Índex

0. Resum 3

1. Introducció 4

1.1 Objectiu 4

1.2 Motivació 5

1.3 Estat de l’art 5

1.4 Planificació temporal 7

1.5 Resum de continguts de la memòria 7

2. Tecnologies implicades 8

2.1 BackEnd 8

2.1.1 Spring Framework 8

2.1.2 Spring Boot 9

2.1.3 Resoldre el problema del codi *Boilerplate* 10

2.1.4 Maven 12

2.2 Persistència 13

2.2.1 Hibernate 13

2.2.2 PostgreSQL 13

2.3 FrontEnd 13

2.3.1 Thymeleaf 13

2.3.2 Twitter Bootstrap 14

2.3.3 jQuery 15

2.4 Maquetació 16

2.4.1 Balsamiq Mockups 16

2.5 Model de dades 17

2.5.1 Project Mogwai 17

2.6 Integritat i qualitat del codi 18

2.6.1 Jenkins 18

2.6.2 SonarQube 19

2.7 Control de versions 20

2.7.1 GIT 20

3. Especificació de requisits 21

3.1 Introducció 21

3.2 Àmbit 21

3.3 Descripció general 21

4. Anàlisi 24

4.1 Casos d’ús 24

4.1.1 Catàleg 24

4.1.2 Alta de servei 25

4.1.3 Edició de servei 25

4.1.4 Alta de subscripció 26

4.1.5 Edició de subscripció 26

4.1.6 Gestió de serveis 27

4.1.7 Cerca d’usuaris 27

4.1.8 Edició d’usuaris 27

4.2 Diagrama de classes 28

5. Implementació 30

5.1 Mockups 30

5.2 Maqueta 32

5.2.1 Dels Mockups a la maqueta 32

5.2.2 El Catàleg de serveis 35

5.2.3 Els formularis d’alta de servei i de subscripció 38

5.2.4 Edició del servei i la subscripció i nova versió del servei 39

5.3 Back-End 41

5.3.1 Controlador 41

5.3.2 Model 43

5.3.3 Gestió de les llibreries i dependències 44

5.4 Front-End 44

5.4.1 Thymeleaf o JSP? 45

5.4.2 Formularis i Handlebars 47

5.5 Seguretat 49

5.5.1 Gestió de les dades 49

5.5.2 Identificació d’usuaris 51

6 Qualitat del codi 51

6.1 Tests Unitaris 51

6.2 Integració (Jenkins) 51

6.3 Anàlisi del codi (SonarQube) 52

7 Conclusions 52

8 Fonts d’informació 52

9 Annexos 52

# 0. Resum

Amb la realització d’aquest Projecte de Final de Carrera (PFC), he buscat elaborar una aplicació web que servís en un futur per a la gent que se l’ha de quedar y que l’ha d’utilitzar. Crec que això dona un valor al projecte, ja que m’implica dins de l’empresa (i, per tant, dins del món laboral) amb la que vaig fer les pràctiques. A més, el fet d’ajudar a aquesta empresa a solucionar un problema real que tenien mitjançant l’aplicació web, ho vaig trobar també molt motivador des del meu punt de vista.

Avui en dia, la programació d’aplicacions web estan sofrint un canvi molt interessant: hem passat durant els últims anys de fer aplicacions per a un tipus de pantalla molt determinat, amb una resolució molt semblant entre totes, a haver de fer dissenys dinàmics o responsius, que segons la resolució i la mida de la pantalla mostrin el contingut de l’aplicació d’una forma o una altra, sense que perilli la integritat d’aquesta. A més, la presencia de cada cop més frameworks fa possible que la programació sigui molt més accessible, sobretot tot allò necessari per començar a programar, és a dir, realitzar la configuració inicial, que sempre ha estat una de les coses més difícils o, almenys, més lentes de fer.

Aquest PFC tracta, en definitiva, de la implementació del catàleg de serveis web i subscripcions existents al bus SOA gestionat per UPCNet, en el que els administradors poden portar al dia l’estat dels serveis i de les subscripcions, en el que es poden demanar nous serveis o subscripcions, i en el que es poden editar les dades dels serveis i subscripcions ja existents, tot això amb una seguretat implementada amb Spring Security y amb les eines de maquetació web més actuals.

# 1. Introducció

## 1.1 Objectiu

L’objectiu d’aquest projecte és el d’elaborar una aplicació web que serveixi per a dos conceptes diferenciats:

* **Eina de gestió dels serveis i subscripcions**: aquests són publicats per la comunitat UPC. S’ha de permetre als diferents sistemes d’informació fer tants serveis com vulguin i subscriure’s a totes les subscripcions que trobin útils, i UPCNet vol portar un registre automatitzat d’aquests moviments.
* **Publicació del catàleg de serveis**: actualment ja hi ha un catàleg de serveis, però aquest s’actualitza de forma manual cada cop que hi ha un nou servei o un servei es dona de baixa. Es ben difícil portar al dia totes les subscripcions de tots els serveis, i automatitzant-ho tot evitem aquesta feina.

Actualment, el registre de nous serveis o noves subscripcions es realitza mitjançant un formulari que s’ha de descarregar, omplir, i enviar. Això quedarà eliminat, ja que tots els registres seran a través de l’aplicació.

Necessitarem, també, gestionar tres rols d’usuaris diferents: administrador, que tindrà accés a tota l’aplicació; responsable oficial, que tindrà accés a tot allò que tingui relació amb el sistema d’informació del que és responsable, i responsable tècnic, que només tindrà accés a les dades del servei o de la subscripció d’un sistema determinat sempre i quan el responsable oficial així o hagi indicat.

Per entrar a l’aplicació es farà un registre bàsic, però que permetrà, en un futur i si la gent d’UPCNet ho troba adient, combinar-lo amb un registre amb LDAP (és a dir, amb l’usuari i contrasenya de la comunitat UPC).

## 1.2 Motivació

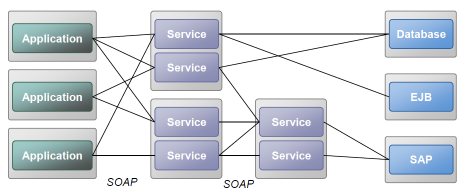
Aquest projecte pretén fer servir tot el que he aprés a l’hora de portar un projecte des del seu principi fins al final, fent ús dels coneixements tècnics adquirits durant les meves pràctiques com a becari durant tres anys i, molt especialment, durant l’últim any a UPCNet, empresa que rebrà aquesta aplicació web per mantenir-la i millorar-la en cas que ho trobin necessari, i de tot el que he après al respecte durant la meva carrera. La meva motivació principal, doncs, és donar sentit amb aquest projecte als anys que he passat estudiant i fent de becari a diferents empreses, traslladant al món real els meus coneixement.

Com que vaig canviar de lloc per continuar com a becari l’últim any de la carrera per aprendre Java, ja que durant els dos anys anteriors feia PHP, vaig trobar adient que el Projecte de Final de Carrera tingués alguna cosa a veure amb el que he estat fent durant l’últim any. També volia posar en pràctica tot el que he aprés de maquetació web mitjançant la llibreria Twitter Bootstrap, i aquest Projecte m’ho permetia.

Per últim, les facilitats que ofereix fer el Projecte de Final de Carrera a una empresa, i el fet que aquest projecte, un cop acabat, no quedarà guardat a un calaix sense cap utilitat, ha estat l’última de les raons que m’han portat a escollir un Projecte d’aquestes característiques.

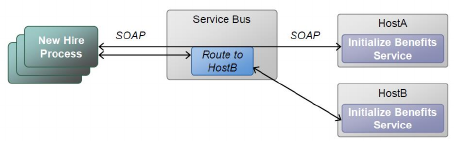
## 1.3 Estat de l’art

Primerament, cal veure què es un bus SOA, i quina utilitat té. Actualment, UPC gestiona més de 100 sistemes d’informació corporatius, que interactuen entre ells mitjançant unes 350 interfícies implementades a mida i amb diferents tecnologies. Aquesta situació, sense l’ajuda d’un bus SOA, es molt difícil de gestionar, ja que, com podem veure a la imatge 1, cada servei fa una connexió punt a punt amb el seu subscriptor, havent d’implementar cada servei, per exemple, la seva pròpia seguretat, o el mòdul de transformació de les dades en diferents protocols.



