.7 Requeriments no funcionals.

1. Interfície d'usuari

L'elecció de d'interfície d'usuari és un punt clau que condicionarà extremadament tant l'arquitectura, com el desenvolupament com el posterior manteniment i sobre tot l'experiència d'usuari.

Hem cregut convenient, donat aquest impacte i sobre tot pel que respecte a l'impacta sobre el client final, que sigui en aquesta fase on les dues parts estableixin aquesta part com un requeriment no funcional.

Tenint en comte que en aquest moment només podem escollir entre dos tipus d'interfície, per una banda la basada en client Web (thin-client) i centrada en els navegadors més importants del mercat com Internet Explorer, FireFox, Opera, etc. i per l'altra la basada en la utilització de formularis d'escriptori (fat-client) hem realitzat un estudi dels avantatges i inconvenients de cadascuna d'elles que ens ha de permetre prendre la decisió adequada:

Característica	Interfície	Descripció
Instal·lació	Navegador	No necessita d'una instal·lació del programari. Els navegadors web tenen la capacitat suficient per accedir al web site del Service Desk i obtenir els elements necessaris per a representar i interactuar amb la informació. **(+)**
	Desktop	Necessita d'una instal·lació del programari que normalment és dependent de la plataforma on s'executa. No és el mateix programari per un Windows 98 que per un Windows Vista o que per un sistema Linux. **(-)**
Actualització	Navegador	Al igual que no necessita instal·lació tampoc necessita actualització de programari. Quan l'usuari accedeixi amb el navegador al web site ja rebrà els elements més actualitzats. **(+)**
	Desktop	Necessita o bé d'un procés d'instal·lació manual o bé d'un procés d'instal·lació automàtic que també són dependents de la plataforma on s'executa i no solen estar absents de problemes **(-)**
Multi plataforma	Navegador	Cada plataforma disposa dels seus navegadors web adaptats a la mateix sistema operatiu, proporcionant independència de la plataforma en vers als continguts de la xarxa. Per això podem afirmar que el programari dissenyat per a navegador és cross-platform. **(+)**
	Desktop	Depenent del llenguatge de programació que utilitzem podem aconseguir abastar més d'una plataforma (java), però per norma general el programari serà dependent de la plataforma on s'executa. **(-)**
Programació	Navegador	La programació és molt complexa i orientada a la no connexió amb els conseqüents problemes a l'hora de controlar les sessions de treball. Per aconseguir un nivell de funcionalitat mínima per a l'usuari ha de combinar multitud de tecnologies i/o llenguatges

Característica	Interfície	Descripció
		(DHTML, ASP, AJAX, XML, Javascript, CSS...) **(-)**
	Desktop	Programació orientada a la connexió i amb facilitats per controlar l'estat de la sessió. Programació més senzilla i que requereix menys base de coneixement. **(+)**
Seguretat	Navegador	Nivell de seguretat més baix. El món dels navegadors està més subjecte a l'exploració de vulnerabilitats per parts mal intencionades **(-)**
	Desktop	Nivell de seguretat més alt **(+)**
Estandardització	Navegador	El món del navegador és més sensible a que els desenvolupadors tinguin cura de les normes d'usabilitat i de l'aplicació dels estàndards de la industria. **(+)**
	Desktop	Té tendència, degut a la facilitat de programació i a l'existència de més possibilitats, a incorporar parts poc estandarditzades.(-).
Accés	Navegador	Permet aconseguir el que s'anomena alta disponibilitat amb les tres A, **a**nywhere, **a**nytime, **a**nyhow. Des de qualsevol lloc, amb qualsevol dispositiu i a qualsevol hora podem interactuar amb el nostre sistema. **(+)**
	Desktop	Només podem accedir a l'aplicació des de les estacions de treball que tinguin el programari degudament instal·lat. **(-)**
Procés de la informació	Navegador	La informació es processa de manera centralitzada. És el host el que s'encarrega tant de la persistència de les dades com de les validacions de la lògica de negoci i de l'execució dels càlculs. En aquest aspecte, al realitzar-se tot en el servidor, el client no necessitat cap capacitació especial. **(+)**
	Desktop	El client normalment s'encarrega tant de la validació de les dades, com de la persistència i dels càlculs. Els clients han de tenir altes capacitats per portar-ho a terme. **(-)**
Funcionalitat	Navegador	El navegador web va ser concebut en els seus inicis només per mostrar pàgines estàtiques fetes amb HTML, la seva evolució l'ha portat cap a la possibilitat de la interacció amb les dades, però això no s'ha fet d'una manera consistent sinó de manera evolutiva i amb la necessitat de conservar les compatibilitats. El producte final és una barreja de tecnologies que en ocasions utilitzen més "trucos" que no pas "tècnicas" i que després de molts esforços humans pot aconseguir donar una nivell de funcionalitat acceptable a l'usuari, però en cap cas comparable a l' ofert pel client desktop. **(-)**
	Desktop	El client de desktop, al estar més lligat a la plataforma i al poder interactuar amb ella, amb les seves llibreries i utilitats i amb les aplicacions que s'hi troben instal·lades, es capaç d'aportar d'un nivell de funcionalitat molt per sobre del navegador web, dotant-lo alhora d'una gran agilitat i rapidesa en el que podríem dir manipulació de la informació. **(+)**

Figura 32. Quadre comparatiu interfície d'usuari

Aquest estudi de pros i contres ens porta a determinar que és un requisit no funcional de l'aplicació que la interfície d'usuari sigui basada en Web.

2. **Plataforma de desenvolupament.**

   Serà també un requisit la utilització de J2EE com a tecnologia de desenvolupament del sistema Service Desk per les seves característiques d'orientació a objectes, seguretat i independència de la plataforma, implementació en open source i per l'amplia acceptació que té en el mercat.

   S'estableix també com a requeriment el desacoblament de l'aplicació amb respecte a la base de dades per tal de que l'aplicatiu sigui fàcilment integrable amb altre programari o sistemes ERP que utilitzin altres SGBD.

4.8 Funcionalitats de les properes versions.

- És un requeriment per a la segona fase que l'usuari rebi feedback via email cada vegada que la seva incidència sigui modificada o actualitzada.
- Desenvolupament d'un servei Web per poder introduir incidències des d'altres sistemes de programari.
- Sistema de cerca indexada per proporcionar al Service Desk propietats de Kwonledge Base.

5. DISSENY.

5.1 Introducció.

És l'objectiu d'aquest apartat el de donar una solució degudament justificada a tots els requeriments especificats en el capítol anterior.

La definició de l'arquitectura, els diagrames estàtics de cadascuna de les capes de l'aplicació, el model ER de la persistència i els diagrames de seqüència ens proveiran de tots els elements clau per poder realitzar la fase d'implementació amb garanties d'èxit.

5.2 Disseny Arquitectònic.

1. Arquitectura de l'aplicació.

Tota aplicació de l'àmbit empresarial conté codi referent a la presentació o interfície d'usuari, codi de processament i validació de dades i codi d'emmagatzemament de les dades. La separació lògica d'aquestes tres parts ens donarà uns avantatges bàsics sobre qualsevol altra aplicació amb arquitectura monolítica o bé de dues capes (base de dades i presentació):

- **Separació de cadascuna de les responsabilitats** en el desenvolupament, permetent al/als desenvolupadors enfocar-se en nivells d'abstracció relativament senzills i dominis fàcilment controlables i traçables.
- **Facilitat en el manteniment.** La separació i encapsulació de cadascuna de les tres feines en facilita el seu posterior manteniment.
- **Alt grau de reutilització**: El nivell més baix de dependència o acoblament dels components augmenta el seu grau de reutilització.
- **Arquitectura extensible** provocat per una **granularitat** molt més fina.
- **Arquitectura escalable**. La possibilitat de separar físicament cadascuna de les parts fins i tot en diferents sistemes de hardware afavoreix l'escalabilitat del sistema.
- **Arquitectura més oberta i flexible**: La separació de la lògica de negoci en una capa diferent ens brinda la possibilitat d'incorporar noves tecnologies, com serveis web, interfícies d'usuari per dispositius mòbils, tecnologies distribuïdes, etc.

Hem de ser també conscients que aquesta decisió arquitectònica ens causarà uns petits inconvenients que haurem de salvar utilitzant **eines de generació de codi** i **frameworks** de treball. Aquests són bàsicament:

- **Increment més que substancial de línies de codi** font en el projecte i increment del nombre d'indireccions en l'execució dels mètodes.
- **Redundància de tasques** (validacions tant a la base de dades, com al business lògic, i en alguns casos en la presentació).

- **Necessitat de comunicar les capes** en temps d'execució.

En la següent imatge podem veure un diagrama que reflecteix, a alt nivell, les decisions arquitectòniques preses:

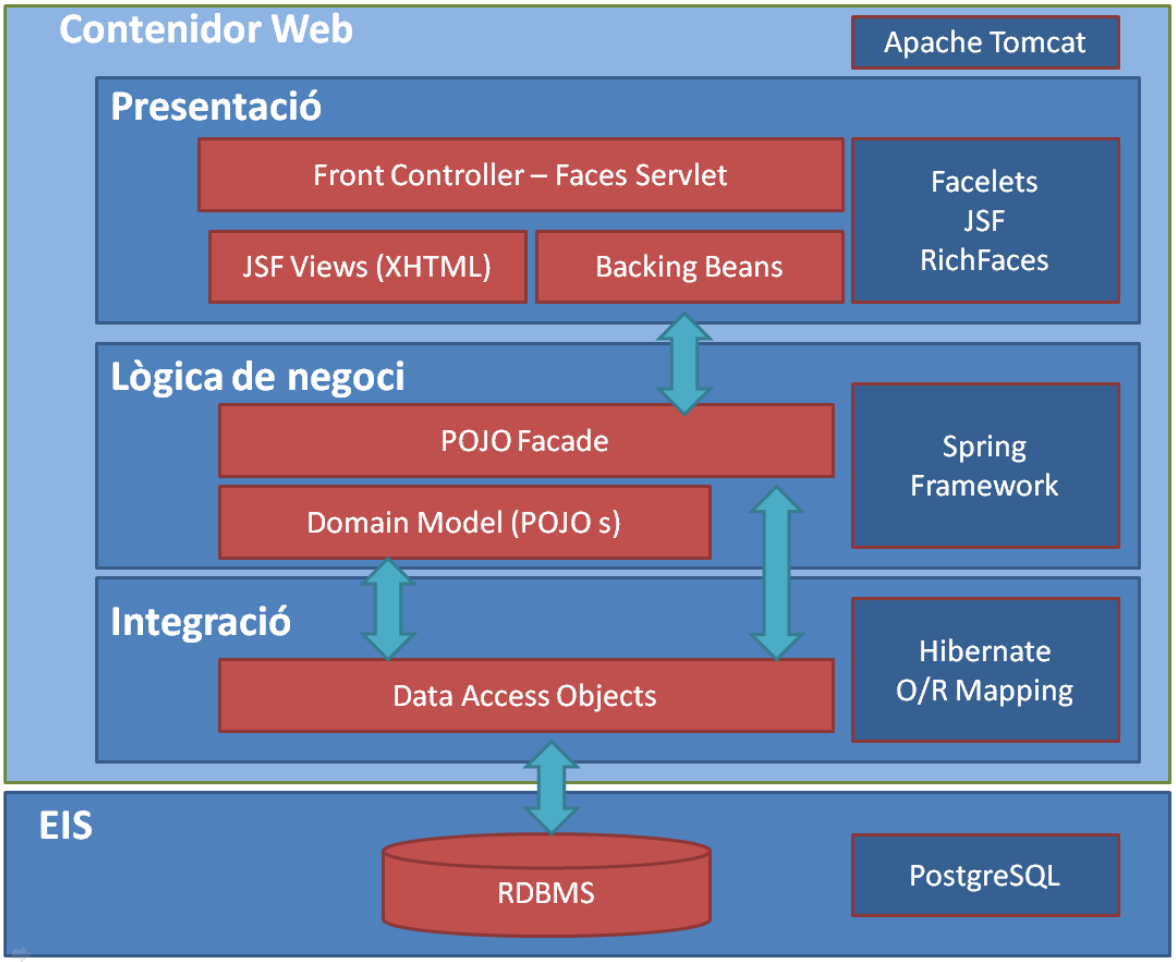


Figura 33. Arquitectura en capes

Passem ara a estudiar més en detall cadascuna de les capes:

2. Business Logic.

Aquesta capa és la que conté tots els objectes i serveis de negoci de l'aplicació. És la que s'encarrega de rebre les peticions de la capa de presentació, validar-les i processar-les utilitzant els recursos necessaris de la capa de base de dades.

Les seves responsabilitats seran:

- Dotar a la capa de presentació dels serveis necessaris.
- Proveir dels elements de control de seguretat, usuaris i rols.
- Validar que les dades rebudes per la capa de presentació siguin correctes.