Imatge 1

Un bus SOA homogeneïtza aquesta xarxa punt-a-punt i serveix com a plataforma per on passen tots els serveis amb una única seguretat, i a on busquen connectar-se tots els subscriptors, convertint una xarxa impossible de gestionar en una xarxa ordenada i que passa tota pel mateix node.



Imatge 2

Un bus SOA (*Service Oriented Architecture*), per tant, permet la construcció d’aplicacions orientades a serveis, ofereix una seguretat única i els mòduls de transformació de les dades en diferents protocols, alleugerant la xarxa, i també posa a disposició de qui ho necessiti un catàleg de serveis per poder buscar aquells serveis que ens puguin interessar per subscriure’ns-hi.

Cal veure també com gestiona UPCNet, avui dia, el catàleg disponible de serveis i subscripcions. Com podem veure a la Imatge 1, actualment existeix un llistat per a la comunitat UPC de serveis disponibles, a on les persones que hi poden accedir poden trobar-hi informació sobre aquests serveis, com per exemple la descripció, la persona de contacte, el contracte, quins sistemes d’informació estan subscrits a aquest servei...



Imatge 3

Aquest catàleg, tal com he explicat abans, s’actualitza de forma manual, és a dir, en el cas d’una alta d’un nou servei, s’agafa el codi d’un servei ja existent, es modifica la informació per a que s’adeqüi al que ha indicat el responsable a la documentació entregada, i per últim es col·loca a la posició que li pertoca segons el sistema d’informació al que pertany. Aquesta forma d’actualitzar el catàleg de serveis fa que sigui molt difícil mantenir-ho tot al dia, ja que, com es pot veure a la Imatge 1, també es vol indicar quins serveis d’informació estan subscrits a un servei determinat.

## 1.4 Planificació temporal

[[ TODO: FALTA AFEGIR DIAGRAMA DE GANTT I EXPLICACIÓ D’AQUEST ]]

## 1.5 Resum de continguts de la memòria

Aquesta memòria està dividida en XXX apartats ben diferenciats entre ells, que son:

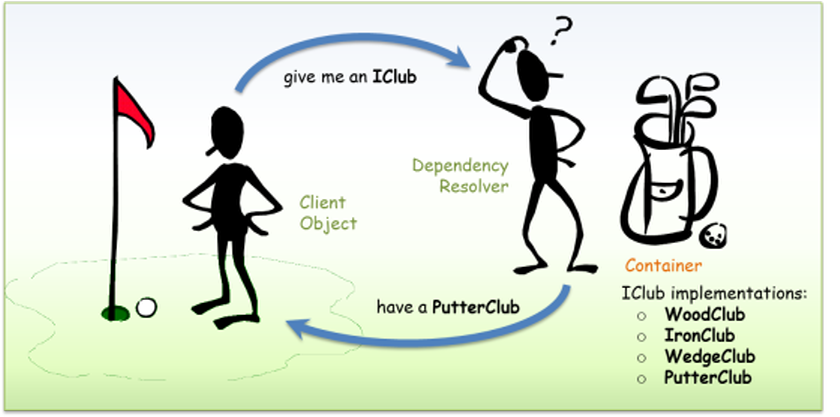
* Introducció
* Tecnologies implicades
* Presa de requisits
* Anàlisi
* Disseny
* Implementació
* Conclusió

# 2. Tecnologies implicades

## 2.1 BackEnd

### 2.1.1 Spring Framework

Spring Framework és el nom que rep tot el conjunt estandarditzat de conceptes, pràctiques i criteris per enfocar la creació d’una aplicació Java des de 0 amb l’ajuda de les eines que ofereix Spring dins del seu projecte principal. Aquest framework, dins del món Java, és el màxim exponent d’allò que es coneix com a Inversió de Control (IoC), que consisteix en un nou paradigma de programació en el que en comptes de programar unes decisions lògiques que configurin una seqüència que volem que el nostre programa segueixi, implementem les decisions que s’han de prendre arrel d’una petició concreta, deixant que una entitat externa generi les accions necessàries en l’ordre correcte. Al final, el que s’utilitza es el patró de disseny anomenat Injecció de dependències, a on subministrem objectes a una classe, en comptes de ser la classe la que crea els objectes. Per tant, la inversió de control, implementada amb el patró d’injecció de dependències, funcionaria de la següent manera:



Imatge 4

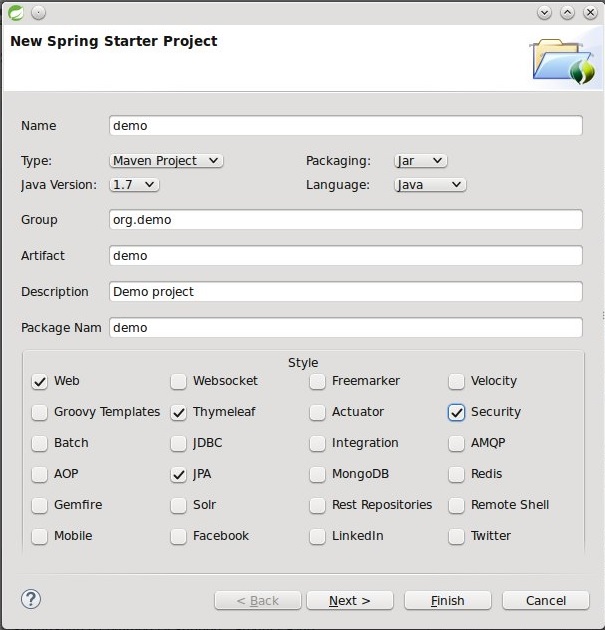
A la imatge 4 tenim un problema que volem resoldre d’una manera concreta. El client, en aquest cas el golfista, demana un dels pals, sense importar quin, que satisfaci la seva necessitat i deixa que una altra persona (el caddie, en aquest cas) ho resolgui.

D’aquesta manera, amb Spring Framework, registrem la lògica de negoci de l’elecció del pal en un altre lloc, i al client només hauríem de rebre la resposta, en aquest cas el pal de tipus Putter, ja que ens trobem molt a prop del forat.

L’elecció d’aquestes eines es deu, en la seva major part, en que son les eines que vaig aprendre a utilitzar durant les pràctiques a UPCNet, a més de ser les eines amb que desenvolupen a l’empresa. No hagués tingut gaire sentit realitzar l’aplicació en PHP, per exemple, quan el suport per part de l’empresa no hagués pogut ser tant important com ha estat amb unes eines que coneixen i amb les que treballen dia a dia que amb unes altres.

### 2.1.2 Spring Boot

Un cop hem decidit que utilitzarem Spring, toca parlar de com començarem l’aplicació. Spring Boot és una eina que ofereix la gent d’Spring per a que la configuració inicial de projectes basats en el seu framework sigui molt més senzilla. Un cop descarregat l’IDE (basat en Eclipse, de sobres conegut per tothom), només hem de seleccionar, dins de l’apartat Nou Projecte Spring Starter, les opcions que hi volem afegir. En el nostre cas, aquestes son les següents:

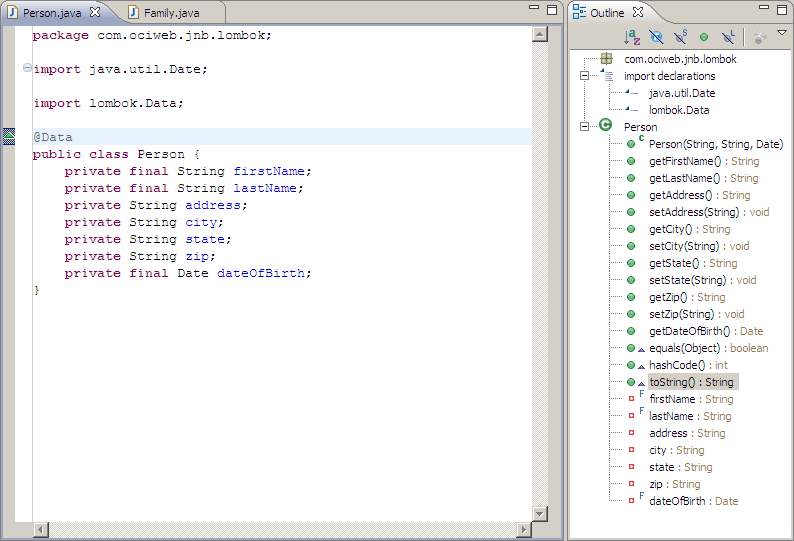


Imatge 5

D’aquesta manera, l’aplicació ens queda configurada per utilitzar tot allò necessari per a poder començar a programar, guardant temps per a les tasques més importants del projecte. A més, Spring Boot instal·la també un Apache Tomcat propi. Per tant, podem executar l’aplicació de forma local sense necessitat d’instal·lar a l’ordinador tota la infraestructura necessària fins que no s’hagi de passar a preproducció.