- Gestionar la persistència de les dades assegurant-ne la seva integritat mitjançant:
  - Sistema de transaccions declarativa.
  - Sistema de control de concurrència optimista.

Dins les possibilitat que ofereix la plataforma J2EE i tenint en compte la premissa de que el programari de Service Desk és força reduït i no ha de participar en cap arquitectura d'aplicacions distribuïdes, hem escollit una **solució lightweight** (poc pesada) que pugui funcionar en qualsevol servidor que disposi d'un **contenidor web** com podria ser **Apache Tomcat** i, per tant, ens allunyem de la necessitat d'utilitzar servidors d'aplicacions més pesats com JBoss, Weblogic, etc.

Així la solució proposada tindrà com a base la utilització d'elements senzills com són els **POJOs** (Plain Old Java Object) en contra d'elements més complexes com són els EJBs (Enterprise Java Beans) que al gaudir dels serveis oferts pels servidors d'aplicacions, han d'incorporar les interfícies demanades.

Al deixar de banda aquests servidors d'aplicacions, delegarem en el framework **Spring**, mitjançant els seus serveis **AOP** (aspect object programming) , la gestió de la seguretat i el control sobre la persistència i les transaccions.

Spring ens ajudarà també, mitjançant el seu sistema de creació i injecció declarativa de beans **IOC**, a acoblar en temps d'execució tots els elements de les tres capes.

Organització del business logic

Dins del paradigma de la orientació a objectes i atenent al ventall de models o patterns ja oferts dins del camp l'enginyeria de programari ens decantem per la utilització del "**Domain Model**", que com estableix Martin Fowler, és un conjunt d'objectes interconnectats mitjançant associacions o herències, on cada objecte representa un element altament significatiu del domini de l'aplicació.

Dins d'aquest patró, cadascuna de les classes que representen els objectes disposen de una part de dades, actuant com a contenidors, i una part de comportament. Hem de tenir en compte, però, que aquest comportament no pot ser completament estès, concretament no pot portar accessos a la capa de dades, donat que molts d'aquests objectes seran transferits a la capa de presentació i que aquesta no té accés en absolut a la base de dades.

Aquesta part del comportament de l'objecte, signada per tots els mètodes referents a la seva persistència o CRUDs (create, read, update, delete) l'haurem de separa en classes diferents com veurem més endavant.

Encapsulació del business logic

En l'enginyeria de programari, "encapsular" és sinònim de "reutilitzar" i de "fàcil de mantenir" i aquesta és la premissa que seguirem en el nostre business logic.

POJO facade serà el pattern a seguir per a realitzat l'encapsulació. Com podreu veure més endavant en el diagrama de classes, el business lògic disposarà d'una interfície anomenada ServiceDeskService amb totes les signatures dels serveis oferts per aquesta

capa i d'una classe anomenada ServiceDeskServiceImpl amb la implementació d'aquests serveis.

Serà en aquest nivell on actuaran els serveis AOP – Managed Transactions i AOP Based Security de **Spring framework**.

Integració del business logic

La capa d'integració, que també es pot considerar part del business logic, és la que s'encarregarà d'interactuar amb la capa EIS.

De la mateixa manera que volem que el nostre programari pugui córrer en qualsevol contenidor web, també és la nostra intenció que aquest pugui utilitzar o integrar-se en qualsevol sistema de base de dades. Per aconseguir-ho utilitzarem el **patró DAO** (Data Access Object), que com veurem més endavant en el diagrama de classes de la persistència, utilitza una interfície atenent al principi de baix acoblament, i la seva implementació per cadascuna de les classes del model de domini.

El nostre business logic utilitza tecnologia d'orientació a objectes, però els sistemes de bases de dades són relacionals (o al menys els més potents), per tant en la implementació de la capa d'integració haurem de fer una transformació del model OO al relacional (**O/R Mapping**). Afortunadament el món de l'open source disposa de frameworks de persistència que ens faran la feina en un percentatge elevadíssim i concretament serà **Hibernate** el que tindrà l'honor de participar en la nostra implementació.

Hibernate suporta la majoria de sistemes de base de dades i disposa d'un *Query Language* orientat a objectes que ens facilitarà molt les cerques a la base de dades, sobre tot les que es construeixen ad-hoc en temps d'execució. Hibernate inclou també un sistema de transaccions i un con control de concurrència mitjançant la utilització de **columnes identificatives de la versió** de cada tupla de la base de dades.

Atenent al principi de baix acoblament farem treballar Hibernate, que ja de per si és poc invasiu, sota l' **API** de java **JPA**, i, per tal de facilitar el desenvolupament i la llegibilitat del codi, utilitzarem el sistema declaratiu **d'annotations** JPA en contra d'arxius de mapeig XML.

Sistema de transaccions declaratiu del business logic

Crec que sempre que es pugui s'ha d'optar per un sistema declaratiu en contra d'un programàtic. En el cas que ens ocupa no es necessari un alt nivell de granularitat en les transaccions i no existeixen excepcions si no més aviat un conjunt de normes assentades, per tant el sistema declaratiu és l'idoni.

Per la implementació, en aquest cas, utilitzarem els serveis AOP (Aspect Oriented Programming) oferts per Spring Framework mitjançant la seva classe JPATransactionManager, que tal com diu el seu nom segueixen la **API JPA**. El fet de que la nostra aplicació utilitzi un POJO façade ens facilita molt el sistema de transaccions, doncs a nivell pràctic, l'únic que haurem de fer és, a cada servei del POJO que tingui alguna relació amb la persistència de dades, incloure una annotation

@Transactional i deixar que Spring realitzi els commits o els rollbacks en funció de les excepcions rebudes.

S'ha de remarcar però, un problema inherent a les tecnologies utilitzades. La transformació del model relacional al model jeràrquic d'objectes mitjançant Hibernate ORM no es pot fer en bloc, es obvi que no tota la base de dades es pot carregar en memòria, i, per tant s'utilitza una tècnica de "lazy load" o també anomenada "load on demand", que consisteix en llegir les dades conforme es necessiten, és a dir conforme s'utilitzen els getters dels business objects i, si l'accés a les dades marcades com a retardades es fa fora de la transacció, es produeix inevitablement un excepció.

Aquest problema te solució dins d'una aplicació monolítica com pot ser una aplicació web senzilla, doncs es pot obrir una transacció a nivell de request mitjançant la utilització d'un filtre i per tant la transacció no finalitza fins que no s'han retornat totes les dades a la interfície d'usuari. Però, en el cas que ens ocupa, en una aplicació on tenim un business logic, no existeix una solució consistent o si més no que no porti la meva feina fora del domini d'aquest treball de fi de carrera, per tant la decisió que he pres és la de suprimir el qualificatiu de lazy load a determinades dades que podien causar aquest tipus de problema, provocant com a dany col·lateral una disminució del rendiment de l'aplicació si el volum esdevé gran.

3. Capa de Base de Dades

La responsabilitat d'aquest capa és la d'oferir els serveis de persistència demandats pel business logic així com un nivell complementari en la validació de la integritat de les dades.

La nostra feina en aquesta part serà molt minsa, ens limitarem a escollir un sistema RDBMS del mercat Open Source que compleixi els estàndards i que ens doni facilitats en la fase de desenvolupament, donat que la fase de producció, aquest sistema de base de dades podrà ser substituït per qualsevol altre tal com s'especifica en els requeriments no funcionals.

La nostra elecció serà PostgreSQL i estarà justificada pel fet de que disposa d'una interfície gràfica per administrar-la que ens facilitarà força la feina i també pel fet de que ja disposem de coneixements bàsics d'aquest sistema de base de dades.

4. Capa de presentació.

Les responsabilitats principals de la capa de presentació són:

- Mantenir l'estat de la sessió.
- Presentar les dades, recollir els inputs de l'usuari i validar-ne una part.
- Establir el sistema de navegació entre pàgines del programari.
- Delegar les entrades i sortides de dades a la capa de lògica de negoci.
- Implementar el sistema de seguretat.

Com en les demés capes, dins de les amplies possibilitats que ens ofereix l'enginyeria de programari i el mercat, hem pres una sèrie de decisions:

1. Els mateixos raonaments que ens han portat a dividir la nostre aplicació en 3 capes, ens porten ara a dividir la presentació en tres elements, tal com marca el patró **MVC** (Model-View-Controller) . Martin Fowler explica el patró de la següent manera: "La petició arriba al controlador, que la tramet al model apropiat del business logic. El model obté les dades de la base de dades i retorna el control al controlador, qui alhora mira el resultat i decideix quina és la vista adient per mostrar la resposta..". Més endavant podrem veure un diagrama de seqüència de una petició del Service Desk que aplica aquest patró.

2. L'evolució tecnològica que han sofert els navegadors Web, incorporant tecnologies enfocades a continguts dinàmics (xhtml, css, javascript, AJAX) i la bona acollida que ha sofert **RIA** (Rich Internet Applications) per part dels usuaris ens obliga a bellugar-nos cap a **JSF (Java Server Faces)**, una tecnologia emergent i de molt futur en el desenvolupament java. JSF ens ofereix:

    1. Desenvolupament similar a una aplicació orientada a objectes tipus Swing.

    2. **Gestió automàtica de les dades** i de les conversions mitjançant uns elements anomenats Backing-beans, fortament tipats i independents dels components UI.

    3. Model de components de interfície d'usuari extensibles i d'alta funcionalitat. Això fa que podem trobar al mercat open source molta **diversitat de components amb alta funcionalitat** (RichFaces, IceFaces, Oracle ADF...).

    4. Components que es renderitzen d'acord amb el tipus de navegador que té el client, oferint al programari **independència amb respecte al client** que realitza la petició.

    Dins de JSF hem escollit la implementació de **JBoss RichFaces** pel seu nivell de funcionalitat, la facilitat d'us, la documentació existent i pel seu look and feel.

    També incorporarem el framework **Facelets** que ens permetrà la utilització de plantilles XHTML que, d'acord amb l'enginyeria d'usabilitat, ens ajudaran a estandarditzar totes les pagines del programari i alhora ens permetrà treballar amb JSF de manera nativa, sense haver d'utilitzar pàgines JSP.

3. **Seguretat**. JSF no disposa d'un sistema de seguretat, per tant el que farem serà incorporar un **filtre servlet** anomenat ServiceDeskSecurityFilter que, de manera desacoblada i transparent a la resta de programari, gestioni tant l'autentificació dels usuaris com les comprovacions corresponents a les autoritzacions associades als rols dels usuaris. Aquest filtre que estarà recolzat pel managed bean SecurityBean.java realitzarà aquestes dues tasques:

    - Autentificació: Redireccionar la sol·licitud entrant a la pàgina de Login en cas de que l'usuari no estigui autenticat.

    - Autorització: Comprovar el o els rols assignats a l'usuari logat i autoritzar o denegar l'accés a la pàgina sol·licitada.

En cas d'error d'autorització mostrarà la següent pantalla:

Figura 34. Pantalla recurs no autoritzat

4. **Redirecció de la pantalla de login**. L'aplicació incorporarà un mecanisme de redirecció de login semblant al d'ASP.NET. Quan l'usuari sol·licita una pàgina i aquest no ha estat encara identificat, l'aplicació el redirecciona a la pàgina de login tot incorporant a la comanda HTTP GET el paràmetre de la pàgina que ha sol·licitat originalment, i , d'aquesta manera la pàgina de login pot redireccionar a l'usuari una altra vegada a la pàgina sol·licitada inicialment un cop aquest ha introduït correctament les seves credencials.

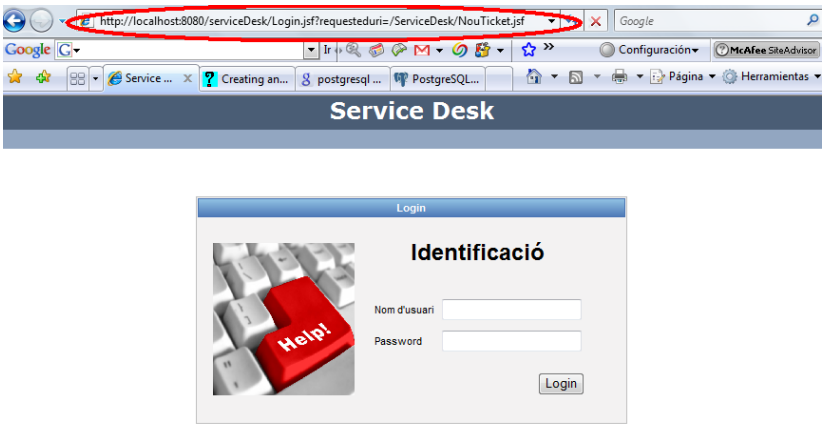


Figura 35. Sistema de redirecció del Login

5. **Impedir la cache dels navegadors d'Internet.** Tots sabem que un dels enemics de les aplicacions empresarials en entorn Web és la "caché" de pàgines que realitzen per defecte els navegadors que mitjançant la utilització de la comanda "enrere" per part de l'usuari pot portar a pàgines de dades que ja han estat prèviament addicionades i que ara poden ser addicionades de nou per accident. En aquest sentit el que farem serà incorporar un phase-listener (NoCachePhaseListener.java) que no és més que un escoltador del lifecycle de JSF, i, que en la fase de RenderResponse incorpora una capçalera HTTP amb les instruccions adients per fer expirar la pàgina immediatament i impedir la caché dels navegadors.