### 2.1.3 Resoldre el problema del codi *Boilerplate*

Es coneix com a codi *Boilerplate* aquell codi que s’ha d’introduir dins de tot programa molts cops amb una alteració mínima, o sense cap alteració. Dins de la programació orientada a objectes (POO), el cas més evident el trobem a cada classe, a on hem de posar tots els getters i els setters per a poder interactuar amb els camps de l’objecte. Una forma de resoldre aquest problema es mitjançant la llibreria Lombok. Això ho podem veure a la imatge següent:



Imatge 6

Com podem observar, important les llibreries de Lombok, podem posar l’anotació @Data a la classe, fent que l’IDE reconegui tots els mètodes que podem veure a la part de la dreta. Això ens estalvia, no només línies de codi, sinó temps, ja que només ens hem de preocupar d’implementar els mètodes no repetitius de cada classe.

A l’informe previ ja vaig parlar d’una eina de resolució del problema del codi *Boilerplate*. Aquesta era SpringRoo, un altre projecte de la gent d’Spring. Per què no he continuat utilitzant SpringRoo i, per contra, he canviat a Project Lombok?

La resposta es ben senzilla:

1. Spring ha deixat de donar suport a SpringRoo, per tant, hi podien haver errors que no fossin resolts per la gent d’Spring, mentre que Project Lombok segueix viu.
2. SpringRoo no era tot el dinàmic que ens volia fer creure, ja que s’havia de tenir la consola tota l’estona engegada amb el RooShell executant-se, i s’havia d’anar vigilant que tot canvi que es fes hi quedés registrat, i això no és necessari amb Lombok.
3. El codi queda molt més net amb Lombok que amb SpringRoo, tal com podem veure a la següent imatge:



Imatge 7

A la imatge, que correspon a les anotacions per a la mateixa classe, podem veure que la diferència més important entre les anotacions a Lombok i SpringRoo es que a SpringRoo hem de marcar tots els atributs que no siguin clau primària de la classe a l’etiqueta @RooEquals. Això comporta que, per a classes amb molts atributs, queda una llista molt gran que fa que el codi perdi llegibilitat. En canvi, a Lombok només hem d’escriure la clau primària, fent que quedi tot més clar.

### 2.1.4 Maven

Es tracta d’un projecte d’Apache Software Foundation, que fa que construir un projecte sigui molt més senzill, ja que Maven decideix la seva estructura , gestiona les dependències de llibreries i comprova que tot sigui correcte dins del cicle de vida d’un projecte. I tot això amb una forma molt senzilla de configurar: mitjançant un fitxer XML, el POM (Project Object Model).

Dins d’aquest POM hi tenim les dependències del projecte, els repositoris remots, els plugins de tercers... Per exemple, aquí hi inclouríem, després d’instal·lar-lo, la menció al Project Lombok.

També defineix una estructura bàsica pel projecte, cosa que fa que aquest pugui ser reutilitzable sense importar l’IDE amb el que treballem.

## 2.2 Persistència

### 2.2.1 Hibernate

Hibernate és una eina de mapeig objecte – relacional que facilita la construcció d’objectes a una aplicació Java a partir d’un model relacional tradicional d’una base de dades. A més, aporta un llenguatge de consultes anomenat HQL (Hibernate Query Language) a la vegada que una API que fa que ja hi hagi consultes predefinides. Per exemple, si volem buscar tots els elements d’un objecte *obj* per la seva *idPrimaria*, només hauríem de fer:

*obj.findAllByIdPrimaria(idPrimaria);*

Y la solució seria un objecte del tipus que fos *obj* amb la *idPrimaria* coincident amb aquella introduïda com a paràmetre.

Hibernate, a més, implementa l’estàndard JPA, part de la plataforma Java.

### 2.2.2 PostgreSQL

El sistema de gestió de Bases de Dades utilitzat es PostgreSQL, que implementa l’estàndard SQL92/SQL99 i l’elecció d’aquesta eina de gestió de la Base de Dades es deu, principalment, a dos factors:

1. En el moment del començament del projecte, no tenia cap experiència en cap altre gestor de Bases de Dades, per tant escollir, per exemple, MySQL sense haver tingut ocasió d’utilitzar-lo mai, afegia una complicació innecessària al projecte.
2. És el gestor utilitzat a UPCNet, per tant, l’elecció d’aquesta eina ha estat una mica com l’elecció de Java.

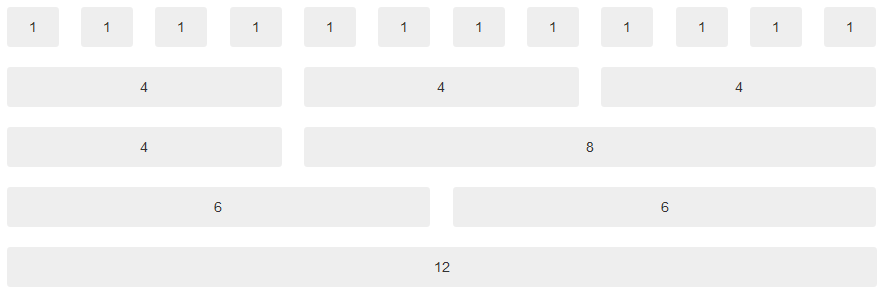
## 2.3 FrontEnd

### 2.3.1 Thymeleaf

Thymeleaf es una llibreria de Java que implementa un motor de plantilles HTML5 que facilita el treballar amb el patró de disseny Model Vista – Controlador (MVC). Conté tota una sèrie d’instruccions que fan que des d’una maqueta en HTML sigui molt senzill tant fer plantilles (per no haver de repetir codi de, per exemple, les capçaleres i els peus de pàgina), mostrar per pantalla les dades obtingudes del BackEnd, permet, si es vol, mantenir la maqueta en HTML pur a la vegada que es desenvolupa a sobre l’aplicació...

### 2.3.2 Twitter Bootstrap

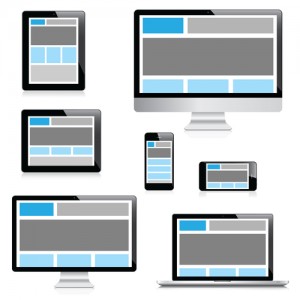
Es un framework desenvolupat per la companyia Twitter, famosa per la xarxa social, que ofereix tota una sèrie d’eines per a que, quan dissenyem una pàgina web, aquesta sigui fàcil de convertir en una pàgina responsiva a tot tipus de pantalles, i que a més segueixi unes pautes que pretenen ser estàndards (com per exemple, les cantonades arrodonides de botons i caixes d’inputs). La part més important d’aquest framework es el sistema de cel·les:



Imatge 8

Com podem veure, Bootstrap divideix la pantalla en 12 cel·les iguals, que nosaltres podem agrupar tal com ens vagi millor.

En cas que la pantalla sigui més petita, les cel·les es posaran una a sobre de l’altre, fent que sempre es vegi el contingut (necessari si estem des d’un telèfon mòbil amb la pantalla en vertical) o, si no volem que això passi, podem posar tot el contingut dins d’una cel·la de 12 posicions, i aquest es comportarà tal com es comporta habitualment, però adaptat segons la pantalla (el contingut es veurà més petit se estem a un telèfon des d’un principi, però es veurà tot). Aquest comportament el podem veure a la imatge 6:



Imatge 9

Podem veure, per tant, que el contingut es mostrarà sempre tal qual volem, sense tenir en compta quina pantalla estiguem utilitzant, i podem no deixar a l’atzar el comportament d’aquesta eina.

L’empresa Google ha posat al mercat el seu propi framework, que treballa de forma una mica diferent. Degut a la inexperiència amb les eines de Google, vaig optar al seu moment per utilitzar Twitter Bootstrap.

### 2.3.3 jQuery

Es tracta d’una llibreria Javascript que permet facilitar la manera de treballar amb els objectes Javascript, de manipular l’arbre del DOM, facilitar la creació d’events i animacions i agregar interacció amb l’eina AJAX. Però quina diferencia real hi ha entre escollir utilitzar jQuery o utilitzar, per contra, Javascript pur? Ja que jQuery es, bàsicament, una llibreria Javascript que el que fa es alleugerar el codi HTML, la forma més senzilla de veure la diferència és amb un exemple:

Com seleccionaríem un element amb una etiqueta determinada (en aquest exemple, seleccionarem el body de la pàgina HTML) i hi canviem el color del background?

* jQuery

$ (‘body’) .css (‘background’, ‘#ccc’);

* Javascript

Function changeBachground(color) {

Document.body.style.background = color;

}

Onload=”changeBackground (‘red’);”

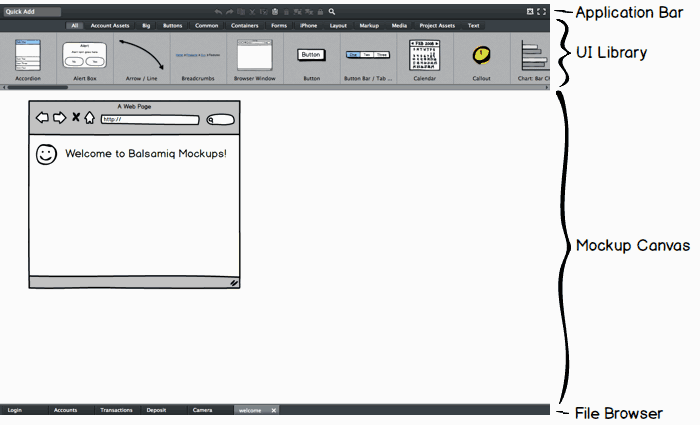
Veiem que el que podem fer amb una línia a jQuery, necessita una miqueta més de treball amb Javascript. A més, jQuery és molt més intuïtiu a l’hora d’interactuar amb els elements de la nostra pàgina.

## 2.4 Maquetació

### 2.4.1 Balsamiq Mockups

Un cop es tenen definides totes les eines amb les que es programarà l’aplicació, toca posar-se en marxa i començar a fer la maqueta del projecte per presentar-la al client i, després, començar a programar a partir d’aquesta maqueta. Vaig poder escollir entre dues opcions a l’hora de fer la maqueta: fer-la a mà, de forma tradicional, amb un llapis y un paper, i després fer fotocopies per tothom, o bé utilitzar una eina com Balsamiq Mockups.

Aquesta eina permet, de forma molt senzilla, realitzar el que s’anomenen Mockups de l’aplicació, que no deixen de ser captures de pantalla de com es veurà la nostra aplicació en un futur, amb tot un seguit d’objectes que pots anar col·locant a la pantalla utilitzant la llibreria que l’aplicació ofereix, i que podem veure a la imatge a continuació, etiquetat com UI Library:



Imatge 10

Com podem veure, i que més endavant quan entri més en profunditat en tota la feina feta amb l’aplicació de Govern del Bus SOA, tenim moltes opcions que només hem de seleccionar i arrossegar-lo cap al lloc del canvas que volem. D’aquesta forma aconseguim una maqueta sense entrar en convencions de colors i formes, només amb l’estructura principal de l’aplicació, de forma ràpida i permet, el que es més important, fer canvis al moment. El fet de poder exportar-ho tot, un cop acabar, a PDF, fa que sigui molt còmode per enviar i imprimir.

## 2.5 Model de dades

### 2.5.1 Project Mogwai

Un cop dissenyada tant la maqueta mitjançant l’eina de Mockups, i després de convertir aquesta maqueta en codi HTML i jQuery, vaig dedicar els meus esforços en fer el model de dades de forma que quedés comprensible i fos fàcilment convertible en codi SQL. L’eina escollida va ser l’ERM Designer que ofereix la gent de Project Mogwai. Aquesta eina et permet construir de forma gràfica una base de dades relacional, i després convertir directament aquesta base de dades en codi SQL, aconseguint, d’aquesta manera, estalviar feina.

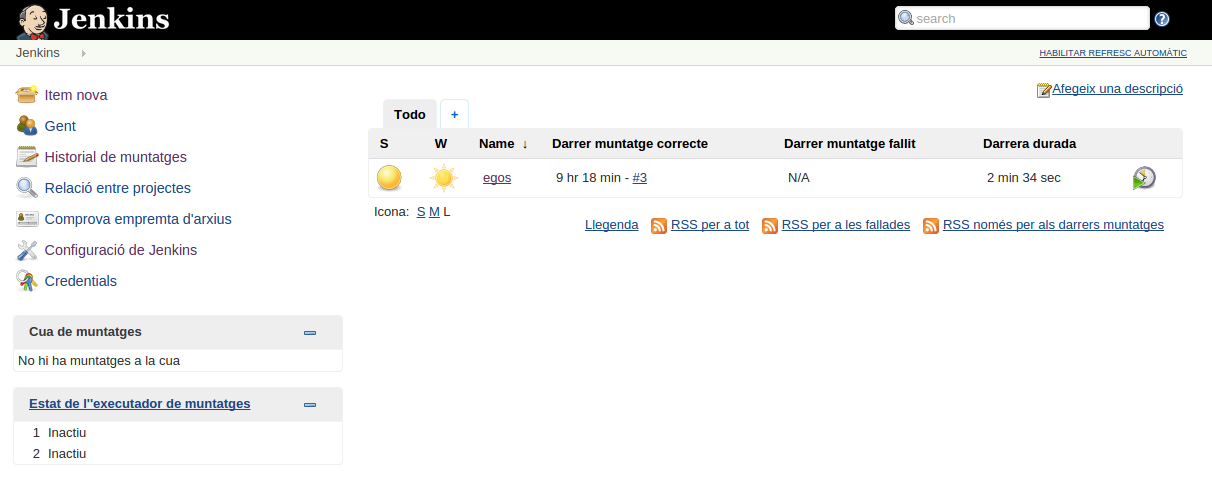
Aquestes han estats les eines escollides per a poder realitzar amb èxit el projecte de final de carrera. Al final del document, a les referències, podreu trobar adreces web que porten a la documentació de totes aquestes aplicacions.

## 2.6 Integritat i qualitat del codi

### 2.6.1 Jenkins

Jenkins es un software d’integració contínua de codi obert. Es tracta d’un sistema que corre en un servidor que funciona con un contenidor de servlets. Però que vol dir això? Primerament, caldria saber que la integració contínua es un paradigma de la programació que entén que es necessari executar un codi i fer-lo passar tots els tests de que es compon tants cops com sigui possible, per detectar el més aviat possible errors. Jenkins té una integració amb GIT, el repositori de versions per excel·lència actualment, així que es descarrega la branca de GIT que li indiquem i l’executa, fent-li passant tots els tests.

Un cop executem una versió al Jenkins, i aquesta acaba de compilar-se, podem veure al mateix Jenkins si tenim errors de compilació o a algun test o no. En cas que la compilació hagi fallat, ens tornarà un error de Java per indicar-nos on falla. Si ha funcionat, haurem d’anar a l’eina de qualitat del codi que estiguem utilitzant, en aquest cas SonarQube.