5. Arquitectura OLAP

Per portar a terme la funcionalitat OLAP sol·licitada en l'anàlisi de requeriments utilitzarem una arquitectura diferent obligada pels components emprats.

Dins del mercat de l'open source no existeix gaire varietat de solucions que donin un nivell de funcionalitat OLAP acceptable a nivell Professional i l'elecció ha estat quasi be obligada, **JPivot** a nivell de presentació i **Mondrian** com a servidor OLAP

multidimensional seran els components a incloure, ambdós avalats per solucions open source de Business Intelligence com poden ser Pentaho JasperSoft o SpagoBI que tenen una bona posició en el mercat de BI.

En la següent imatge podem veure l'arquitectura de la solució que aplicarem:

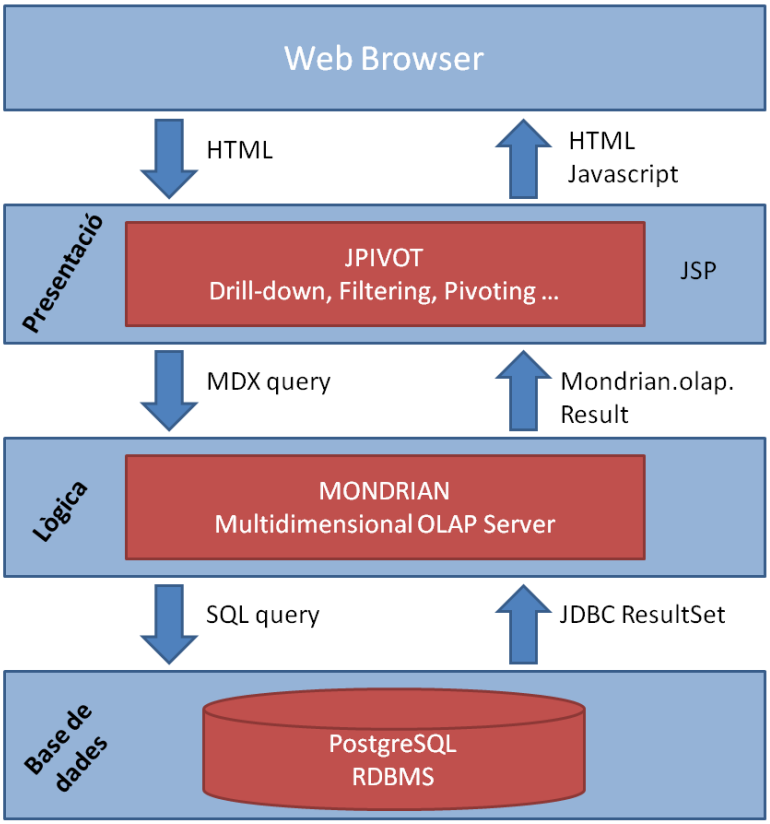


Figura 36. Arquitectura OLAP

6. Upload de documents

L'upload de documents és una de les operacions que podem considerar costoses des del punt de vista tècnic. En aquest sentit incorporarem un control de RichFaces que aplicant la tecnologia AJAX permetrà a l'usuari realitzar varis uploads de documents al mateix temps des de la mateixa pantalla d'entrada de dades, sense la necessitat d'obrir una nova pàgina. Aquest és una imatge del component treballant dins de la pàgina EdicioProgres.xhtml:

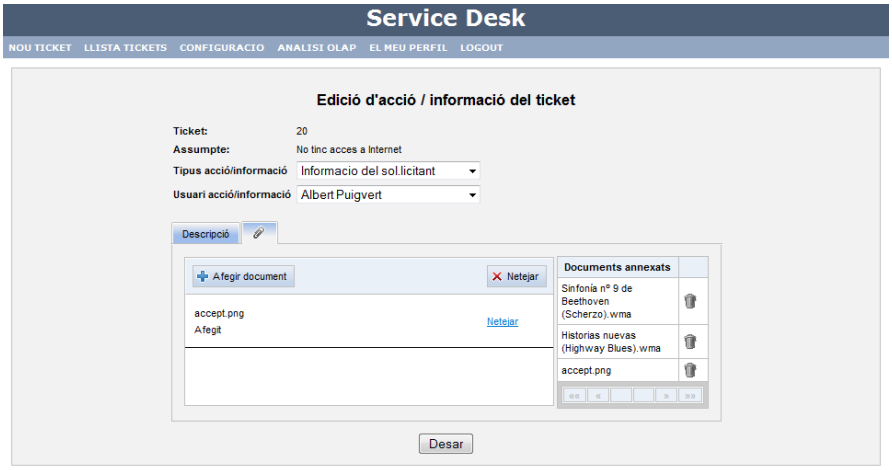


Figura 37. Upload de Documents

Si bé l'upload de documents es fa a nivell de request, doncs perfectament podem obrir dues pestanyes de FireFox (mateixa sessió) i fer uploads de fitxers al mateix temps sense que es produeixi cap barreja de dades, si que és veritat que el managed bean utilitzat serà a nivell de sessió (FileUpload.java). L'objectiu del bean de sessió i no de request ve determinat per un bug important de JSF en el seu lifecycle i pel fet de que el bean de sessió, mitjançant l'annotation @PreDestroy, ens donarà la possibilitat de netejar tots els arxius temporals utilitzats per fer els uploads, abans de que la sessió es destrueixi.

S'ha optat per l'opció de **desar els documents en la base de dades** en contra de fer-ho en el filesystem del servidor. Els motius principals han estat:

Millor control de seguretat. Els documents disposen del mateix control de seguretat que el business object que els conté.

Més facilitat des del punt de vista de desenvolupament de l'aplicació i per tant menys cost.

Integració dins del sistema de transaccions i per tant evitar la possibilitat d'inconsistències entre el que diu la base de dades i el filesystem.

Facilitat de backup, doncs totes les dades es troben a la base de dades.

7. Sistema de control de concurrència optimista.

Totes les taules de la base de dades disposen d'una columna anomenada Versió del tipus integer per controlar la concurrència. Aquesta columna està mapejada amb la seva classe corresponent del business logic mitjançant l'annotation de JPA @Version. D'aquesta manera Hibernate s'encarrega de realitzar aquest control i nosaltres només haurem de salvar l'obstacle que representa el treballar en una aplicació Web pel fet de ser stateless i incloure un control inputHidden en els formularis per tal de conservar la versió inicial de les dades a modificar durant els diferents round trips de la informació. El resultat que rep l'usuari quan el sistema detecta problema de concurrència és el següent:

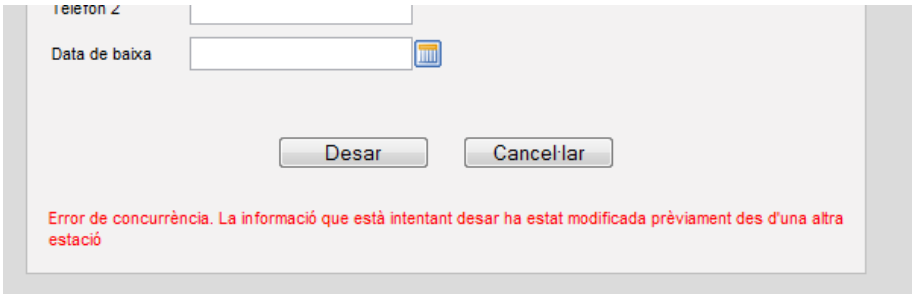


Figura 38. Missatge control concurrència

8. Sistema d'assignació automàtica de claus

Dins de les diferents modalitats que ofereix Hibernate, he seleccionat la que podem anomenar "Assignació basada en taula de claus" i que consisteix en disposar d'una taula a la base de dades que contindrà un registre am el valor de la propera clau a assignar per cadascuna de les classes del Domain Model que necessitin aquest sistema.

Hibernate necessitarà el corresponent mapeig amb el business objects mitjançant l'annotation de JPA @GeneratedValue i la configuració del fitxer orm.xml amb la definició de cadascuna de les claus

5.3 Diagrames de disseny del business logic.

Domain Model

Tal com hem indicat a l'apartat d'arquitectura, el nostre business logic estarà fonamentat en el pattern "**Domain Model**". Aquest és el diagrama estàtic de classes que el composen:

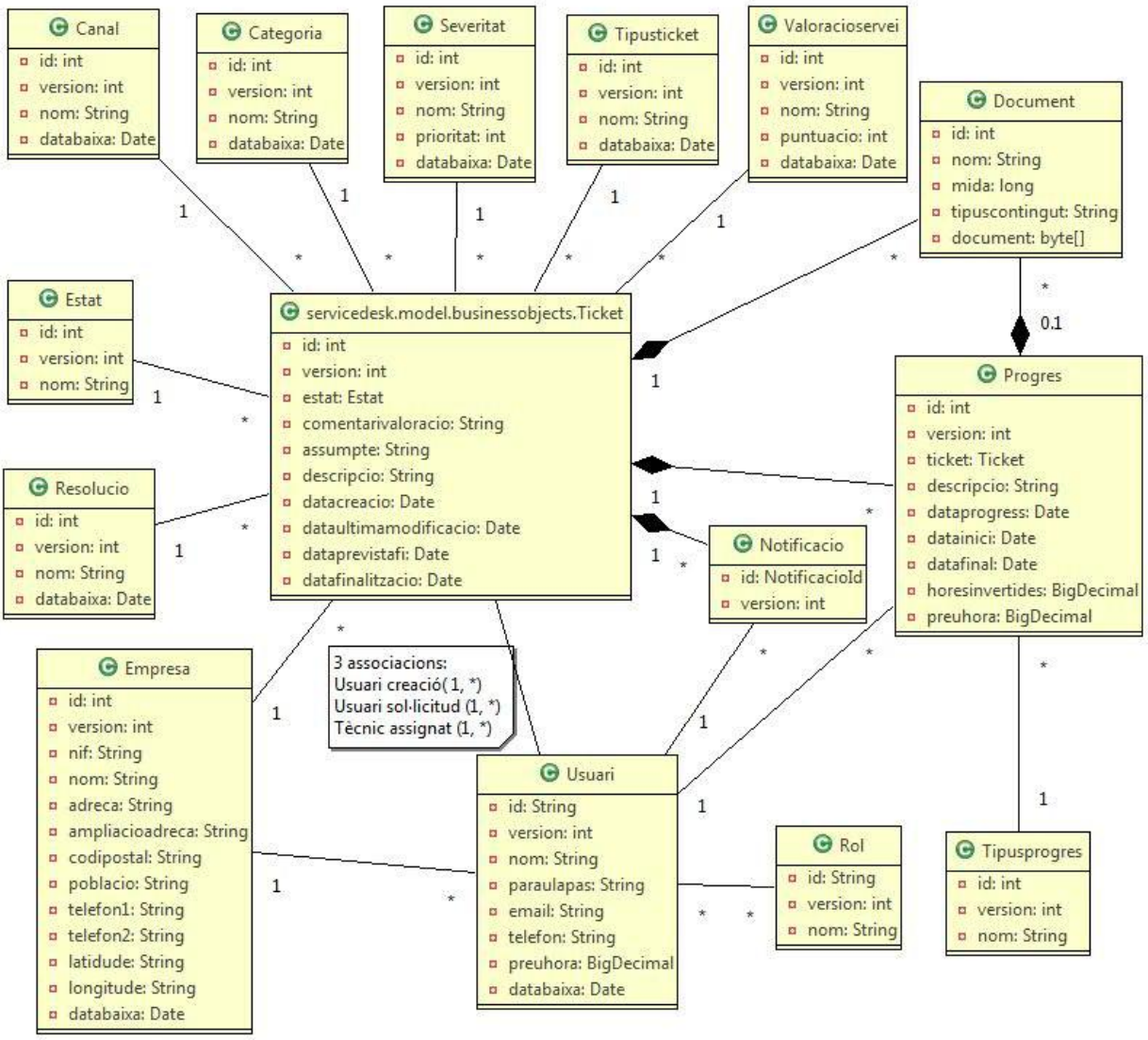


Figura 39. Diagrama disseny business logic

POJO Facade

Seguidament podem veure el diagrama de classes fidels al pattern "**POJO Facade**" que tenen com a funció principal l'encapsulació del business logic. (En el diagrama no s'han inclòs per problemes d'espai, les relacions amb el diagrama anterior)

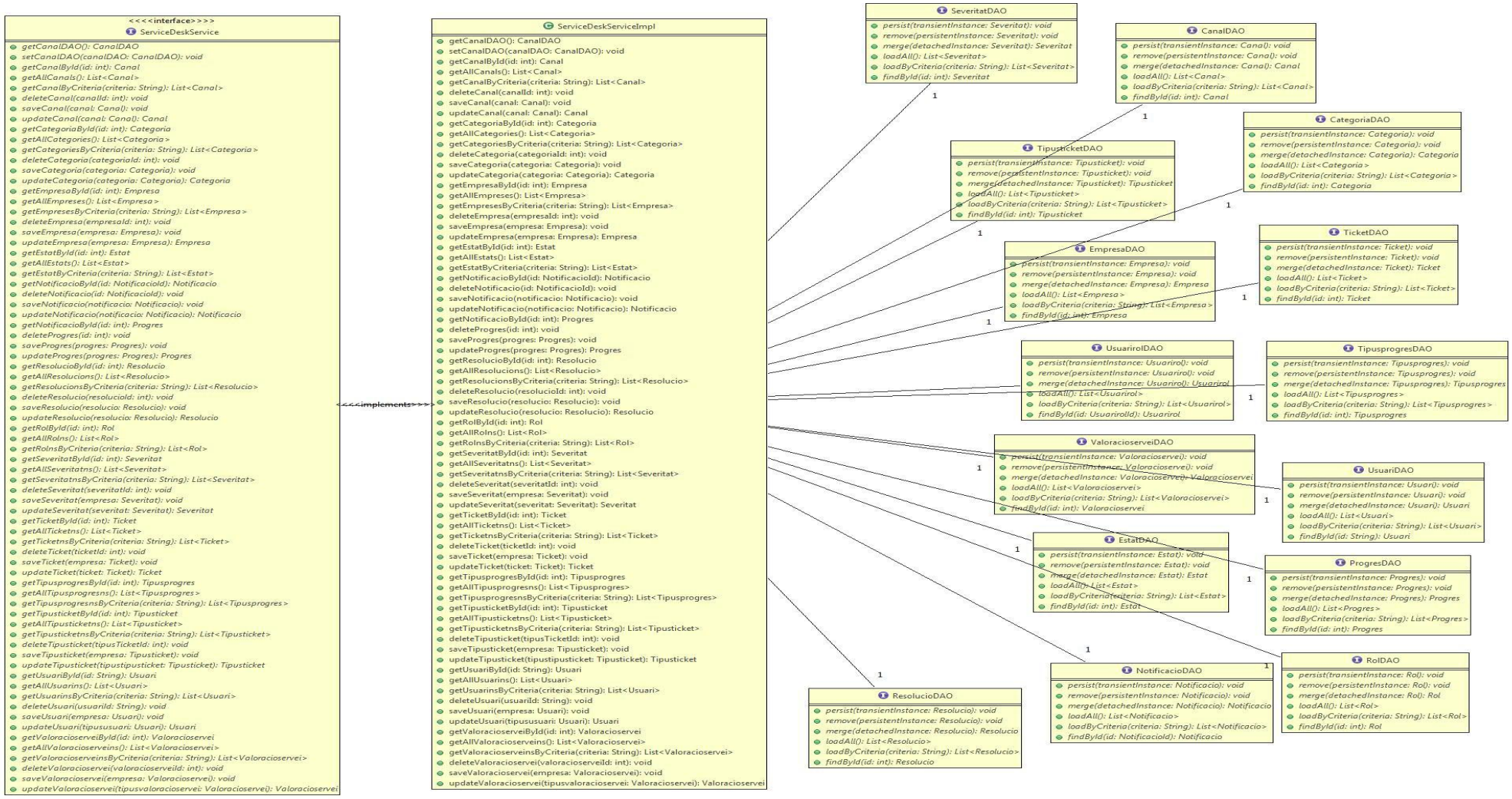


Figura 40. Diagrama POJO Facade

DAO

El següent diagrama intenta mostrar, si bé hi ha masses classes per a ser ben representades, la representació del **pattern DAO**. Cada classe del Domain Model (no representada en aquest diagrama) té associada una interfície DAO per a proporcionar el nivell de desacoblament, i cada interfície DAO és implementada, en aquest cas per una classe DAO que utilitza el framework de persistència Hibernate.

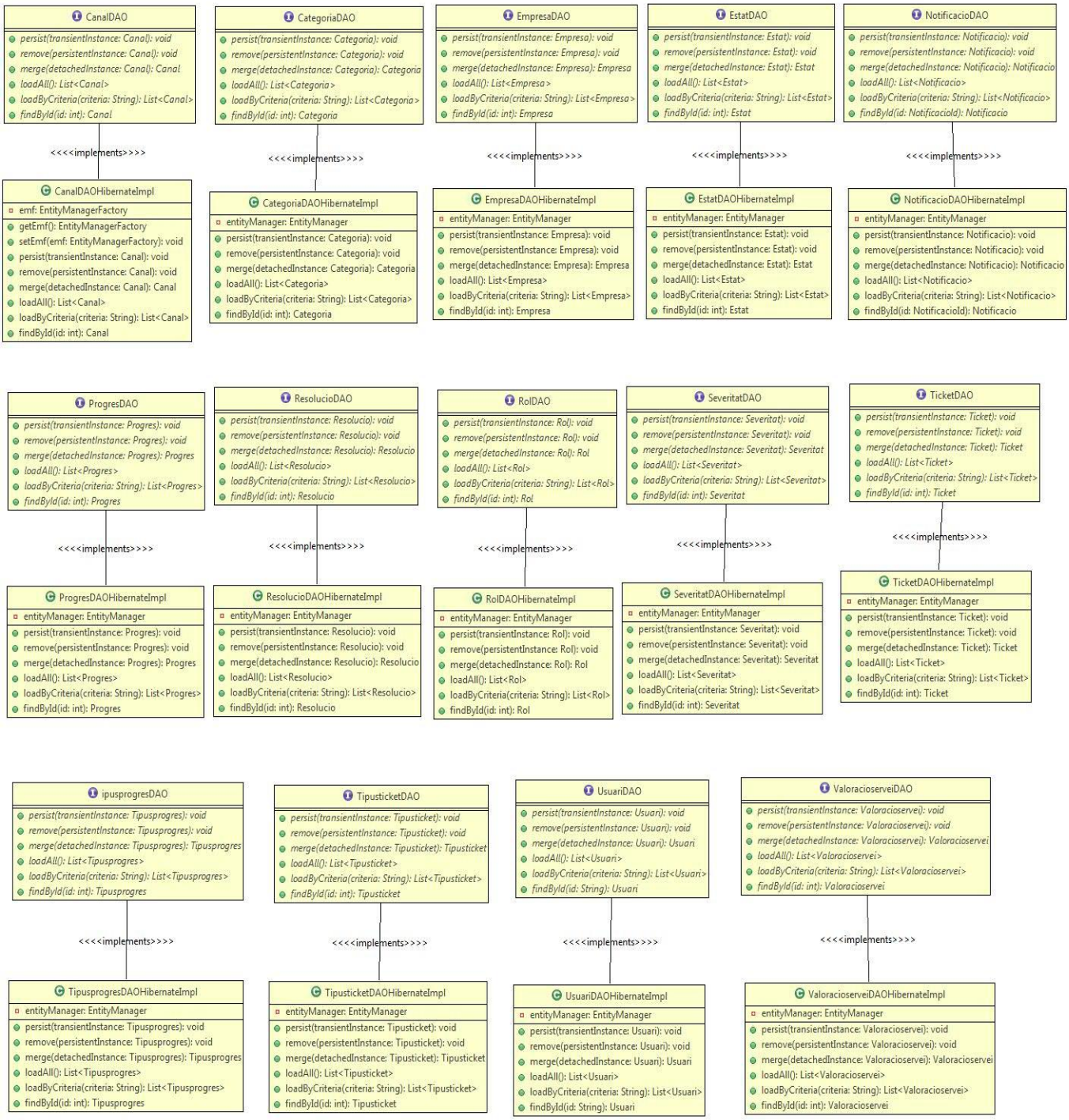


Figura 41. Diagrama DAO

Validacions

De la mateixa manera que el patró DAO proveeix d'una interfície i una classe que implementa la persistència per cada classe del domain model, també s'ha creat una interfície i una classe que implementa la validació de cada unitat de treball continuant amb el principi de desacoblament.

El concepte DRY (dont repeat yoursealf) ens fa pensar en un únic sistema de validació de dades portat a terme pel business logic i que es pugui propagar per la capa de presentació i arribar a l'usuari sense necessitat de validacions redundants. En aquest sentit hem realitzat el següent disseny:

- S'ha creat la classe ValidationError amb dos atributs, el nom de l'atribut erroni i l'error textual.
- Per cada classe POJO s'ha creat una interfície amb els mètodes de validació i la seva corresponent implementació. Aquests mètodes retornen un List<ValidationError> amb tota la col·lecció de les broken rules.
- De manera **preventiva** tots els mètodes validadors poder ser cridats des de la presentació, i de fet aquesta ho fa, per tal de retornar els possibles errors a l'usuari.
- De manera **defensiva** tots els CRUDs (create, read, update, delete) del patró DAO abans d'executar la gravació a la base de dades realitzen abans una crida als validadors i en cas d'existència d'error llencen una excepció amb un missatge que és el resultat de la concatenació del errors.

Diagrama estàtic del sistema de validacions

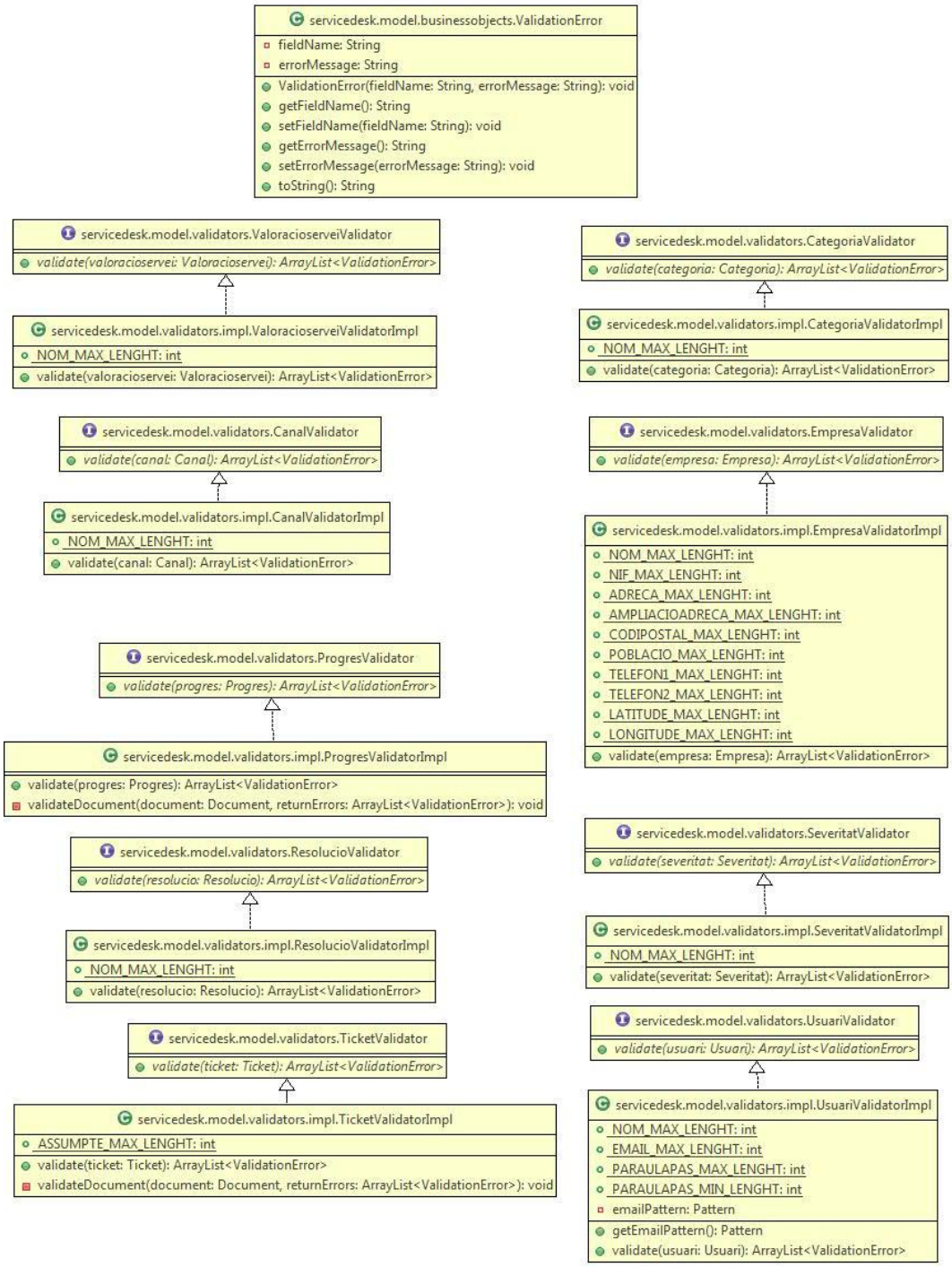


Figura 42. Diagrama Validacions

5.4 Diagrames de disseny de la persistència

El diagrama de model de dades (E/R) passa a tenir una importància secundària en les aplicacions de n-capes on el business logic és l'encarregat tant de realitzar les validacions com d'establir les associacions, relacions i les cardinalitats entre objectes. D'aquesta manera només presentarem una versió light del model E/R