### 2.6.2 SonarQube

Conegut realment com a Sonar solament, es la plataforma escollida per avaluar codi font. No només es capaç de detectar errors greus al codi, sinó que ens diu quines parts d’aquests hem d’arreglar per complir amb les convencions de codificació i que el codi tingui un manteniment més senzill en el futur. Hi ha regles molt interessants, com per exemple aquelles que avisen del problema dels objectes mutables, com altres que potser no tenen tanta importància, com per exemple que els Javadocs haurien d’acabar amb un punt i seguit. De totes formes, el programador pot escollir quines regles seguir o no, i ignorar o desactivar aquestes regles.

Una altra de les coses interessants que ens ensenya es la cobertura que tenen els tests unitaris sobre el codi, ja sigui per línia o per branques. És a dir, si trobem un *if*, haurem de fer un test que compleixi la condició i un altre que no per poder assolir un 100% de cobertura en aquesta branca.

També genera una taula que compara tots els projectes que utilitzen aquesta eina, cosa que pot fer que els diferents equips que fan aplicacions a la mateixa empresa competeixin per veure quina té una qualitat de codi més elevada.

## 

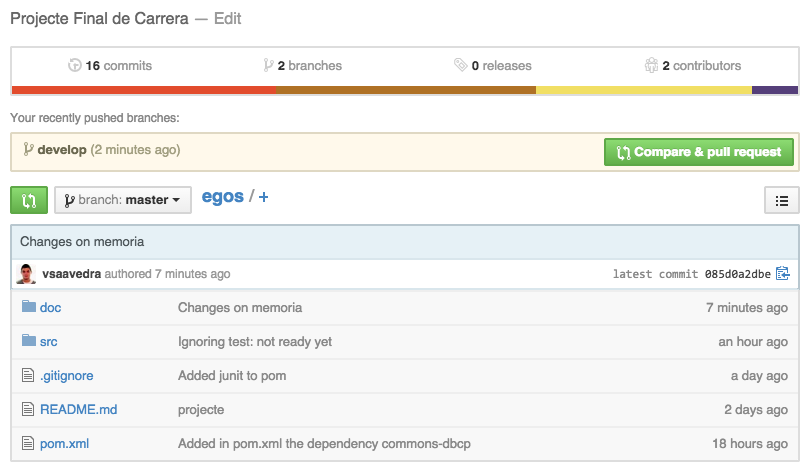
## 2.7 Control de versions

### 2.7.1 GIT

Dissenyat per Linus Torvalds, està pensant per aquelles aplicacions que tenen molts arxius diferents de codi font. Per tant, una aplicació Java utilitzant un model MVC es perfecte per utilitzar GIT. Permet crear diferents entorns per a que els diferents programadors d’un projecte treballin sempre en local i només comparteixin el seu codi un cop estan segurs que no molestarà als altres amb errors. Cada programador por fer una branca diferent per a que, per exemple, la correcció d’un error, no interfereixi amb el desenvolupament normal de l’aplicació. Per a aquesta aplicació hi ha tres branques:

* Master: creada pel GIT, en aquesta branca posaré les versions definitives del projecte.
* Origin: creada pel GIT, aquesta branca marca el meu treball de forma local.
* Develop: creada per mi, aquesta branca tindrà les actualitzacions del codi realitzat a Origin, abans de que aquest nou codi passi per la integració de Jenkins i la qualitat del SonarQube.

Tot i la possibilitat d’escollir diferents serveis web que actuïn com a repositoris, m’he decantat per un dels principals: GitHub.



# 3. Especificació de requisits

## 3.1 Introducció

En aquest apartat de la memòria quedaran marcades les pautes que seguiré per dur a terme l’aplicació de govern del bus SOA demandat per UPCNet. Aquestes pautes han estat recollides directament del client.

Aquesta especificació de requisits està subjecte en tot moment a possibles canvis i variacions proposats tant pel client com pel desenvolupador, en aquest cas jo, mentre duri la implementació de la solució de l’aplicació web.

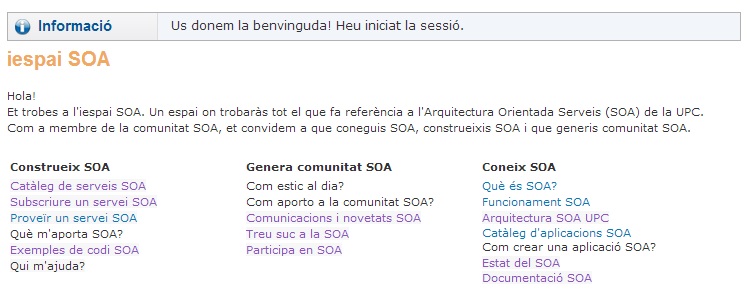
## 3.2 Àmbit

L’objectiu d’aquesta aplicació és el de desenvolupar una aplicació web que ajudi a mantenir el catàleg de serveis que gestiona UPCNet. A més, permetrà als usuaris d’aquest catàleg oferir nous serveis, així com subscriure’s a serveis ja existents. Tant l’administrador com els usuaris tindran una pantalla a on podran veure l’estat dels seus serveis i de les seves peticions de canvis, que es faran en format de comentaris.

El catàleg de serveis tindrà un senzill filtre y una sèrie d’opcions per a que els usuaris trobin el més fàcilment possible el que busquen, ja que l’objectiu es que la xarxa de serveis web creixi tant com sigui possible. Tot això tindrà, a més una capa de seguretat, que constarà d’un sistema de registre a l’aplicació propi i d’un sistema de gestió dels usuaris.

## 3.3 Descripció general

Primer de tot, es va decidir que miraríem el que hi ha ara mateix. Com s’ha pogut veure a l’apartat 1.3, sobre l’estat de l’art del projecte, actualment UPCNet gestiona el catàleg de serveis i subscripcions web a través de la pàgina IEspaiSOA.



Imatge 11

Aquí trobem, dins l’apartat “Catàleg de serveis”, el que hem pogut veure a la imatge X de la pàgina Y, que no es altre cosa que el catàleg de serveis. El gran problema d’aquest catàleg és que el seu manteniment es fa de forma manual i, per tant, es molt complicat que estigui correctament actualitzat sempre, ja que hi poden haver errors humans.

No només podem veure, dins d’aquest catàleg, els serveis que hi ha disponibles. També podem veure diferents versions del mateix servei, una sèrie d’enllaços que ens porten al contracte, a on podem trobar les característiques del servei i les probes que podem fer sobre ell; o als WSDLs d’integració i al d’explotació. WSDL son les sigles de Web Service Description Language, i es un arxiu XML que descriu el comportament del servei web, és a dir, a on hi podem trobar coses com el tipus de dades que utilitza, el format dels missatges que ha de rebre o que envia, el tipus de port que utilitza...

Entre altres coses hi trobem també una descripció, els Sistemes d’Informació subscrits a aquest servei, i una persona de contacte. Podem veure, també, que un Sistema d’Informació pot estar subscrit a més d’un servei, i que pot oferir més d’un servei.

Si tornem a la pàgina principal de l’IEspaiSOA, hi trobarem els dos enllaços que ens porten als formularis que serveixen tant per subscriure’s a un servei SOA, com per oferir un nou servei SOA.

Dins d’aquests enllaços, hi trobem pàgines (semblants en els dos casos) a on trobem petites instruccions que ens serviran per poder oferir un nou servei o subscriure’ns a un d’existent, tal com podem veure a la següent imatge:



Imatge 12

També en podem descarregar els documents que contenen els formularis d’alta de servei o de subscripció, o podem donar d’alta directament amb el formulari web. Aquests formularis varien una miqueta en funció de si el que busquem es fer l’alta d’una cosa o d’una altra. Un dels principals problemes es la quantitat de camps que hi ha, cosa que fa que l’usuari s’espanti, per tant se’m demana trobar una solució a això. També se’m demana fer una revisió de l’estil del catàleg, per donar-li una visió una mica més actual i facilitar que els usuaris puguin trobar més fàcilment els serveis que necessiten.