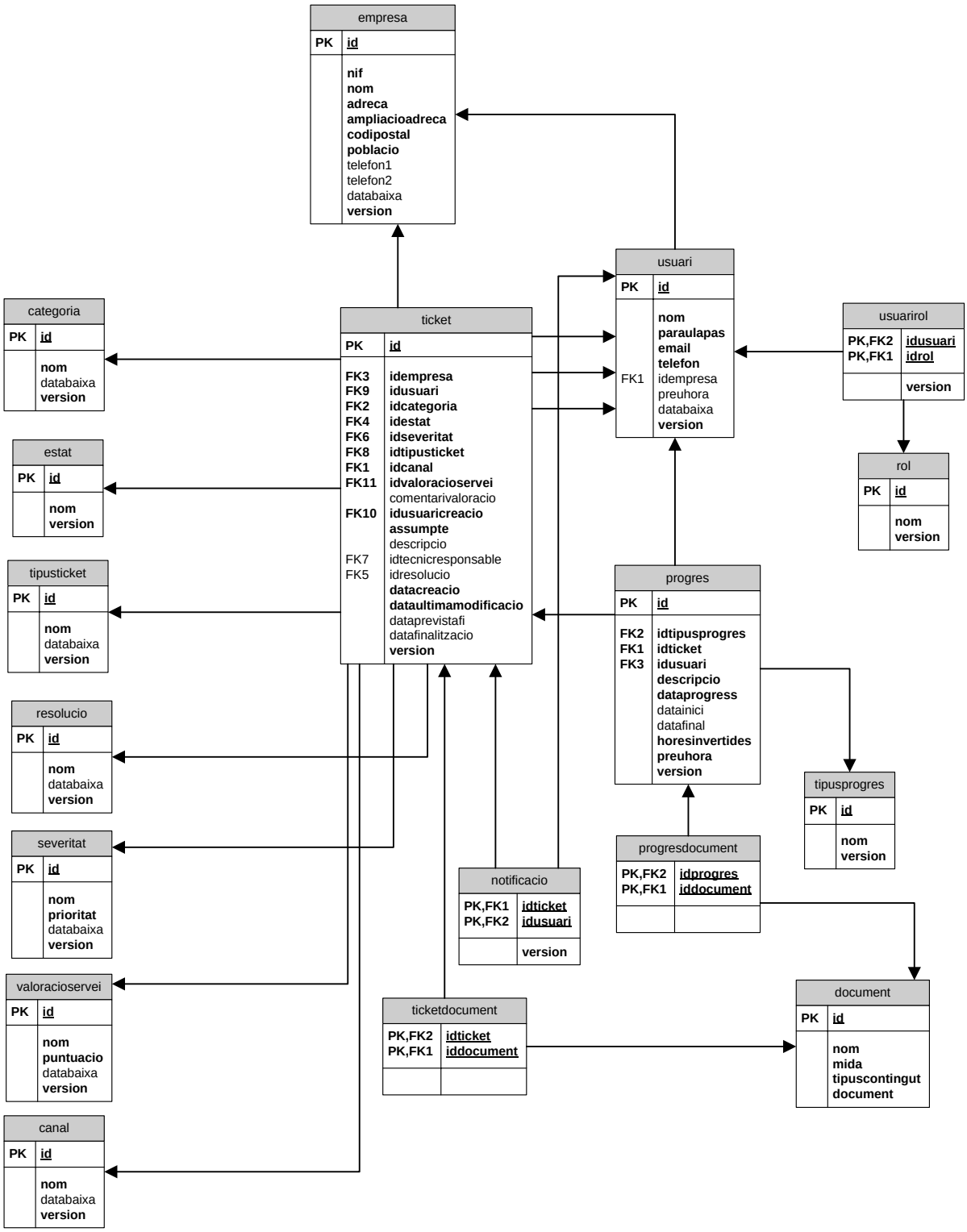


Figura 43. Diagrama Entitat Relació

5.5 Diagrames de disseny de la presentació

JSF encaixa perfectament en l'arquitectura de presentació basada en **MVC** oferint una diferenciació molt clara entre els tres elements vista, controlador i model.

Cada **vista** està definida en un arxiu amb notació XHTML on s'utilitzen les tags pròpies de la implementació, en el nostre cas RichFaces i l'Expression Language per comunicar-se amb el managed bean. Els managed bean contenen accions que són una extensió de la capa del **controlador** FacesServlet, que és un **front contoller**, i delegen les peticions de l'usuari a la lògica de negoci o **model**.

El següent esquema ens donarà una idea molt clara de la implementació a alt nivell del patró **MVC** en la nostra aplicació:

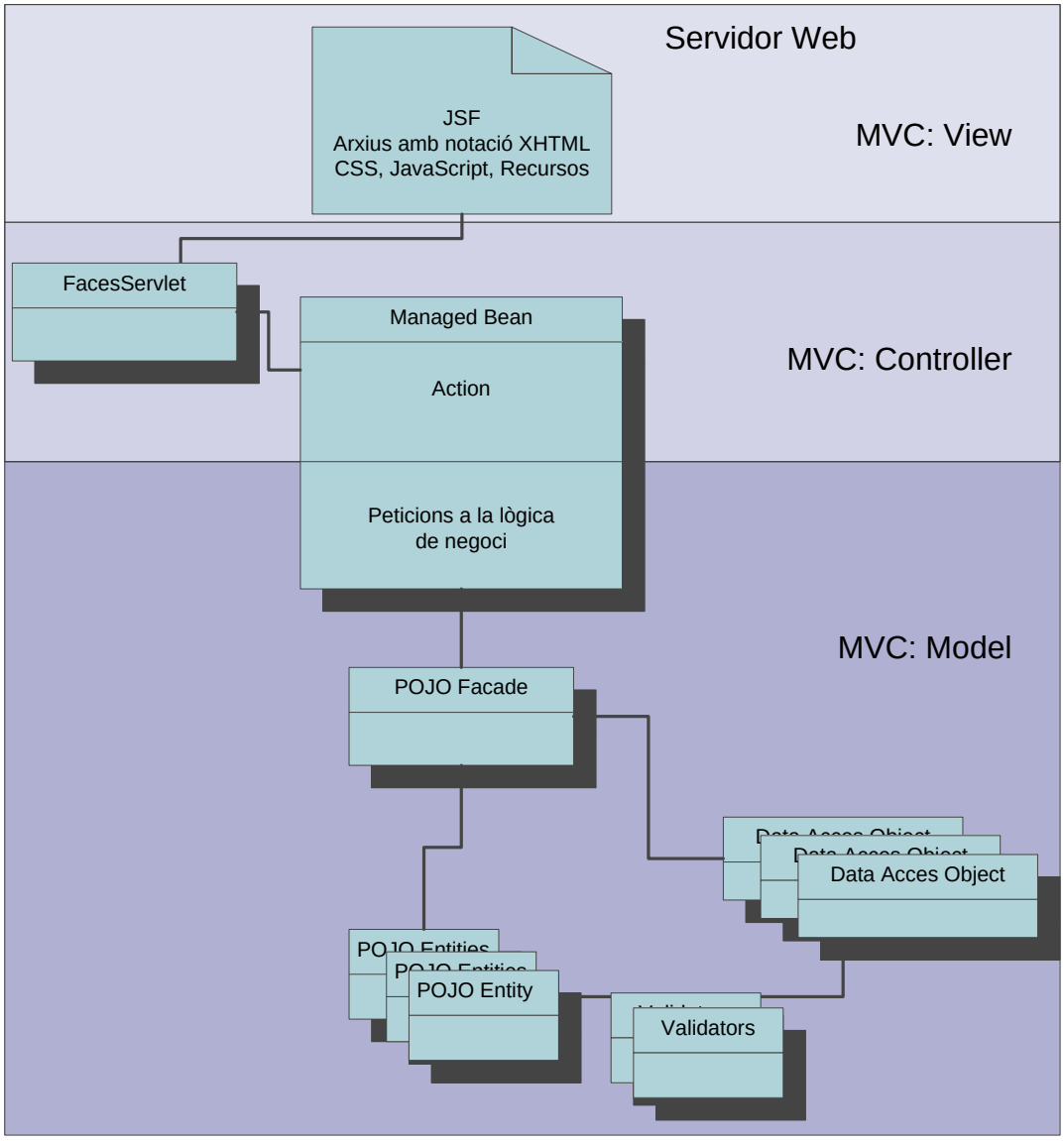


Figura 44. Diagrama MVC

La implementació a un nivell més detallat la podem trobar en el següent diagrama estàtic de classes:

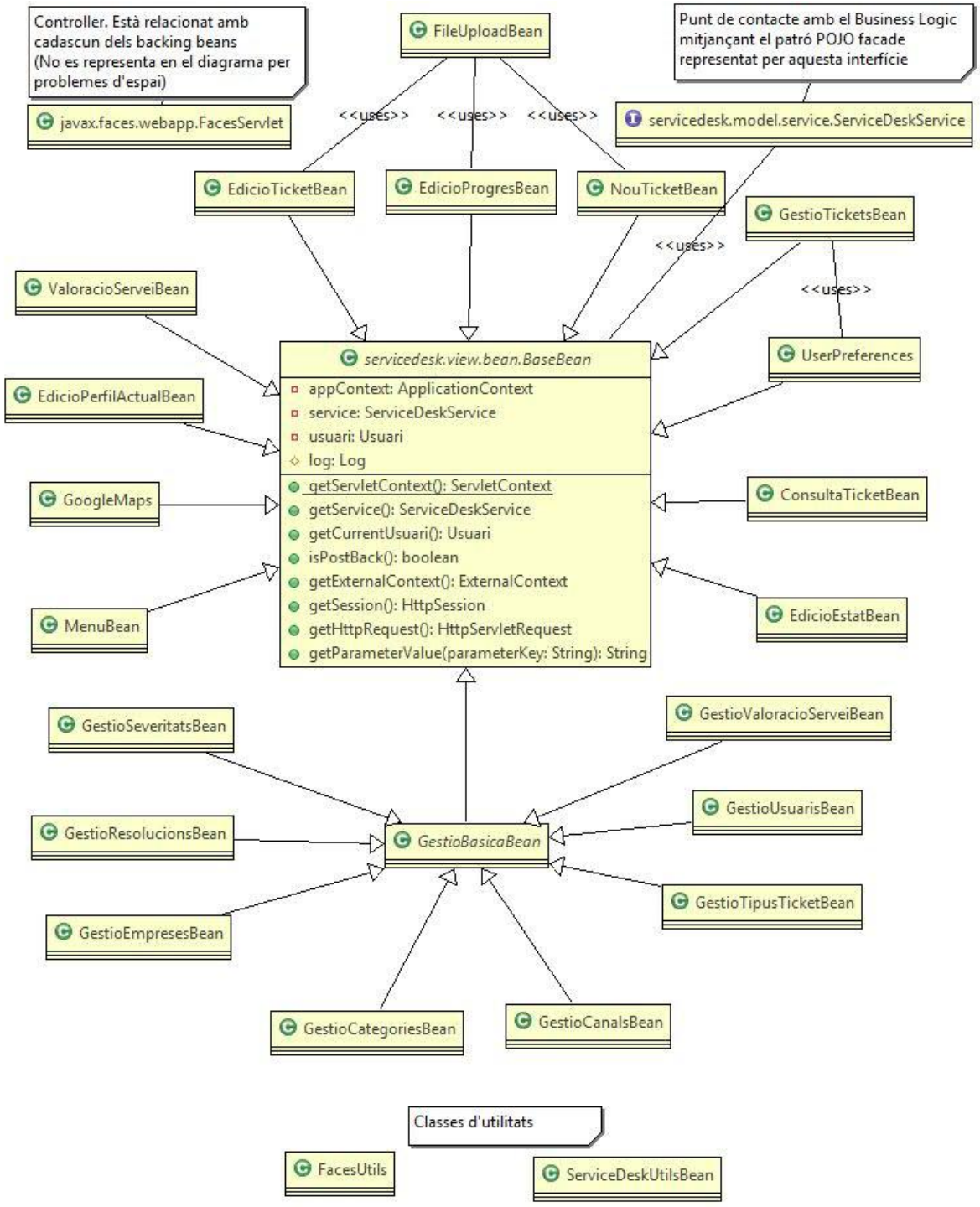


Figura 45. Diagrama classes presentació

Les vistes, que no tenen el qualificatiu de classe no es representen en aquest diagrama, si bé hem de pensar que en un 99% dels casos a cada managed bean li correspon una vista i viceversa. Seguint la filosofia JSF, les vistes estaran codificades en arxius xhtml.