Pel que fa a la gestió de peticions de serveis i subscripcions, actualment es fa tot a través de correus electrònics, cosa que fa que el procés sigui lent. A més, per l’usuari, suposa un problema haver d’estar pendent tot el dia del correu, per comprovar si hi ha alguna petició de canvi d’algun paràmetre dels serveis, o una petició d’alta de servei nou o de subscripció. Per a evitar en la mesura del possible la utilització del correu electrònic, s’ha pensat en facilitar una plataforma de comentaris a l’aplicació eGOS que permetrà mostrar ràpidament alertes dins de l’aplicació si hi ha algun comentari nou, així com mantenir un històric de comentaris per saber què s’ha anat canviant amb el temps.

També s’incorporarà un filtre que permetrà a l’usuari administrador buscar i ordenar els servis segons li convingui.

Per últim, l’aplicació haurà de contar, necessàriament, amb un sistema de gestió d’usuaris per habilitar l’entrada a l’aplicació i modificar les funcionalitats d’aquesta en funció del rol de l’usuari. En un futur, s’haurà d’habilitar l’entrada a l’aplicació en funció de l’LDAP d’UPC, encara que de moment pot funcionar amb una gestió d’usuaris pròpia.

# 4. Anàlisi

## 4.1 Casos d’ús

Un cas d’ús es la descripció dels passos o activitats que s’hauran de dur a terme a cada procés de l’aplicació. Dins d’aquesta aplicació tindrem els següents casos d’ús:

* Catàleg
* Alta de servei
* Edició de servei
* Alta de subscripció
* Edició de subscripció
* Gestió de serveis
* Cerca d’usuaris
* Edició d’usuaris

### 4.1.1 Catàleg

|  |  |
| --- | --- |
| **Cas d’ús** | Catàleg de serveis |
| **Actors** | Usuari i administrador |
| **Propòsit** | Consulta de catàleg de serveis.  Filtrat dels serveis per servei o subscripció  Filtrat de serveis per cerca de nom.  Filtrat de serveis per metamodel, etiquetes o formats de sortida.  Mostra les alertes dels serveis que gestiona l’usuari.  Permet entrar a un servei en particular.  Permet afegir un nou servei |
| **Pre-condicions** | Cap. Aquesta pàgina podrà ser consultada per un usuari sense validació. |
| **Post-condicions** | Cap |

### 4.1.2 Alta de servei

|  |  |
| --- | --- |
| **Cas d’ús** | Alta de servei |
| **Actors** | Usuari i administrador |
| **Propòsit** | Demanar l’alta d’un nou servei emplenant tots els camps. |
| **Pre-condicions** | L’usuari ha d’estar identificat a l’aplicació. |
| **Post-condicions** | S’ha d’haver creat un nou servei amb l’estat “Esborrany”. |

### 4.1.3 Edició de servei

|  |  |
| --- | --- |
| **Cas d’ús** | Edició de servei |
| **Actors** | Usuari i administrador |
| **Propòsit** | * Usuari   Permet afegir comentaris demanant canvis que només seran visibles per ell mateix, el seu equip i l’administrador.  Permet afegir, a més del comentari, un arxiu, que hauria de ser un manual d’ús.  Permet consultar les dades del servei.  Permet afegit noves versions de serveis ja existents.   * Administrador   Permet modificar les dades del servei en funció de les peticions de l’usuari.  Permet afegir comentaris. |
| **Pre-condicions** | L’usuari ha d’estar identificat a l’aplicació. |
| **Post-condicions** | En cas de modificació o de comentari afegit, ha de quedar constància a la base de dades. En cas de consulta, no ha de canviar res. |

### 4.1.4 Alta de subscripció

|  |  |
| --- | --- |
| **Cas d’ús** | Alta de subscripció |
| **Actors** | Usuari i administrador |
| **Propòsit** | Demanar la subscripció a un servei existent. |
| **Pre-condicions** | L’usuari ha d’estar identificat a l’aplicació. |
| **Post-condicions** | S’ha d’haver creat una subscripció amb l’estat “Esborrany”. |

### 4.1.5 Edició de subscripció

|  |  |
| --- | --- |
| **Cas d’ús** | Edició de subscripció |
| **Actors** | Usuari i administrador |
| **Propòsit** | * Usuari   Permet afegir comentaris demanant canvis que només seran visibles per ell mateix, el seu equip i l’administrador.  Permet consultar les dades de la subscripció.   * Administrador   Permet modificar les dades del servei en funció de les peticions de l’usuari.  Permet afegir comentaris. |
| **Pre-condicions** | L’usuari ha d’estar identificat a l’aplicació. |
| **Post-condicions** | En cas de modificació o de comentari afegit, ha de quedar constància a la base de dades. En cas de consulta, no ha de canviar res. |

### 4.1.6 Gestió de serveis

|  |  |
| --- | --- |
| **Cas d’ús** | Gestió de serveis |
| **Actors** | Usuari i administrador |
| **Propòsit** | Consulta dels serveis i subscripcions gestionades per l’usuari. En cas que sigui l’administrador, serà una vista del catàleg amb un filtre més potent.  Consultar els serveis que tenen comentaris sense veure. |
| **Pre-condicions** | L’usuari ha d’estar identificat a l’aplicació. |
| **Post-condicions** | Cap |

### 4.1.7 Cerca d’usuaris

|  |  |
| --- | --- |
| **Cas d’ús** | Cerca d’usuaris |
| **Actors** | Administrador |
| **Propòsit** | Permet la cerca mitjançant d’un filtre dels usuaris registrats a l’aplicació.  Habilita l’edició, la creació i l’eliminació d’usuaris. |
| **Pre-condicions** | Només serà accessible per l’administrador de l’aplicació |
| **Post-condicions** | Cap |

### 4.1.8 Edició d’usuaris

|  |  |
| --- | --- |
| **Cas d’ús** | Edició d’usuaris |
| **Actors** | Administrador |
| **Propòsit** | Permet l’edició de les dades personals d’un usuari en particular.  Permet eliminar el registre de l’usuari. |
| **Pre-condicions** | Només serà accessible per l’administrador de l’aplicació |
| **Post-condicions** | L’edició o l’eliminació de l’usuari ha de quedar registrat a la base de dades. |

## 4.2 Diagrama de classes

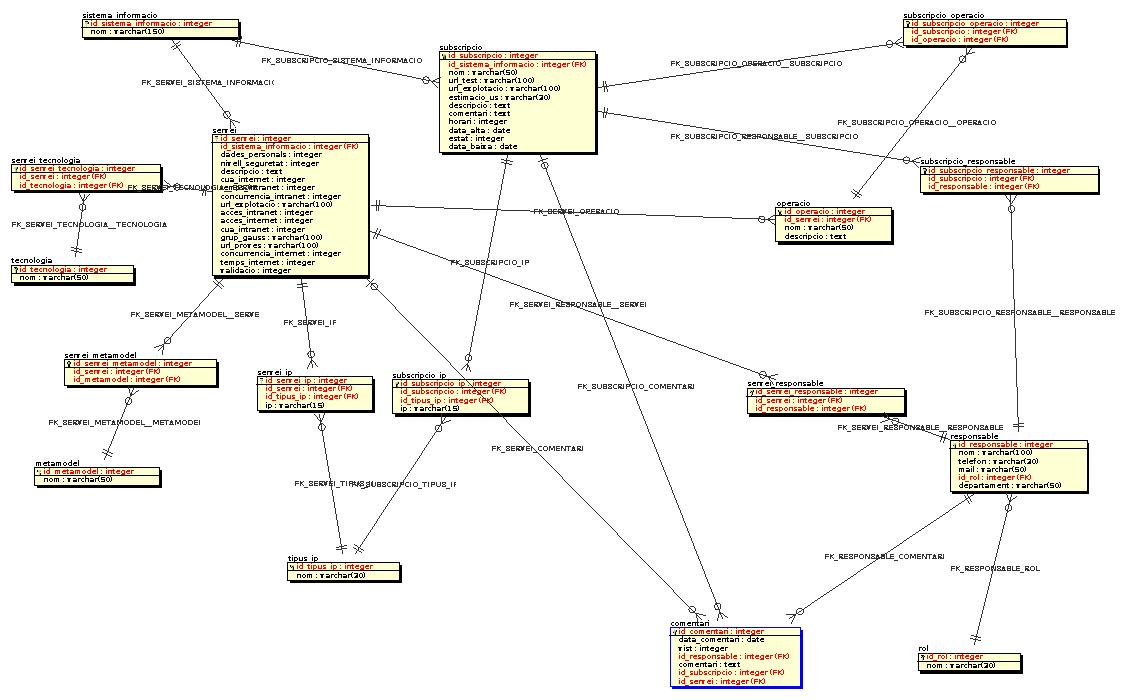
Un diagrama de classes és un tipus de diagrama estàtic que descriu l’estructura d’un sistema mostrant les seves classes, els seus atributs i les relacions entre ells. Per a l’elaboració d’aquest diagrama de classes s’ha utilitzat l’eina ERM Designer, de l’equip de Mogwai. La base de dades seleccionada per a l’aplicació és PostgreSQL, i s’ha utilitzat el client PgAdminIII durant tot el desenvolupament de la mateixa.