En el diagrama podem veure que hi ha una classe base “BaseBean”, de la qual hereta tota la resta de managed beans, que incorpora els serveis comuns i com a més important una instància de la interfície ServiceDeskService (POJO facade) que és la que dona accés als managed beans a tots els serveis del business logic.

Per una altra banda podem també veure l'existència d'un front controller

També ens trobem amb la classe FacesServlet que és la que actuarà de Front Controller amb les següents responsabilitats:

- Processar la petició.
- Enllaçar o mapejar la vista xhtml i el seu managed bean corresponent.
- Executar les accions corresponents del managed bean en funció de les operacions realitzades per l'usuari.
- Gestionar el flow de pàgines, és a dir el sistema de navegació.

Com hem dit en un apartat anterior, al no treballar dins un servidor d'aplicacions, nosaltres serem els responsables de la seguretat i això ho portarem a terme mitjançant el pattern **Intercepting Filter**. Aquest pattern ens permet introduir un filtre d'autenticació i autorització dins de la cadena de filtres web.

Aquest és el diagrama estàtic de classes per a la implementació del sistema de seguretat:

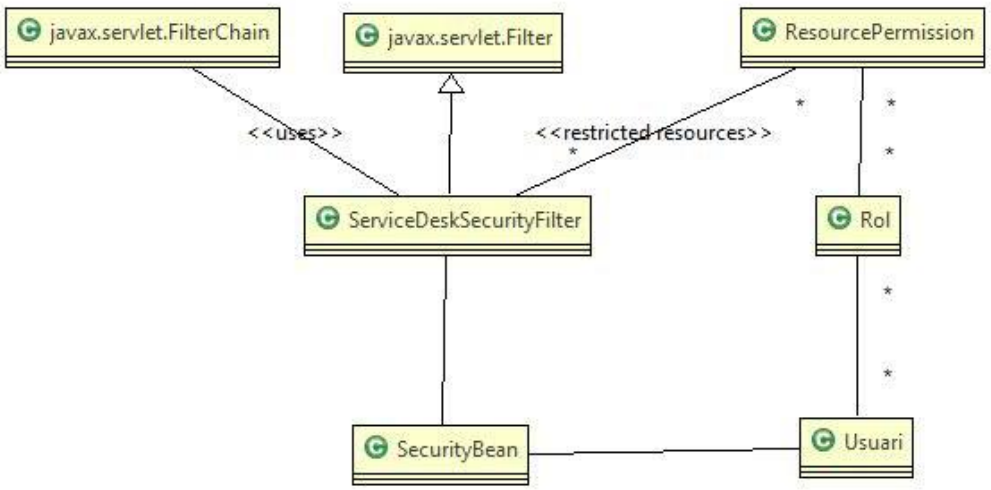


Figura 46. Diagrama classes filtre de seguretat

5.6 Tot connectat.

La intenció d'aquest apartat és només mostrar les relacions entre els components de les diferents capes i com treballen de manera conjunta. En aquest sentit hem construït un diagrama de paquets i dos diagrames de seqüència de dos casos d'us significatius:

Diagrama de paquets.

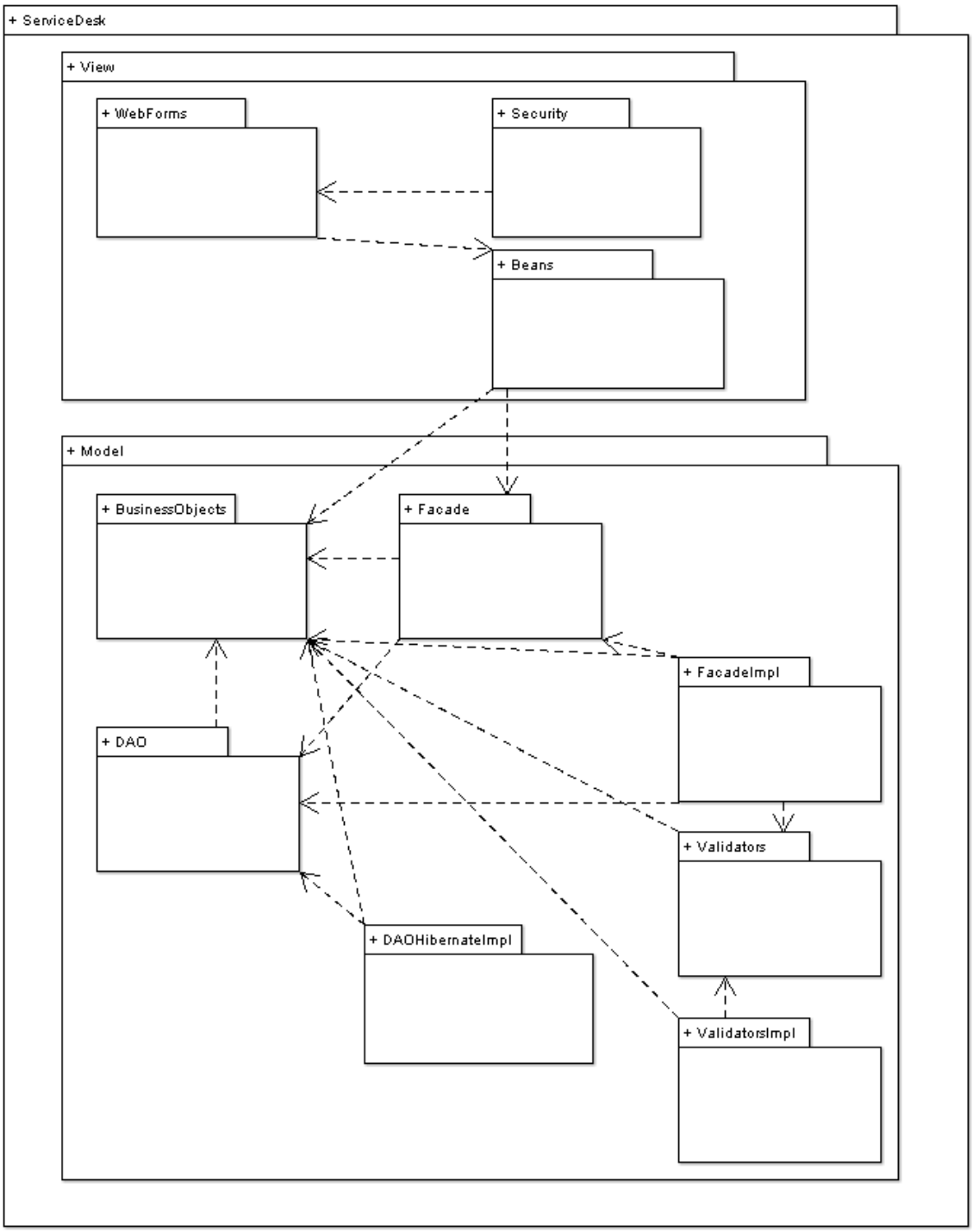


Figura 47. Diagrama de paquets

Diagrama de seqüència pel cas d'us: Nou ticket.

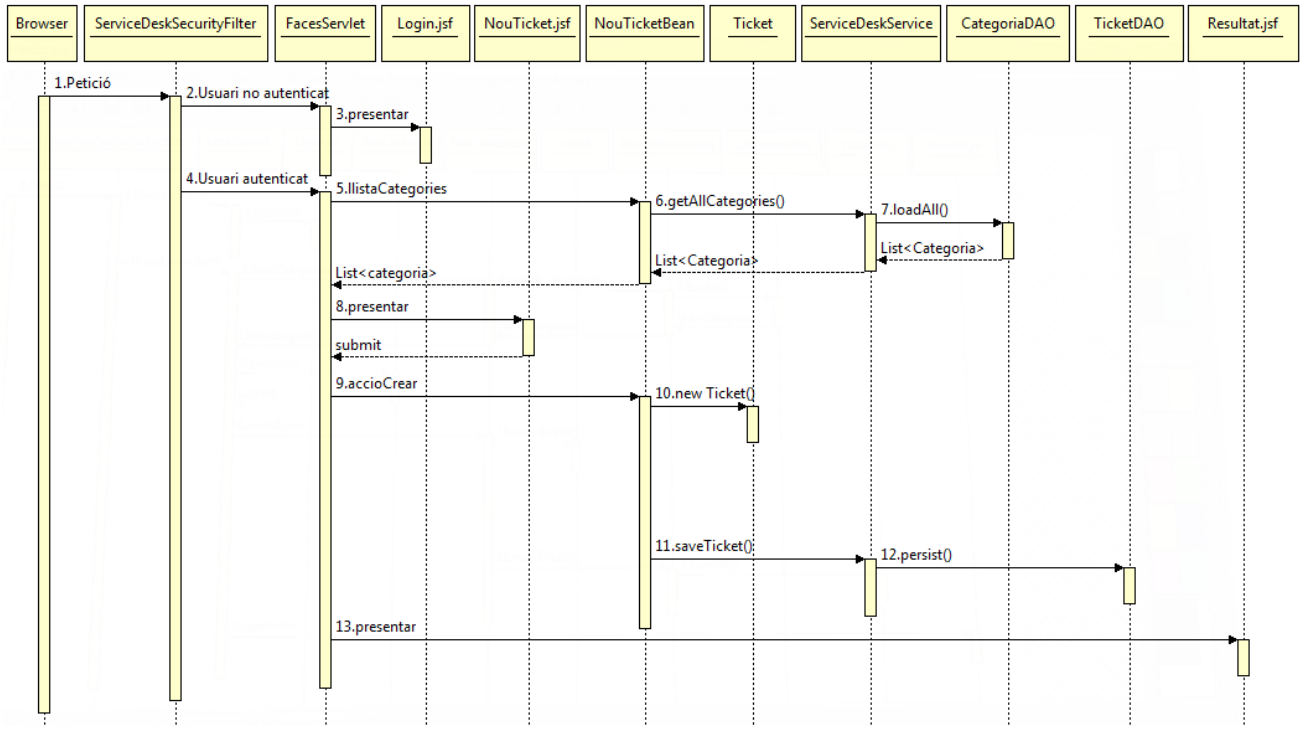


Figura 48. Diagrama de seqüència Nou Ticket

Diagrama de seqüència pel cas d'us: Edició del meu perfil d'usuari

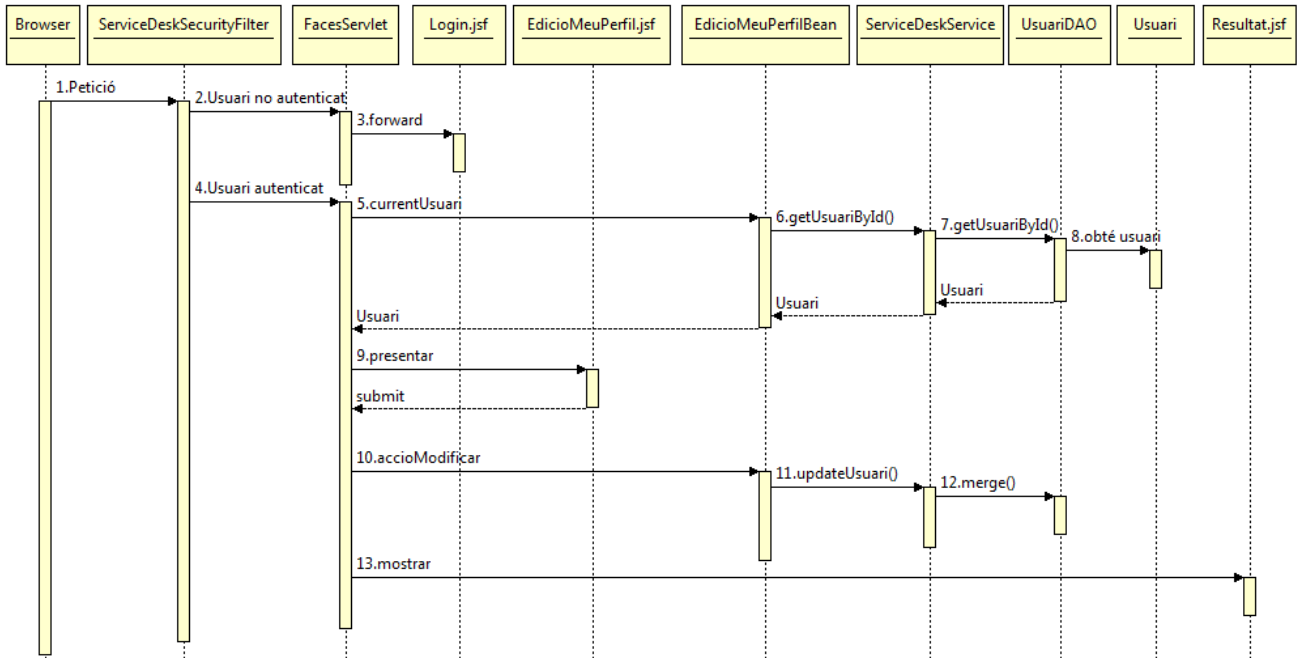


Figura 49. Diagrama de seqüència Edició Perfil

6. IMPLEMENTACIÓ.

6.1 Resum del programari utilitzat a la implementació

Per implementar el sistema hem utilitzat el següent programari open source:

- Plataforma Java. JDK 6.0
- Spring Framework 2.5
- Hibernate ORM 3.2.6
- JSF Mojarra 1.2
- Facelets 1.1
- JSF RichFaces 3.2
- SGBD PostgreSQL 8.2 i JDBC Driver
- Mondrian 3.0
- JPivot 1.8
- GMaps for JSF 1.1
- Servidor Web Apache Tomcat 6.0.
- Contenidor web Apache *Tomcat* 5.0.28 (inclòs amb JBoss)

Per realitzar el testing hem utilitzat els següents navegadors:

- Internet Explorer 7
- Mozilla Firefox 3

6.2 Entorn de desenvolupament.

- Utilitzem com a IDE, JBoss Developer Studio 1.1.0 que té com a core l'Eclipse 3.3
- Plugin AmaterasUML per confeccionar diagrames de classes i de seqüència.
- PgAdminIII per administrar la base de dades i creació de sentències SQL.
- Mondrian WorkBench 2.3 per a la creació de cubs ROLAP.

7. Valoració econòmica.

Tot els programari utilitzat en la implementació és open source i per tant no existeix cap cost sobre les llicències. L'únic cost a tenir en comte és el dels recursos humans utilitzats en el projecte.

Seguidament podem veure una valoració aproximada:

Recurs	*Cost hora*
Cap de projecte	60 €
Analista	45 €
Analista Programador	35 €
Tècnic Sistemes	40 €

Activitat	***Hores***	***Recurs***	***Cost***
Gestió del projecte	40	Cap de projecte	2.400 €
Anàlisi	30	Analista	1.350 €
Disseny	30	Analista programador	1.050 €
Desenvolupament	80	Analista programador	2.800 €
Testing	16	Analista programador	560 €
Producció	8	Tècnic Sistemes	320 €
Formació usuaris	8	Analista programador	280 €
	212		**8.480 €**

Figura 50. Resum de costos

8. Conclusions.

Crec que el compliment dels objectius establerts tant a nivell personal com a nivell del projecte s'han assolit completament i fins i tot podríem dir que han estat sobrepassats, per tant la valoració global del TFC es pot considerar com molt positiva.

Per una banda l'adquisició de coneixements substancials sobre la plataforma J2EE que fins el moment desconeixia, per l'altre l'aplicació de coneixements teòrics adquirits sobre arquitectura d'aplicacions en un marc real i per últim la realització des d'un punt de vista acadèmic de les diferents fases del cicle de vida del programari m'han proporcionat una visió força estructurada i molt clara del que representa l'Enginyeria de Programari.

Fent una reflexió sobre tot el que he treballat durant aquest quadrimestre m'agradaria destacar la importància vital de la presa de decisions. Ens movem, sobre tot quan parlem de Java, en un món ple d'APIs i de gran varietat d'implementacions, al mateix temps que disposem d'un gran nombre de disposicions arquitectòniques possibles per a la nostra solució i en aquest sentit, crec que el TFC mitjançant l'obligatorietat de l'elaboració d'una memòria, ens força no tant sols a decidir sinó a decidir de manera totalment justificada, i aquest crec que és un dels aspectes més valorables.

Com a aspecte no tant positiu comentar que, des del meu punt de vista, JSF no és encara una tecnologia recomanable per ser aplicada al mon professional o empresarial, o al menys jo no la recomanaria.

L'experiència que he viscut en aquests 4 mesos ha estat especialment dura deguda per una part al gran nombre de bugs que presenta aquesta tecnologia, tant en situacions bàsiques, com en situacions on s'intenta treure un bon partit als components, i per l'altre a la falta de una documentació sòlida i professional. He hagut de descartar varis componentes de RichFaces, algun d'ells tant bàsics com pot ser un combo box, altres més complicats com uns list shuttle o el que representa google maps, pel seu mal funcionament i/o manca de funcionalitat bàsica.

També m'ha semblat del tot insegur la utilització de les vistes xhtml on l'EL (expression language) s'avalua només en temps d'execució i és extremadament permissiu en el sentit de que la seva mala utilització no produeix cap error, simplement, quan alguna cosa no es correcte la ignora, provocant comportaments no esperats i des del meu punt de vista això no és admissible avui en dia.

El sistema de navegació de JSF definit en l'arxiu de configuració faces-config.xml que a priori és una de les parts podríem dir estrella d'aquesta tecnologia, crec que te mancances importants, sobre tot pel fet de que no permet un sistema per passar paràmetres entre les vistes i això obliga a establir un altre sistema al marge d'aquest per les vistes que requereixen paràmetres.

El sistema de managed beans o backing beans a nivell d'aplicació, sessió o request és força interessant i d'alguna manera s'acosta o fins i tot intenta millorar el que en ASP.NET s'anomena "code behind", però des del meu punt de vista li manca una scope de conversació, és a dir la possibilitat de desar l'estat entre un grup de requests. He llegit que Seam ha cobert aquesta mancança en el seu framework.

He fet un càlcul aproximat del temps que he perdut en temes que podria emmarcar dins de les mancances de la tecnologia i aquest sobrepassa el 60%. He passat hores i hores cercant solucions a un munt de problemes i només he fet que trobar cents i cents de persones que tenien els mateixos problemes que jo, però cap solució. He passat moments realment angoixants.

9. Glossari.

- **AJAX** - Asynchronous JavaScript And XML. Tècnica de desenvolupament web per a construir aplicacions interactives riques. Combina quatre tecnologies: XHTML, XML, CSS i Javascript.

- **Annotations** – Una annotation, en el llenguatge de programació java és una sintaxis especial que es pot utilitzar per afegir informació de metadades al codi Font. Tant les classes, com els mètodes, com les variables i els paquets estan subjectes a la possibilitat d'incorporar anotacions. És una manera fàcil i molt intel·ligible d'afegir informació de tipus declaratiu.

- **AOP** – Aspect Oriented Programming. Complementa la POO (Programació Orientada a Objectes) permetent una adequada modularització de les aplicacions i possibilitat una millor separació de conceptes. Gràcies a AOP es poden encapsular diferents conceptes que composen una aplicació en entitats ben definides, eliminant les dependències entra cadascun dels mòduls.

- **Concurrència optimista** – És un control d'integritat de dades. Quan un usuari intenta modificar una unitat d'informació, el programari ha de determinar si aquesta ha estat modificada per algún altre usuari des del moment en que va ser llegida. En cas afirmatiu ha d'avortar la modificació.

- **CSS** – Cascading Style Sheets. Llenguatge que s'utilitza per definir la presentació d'un document que està escrit amb HTML/XHTML o XML. Formalitzat i estandarditzat pel W3C (World Wide Consortium).

- **DAO** – Data Access Object. Patró de disseny de programari que subministra una interfície entre els objectes de negoci i la seva persistència.

- **EIS** – Enterprise Information System. És un terme que s'utilitza moltes vegades per referir-se al sistema de base de dades.

- **EJB** – Enterprise Java Bean. És una de les APIs definides per java dins de l'estàndard J2EE. Defineix un objecte de negoci que compleix unes interfícies i que serà gestionat per un contenidor d'un servidor d'aplicacions J2EE.

- **Encapsulació** – Es un dels principis bàsics de la programació orientada a objectes que consisteix en la capacitat d'aïllar el funcionament intern d'un objecte del seu exterior.

- **Framework** – Marc de treball o bastiment. Proveeix al programari d'una estructura i d'un model de treball per tal de facilitar-ne el desenvolupament.

- **Java script** - És un llenguatge de programació interpretat i orientat a objectes/prototips que capacita als navegadors web per realitzar operacions a la banda del client. És una de les peces clau pel desenvolupament d'aplicacions web riques.

- **JSF** – Java Server Faces – És un framework per aplicacions java basades en web. Les seves API formen part de la definició Java EE.

- **JPA** – Java Persistence API. És una API definida en el estàndard JEE que pretén regular la manera de treballar de les eines O/R Mapping.
- **MVC** – Model View Controller. Patró d'enginyeria de programari que separa les dades d'una aplicació, de la seva vista i de la seva lògica de control.
- **MDX** – Acrònim de Multidimensional Expression. LLenguatge propietari de Microsoft per interrogar bases de dades multidimensionals (cubs OLAP) utilitzat també pel servidor open source Mondrian.
- **O/R Mapping** – Object / Relational Mapping. És una de les tècniques utilitzades per persistir i recuperar dades en format jeràrquic des de fonts de dades relacionals.
- **OLAP** – Sigles d'On-Line Analitycal Processing. És una de les solucions utilitzades en el que actualment s'anomena inteligència de negoci i que consisteix en la creació i utilització d'estructures de dades multidimensionals per poder realitzar anàlisis de forma adhoc.
- **POJO** – Plain Old Java Object. És un model d'objecte molt simple que no ha d'implementar cap interfície de manera obligatòria i que va sorgir per contrarestar la complexitat de les definicions EJB de versions anteriors a la 3.
- **XHTML** – eXtensible Hypertext Mark-up Language. Llenguatge de marques HTML que aplica una notació XML.

10. Bibliografia consultada.

MARTIN FOWLER – *Patterns of Enterprise Application Architecture*. Adsison Wesley

KITO D. MANN – *Java Server Faces In Action*. Manning

CHRIS RICHARDSON – *POJOs In Action*. Manning

Programación Java Server con J2EE Edición 1.3 – Anaya Wrox

Profesional Java JDK 6 – Anaya Wrox

CAMPDERRICH, BENET. *Enginyeria del programari; Anàlisi orientada a objectes* . UOC. (material de l'assignatura).

F.XHAFA. *Tècniques de desenvolupament de programari*. UOC (material de l'assignatura).

JOHNSON, ROB. *J2EE Design and development*. Wrox Press.

Josep M. Ganyet. Interacció Humana amb els Ordinadors. Apunts assignatura. UOC

Bruno Aranda – *Facelets Essentials* – Appress

JSF For Non Belivers – IBM articles

Tutorial integracion JSF, Spring, Hibernate
http://www.programacion.com/tutorial/jap_jsfwork/

Manual de Referencia de Spring

Manual de Referencia de Hibernate

Manual de Referencia de RichFaces

11. Annex. Guia d'instal·lació

1 Instal·lació de l'aplicació.

ServiceDesk és una aplicació Web que té com a objectiu principal el registre, seguiment i resolució d'incidències relacionades amb l'àrea informàtica.

1.1 Requeriments

1.1.1 Servidor - Hardware i sistema operatiu

L'aplicació ServiceDesk està desenvolupada integrament seguint l'estàndard J2EE (actualment anomenat Java EE) i per tant gaudeix del qualificatiu de multiplataforma. Qualsevol computador que pugi executar la màquina virtual de java JVM, serà per tant, un candidat a instal·lar l'aplicació de ServiceDesk, ja es tracti d' un ordinador portàtil, d'un de sobretaula o d'un servidor, ja sigui sota el sistema operatiu Unix, Linux, MAC o Windows.

Els requeriments de hardware dependran totalment de l'ús que es vulgui donar a l'aplicació, del volum d'informació prevista, del nivell de peticions concurrents i dels temps de resposta requerits pel negoci (SLA). En un entorn que podem considerar estàndard serà suficient amb un ordinador que disposi d'un processador d'un sol nucli de 2Ghz i amb una memòria RAM superior a 512 Mb. En qualsevol cas es tindran en compte els requeriments propis del programari de gestió de base de dades PostgreSQL i de servidor Web Apache Tomcat que són requerits per la instal·lació.

1.1.2 Servidor – Software

La plataforma J2EE requereix de la instal·lació de la màquina virtual Java JVM i d'un contenidor J2EE. L'aplicació ServiceDesk necessita ser executada sobre una versió 5 o superior de la màquina virtual i ha estat testejada sobre el contenidor Apache Tomcat 6.0. Podeu baixar-vos aquest programari utilitzant els següents links:

- Apache Tomcat
- Sun JVM

ServiceDesk utilitza per a la persistència de les dades l'open source DBMS PostgreSQL. La versió testejada pel programari ha estat la 8.2.5, per tant es recomana la instal·lació d'aquesta versió. En el següent link podeu accedir al download corresponent:

- DBMS PostgreSQL

1.1.3 Client

Es pot accedir al ServiceDesk mitjançant qualsevol navegador Web que suporti javascript. El programari ha estat testejat amb les versió 7 d'Internet Explorer i amb la versió 2 de FireFox.

1.2 Fitxers subministrats

El paquet d'instal·lació es compon dels següents tres fitxers:

Nom	Descripció
CreacioBDSD.sql	Conté les instruccions en llenguatge SQL per poder crear la base de dades i un usuari de connexió en PostgreSQL
CreacioTaulesSD.sql	Definició de l'esquema de la base de dades en llenguatge SQL.
ServiceDesk.war	Fitxer per desplegar l'aplicació en el servidor Web Tomcat. Conté també els fonts de l'aplicació.

Els fonts java es troben dins la següent estructura de paquets en el directori src del fitxer ServiceDesk.war.

Paquet	Classes
servicedesk.model	Classes que composen el que tradicionalment s'anomena business logic
servicedesk.model.businessobjects	POJOs que pertanyen al Domain Model.
servicedesk.model.dao	Interfícies que defineixen els mètodes de persistència (CRUDs). A cada classe del Domain Model li correspon una interfície DAO
servicedesk.model.dao.hibernate	Implementació de l'anterior interfície mitjançant Hibernate ORM.
servicedesk.model.service	Interfície que implementa POJO façade per mostrar un únic servei amb tots els mètodes que necessita la capa de presentació
servicedesk.model.service.impl	Implementació de l'anterior interfície. La implementació està feta mitjançant instàncies que compleixen la interfície DAO (injectades per Spring IOC) garantint un desacoblament absolut envers a la implementació dels DAO feta per Hibernate.
servicedesk.model.validators	Interfícies amb els mètodes necessaris per validar cada objecte del Domain Model.

Paquet	Classes
servicedesk.model.validators.impl	Implementació de l'anterior interfície.
servicedesk.view	Classes que composen la capa de presentació
servicedesk.view.security	Filtres web i managed beans relatius a la implementació del sistema de seguretat.
servicedesk.view.bean	Managed beans a nivell de session i de request que recolzen totes les classes .xhtml que es troben dins del directori principal.

1.3 Procés d'instal·lació

Un cop hem complert els requisits previstos, és a dir, ja tenim instal·lat:

- La màquina virtual de java JRE amb versió 5 o superior.
- Apache Tomcat versió 6 o superior
- PostgreSQL versió 8.2 o superior

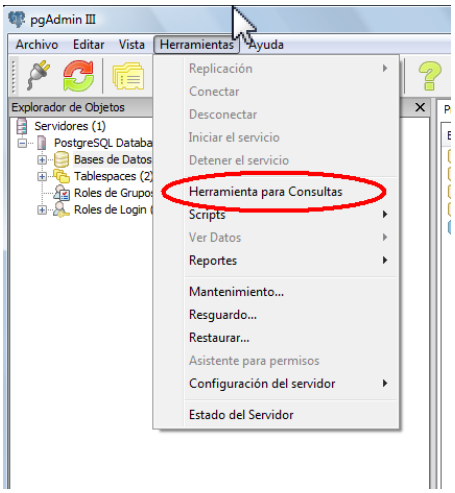
Realitzarem els següents passos:

1.3.1 Creació de la base de dades i registres de prova.

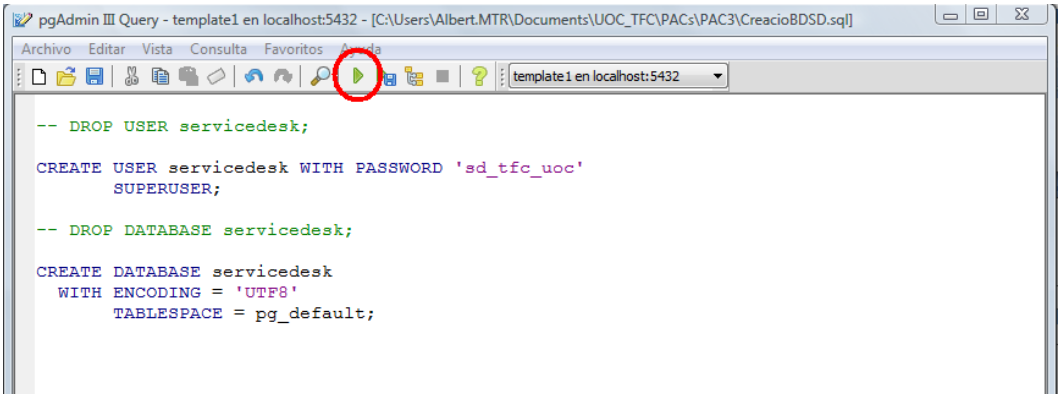
La creació de la base de dades la podrem fer utilitzant la interfície gràfica de postgreSQL o be utilitzant la línea de comandes.

a) Instal·lació utilitzant la interfície gràfica:

a. Executem l'eina d'administració de postgreSQL pgAdminIII
b. Al menú "Herramientas" seleccionem l'opció "Herramientas para Consultas"

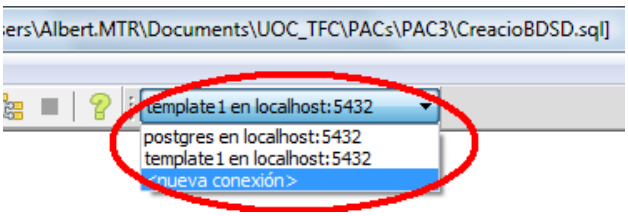


c. A la pantalla de consultes fem "Archivo" "Abrir" i anem a cercar el fitxer d'instal·lació del ServiceDesk anomenat "CreacioBDSD.sql" i premem el botó executar:

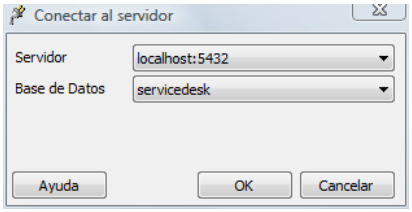


A la part inferior de la pantalla es mostrarà el resultat de l'execució.

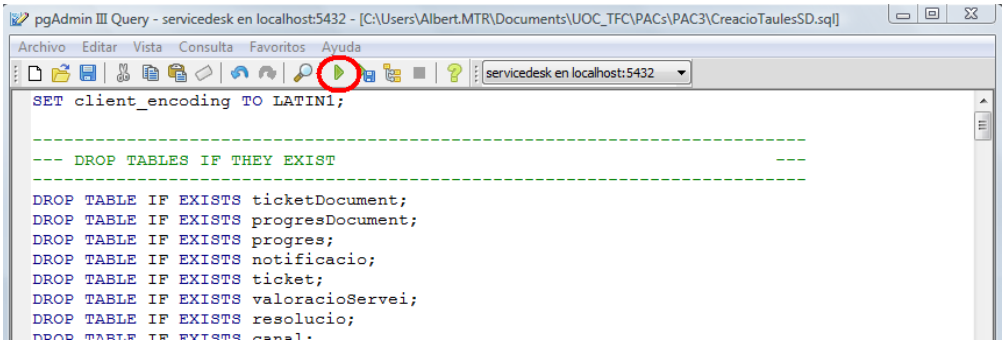
d. A la part superior de la pantalla trobarem un desplegable de connexions. Hem de seleccionar "nueva conexión".



e. A la següent pantalla seleccionarem la base de dades que acabem de crear, que s'anomena servicedesk



f. Seguidament tornarem a fer "Archivo" "Abrir" i obrirem el fitxer d'instal·lació del ServiceDesk anomenat "CreacioTaulesSD.sql" i premem el botó executar:



A la part inferior podrem veure el resultat de l'execució.

b) Instal·lació utilitzant la línea de comandes.

a. Les comandes que executarem són pròpies de postgreSQL i per tal de tenir-les disponibles o be inclourem el directori C:\Program Files\PostrgreSQL\versio\bin a la variable de cerca del sistema (PATH), o be ens situarem en aquest directori.

b. Per crear la base de dades executarem la següent comanda:

```
....\bin> createdb –h localhost –U superUser –W –E UTF8 servicedesk
```

on superUser és el nom d'usuari que hem creat a la instal·lació de postgreSQL i que té drets de súper usuari. A l'executar la comanda se'ns demanarà que introduïm el password del súper usuari.

c. Per crear l'usuari del ServiceDesk executarem:

```
....\bin> createuser –h localhost –U superUser –W –P –s –D –R servicedesk
```

on superUser és el nom d'usuari que hem creat a la instal·lació de postgreSQL i que té drets de súper usuari. A l'executar la comanda se'ns demanarà que introduïm dues vegades el password de l'usuari que estem creant i al que assignarem sd_tfc_uoc , després ens demanarà que introduïm el password del súper usuari per autoritzar l'operació.

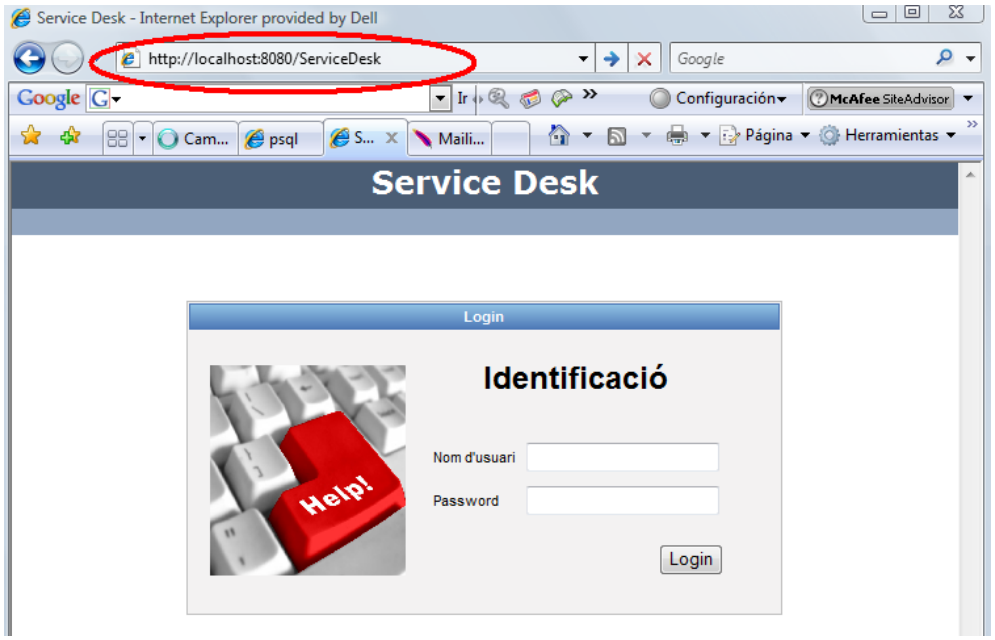
d. Per crear l'esquema de la base de dades executarem:

```
....\bin> psql –h localhost –d servicedesk –U servicedesk –f CreacioTaulesSD.sql
```

on CreacioTaluesSD.sql és un dels fitxers subministrats en la instal·lació del ServiceDesk. Si aquest fitxer no es troba en el mateix directori haurem de qualificar-lo amb el seu path corresponent. Aquesta comanda ens demanarà el password de l'usuari del servicedesk que és sd_tfc_uoc

1.3.2 Desplegament de l'aplicació al servidor Web Tomcat

Aquest pas es completarà simplement dipositant l'arxiu subministrat en la instal·lació amb el nom de "servicedesk.war" al directori webapps del nostre servidor Tomcat.
Si el servidor està aturat el desplegament el realitzarà Tomcat a l'arrancar. Si no està aturat, Tomcat analitza periòdicament aquest directori i per tant el desplegarà en uns segons.
Per provar l'aplicació obrirem en nostre Internet browser i, si accedim des de la màquina local introduirem la següent adreça:



L'aplicació te donats d'alta varis usuaris per representar cadascun dels rols, però s'ha creat un usuari amb nom "test" i password "test" que té assignats tots els rols per tal de poder probar totes les funcionalitats.

Si volem probar cadascun dels rols ho podem fer mitjançant els usuaris:

Usuari	Password	Role
test	test	Tots els rols assignats
admin	admin	administrador
coordinador	coordinador	coordinador
tecnic	tecnic	tecnic
responsable	responsable	responsable
apuigvert	apuigvert	sol·licitant