Gràcies a l’aplicació ERM Designer, que permet programar un diagrama de classes i les seves connexions entre sí, a més de després oferir el codi sql per a introduir-lo al nostre client de base de dades, crear-ne una partint de la pressa de requisits és molt més senzill.

Les taules que formen part del diagrama de classes de la nostra aplicació son les que surten a la següent llista:

|  |  |
| --- | --- |
| Taules mestres | Servei |
|  | Sistema d’informació |
|  | Subscripció |
|  | Operació |
|  | Tecnologia |
|  | Metamodel |
|  | Comentari |
|  | Responsable |
|  | Rol |
|  | Etiqueta |
|  | Estat |
|  | Tipus\_ip |
| Taules d’enllaç | Subscripció\_Operació |
|  | Servei\_Responsable |
|  | Subscripció\_Responsable |
|  | Servei\_IP |
|  | Subscripció\_IP |
|  | Servei\_Etiqueta |

Com podem veure, tenim taules mestres per els principals elements de la nostra aplicació. Un cop acabat el disseny del nostre model de dades a l’aplicació, va quedar com es mostra a la següent imatge:



A la part esquerre de l’esquema podem veure com es relacionen les entitats servei i subscripció amb algunes de les taules necessàries per al formulari d’alta, a més de amb el sistema d’informació, que serà l’entitat que gestioni un servei determinat i les subscripcions d’aquest servei. Podem veure totes les entitats que més endavant serviran per filtrar el nostre catàleg, com poden ser la tecnologia amb que està implementat; el metamodel, és a dir, una organització per categories, y les etiquetes o tags, que serveixen per filtrar d’una forma més descriptiva.

A la part dreta de l’esquema, per contra, podem veure com es relacionen els serveis i les subscripcions amb les taules de comentaris i les de responsables.

Pel que fa als comentaris, el client va decidir que la millor forma de comunicació entre els usuaris de l’aplicació i els administradors d’aquesta era a través de comentaris. Aquests es poden deixar tant a les pàgines de cada servei com a les de cada subscripció. Serà el canal de comunicació per on es podran demanar modificacions a la configuració tant dels serveis com de les subscripcions.

A més, hi haurà una petita alerta que es podrà veure tant al catàleg si s’està identificat a l’aplicació, com a l’apartat ‘Els meus Serveis’ que permetrà veure a on hi ha missatges que encara no em visualitzat.

Hi haurà responsables i responsables tècnics, definits per la taula ‘rols’ i que podran fer unes operacions o unes altres. A aquesta taula ‘rols’ també hi trobarem el d’administrador de l’aplicació.

Es pot veure una imatge del model de classes als annexos, en una millor resolució.

# 5. Implementació

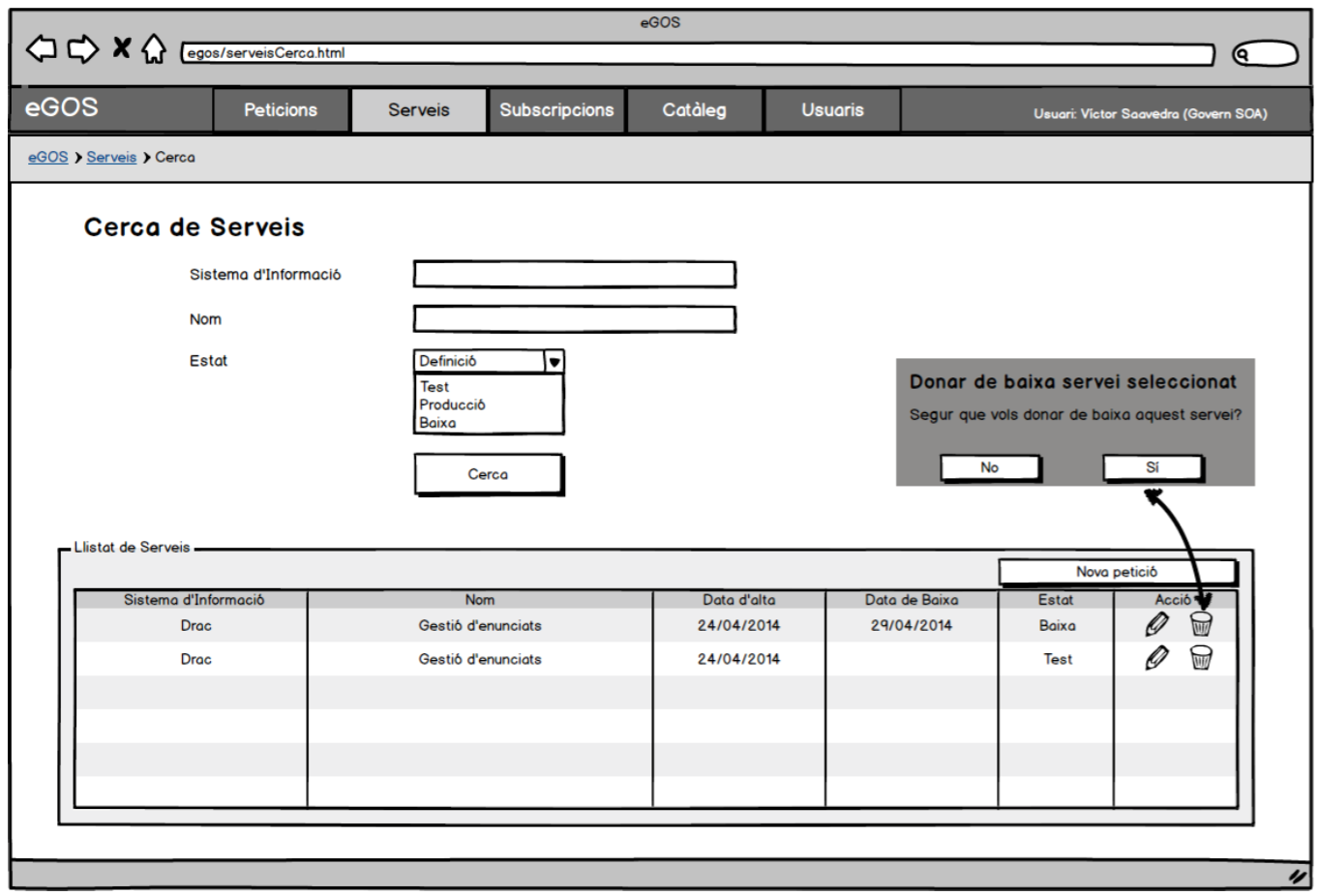
Dins d’aquest apartat, descriuré la feina realitzada durant el procés de disseny preliminar amb els Mockups i la realització de la maqueta. També parlaré de la implementació del model vista-controlador al back-end de l’aplicació, i de les pecurialitats trobades al front-end.

## 5.1 Mockups

Primer de tot, un cop feta la presa de requisits i detectats els casos d’ús, es va decidir que era l’hora de fer els Mockups.

Com s’ha explicat abans, la diferència entre els Mockups i realitzar un prototip amb un llapis és que un cop tens el primer Mockup, realitzar canvis sobre aquest es molt més senzill que en una fulla de paper amb un llapis, ja que l’arxiu queda guardat.

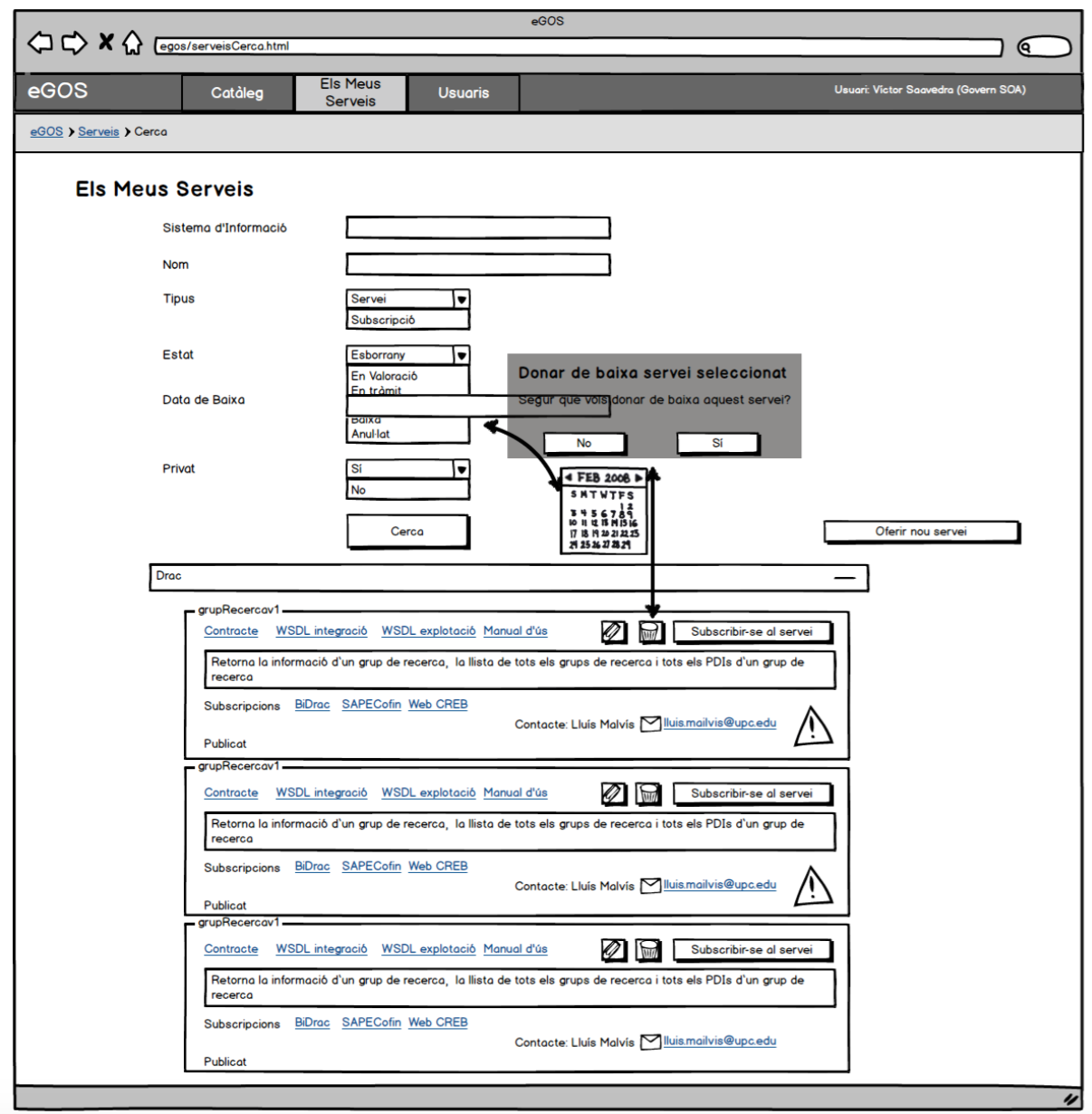
En aquest apartat s’ensenyarà, tot seguit, la diferència entre el primer Mockup que es va fer per al cas d’ús del catàleg de serveis, i el que al final el client va donar per bo. D’aquesta manera es pot veure el canvi que hi va haver conforme es va anar presentant al client. Només com a anotació, cal dir que es van fer 5 versions dels Mockups fins que al final el client va quedar content amb el resultat i va donar llum verda per passar a la següent etapa del projecte.



Com podem veure, en una primera versió preliminar a l’apartat ‘Els meus serveis’ (uns dels primers que es van traspassar a Mockups), els serveis estaven organitzats en una taula, i també hi havia un formulari de cerca (que actualment encara es manté). No hi havia, però, cap senyalització sobre els possibles missatges que el servei podia rebre, i l’ordenació del servei era bastant senzilla. Al client no va terminar d’agradar-li aquesta versió, i un cop fet el Mockup del catàleg (que es podrà veure als annexos) va voler que l’organització d’aquest apartat, pel que fa als serveis de cada usuari, fos igual.

Fer iteracions amb els Mockups és molt senzill, ja que tots els elements que es veuen a la part de la taula es poden enllaçar entre ells (de forma que, movent l’element principal, en aquest cas la taula, es mouen tots en la mateixa posició), i sempre es poden moure per separat o eliminar sense cap complicació.

Després de cinc iteracions, el resultat final va ser el següent:



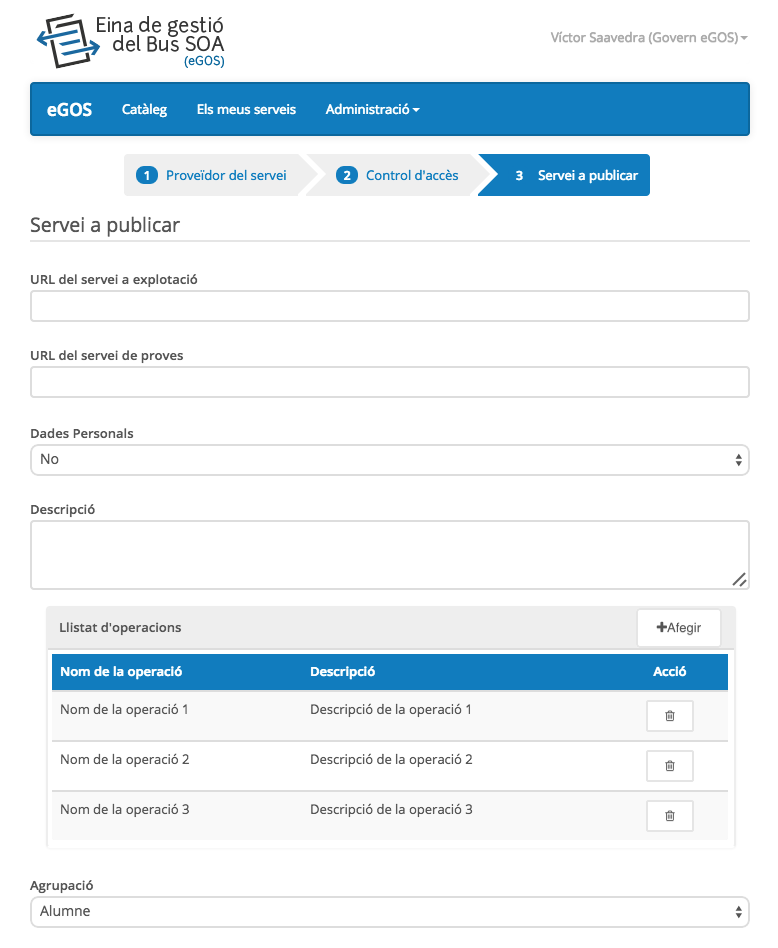
Que és el que podem veure a l’aplicació, un cop traspassat a la maqueta.

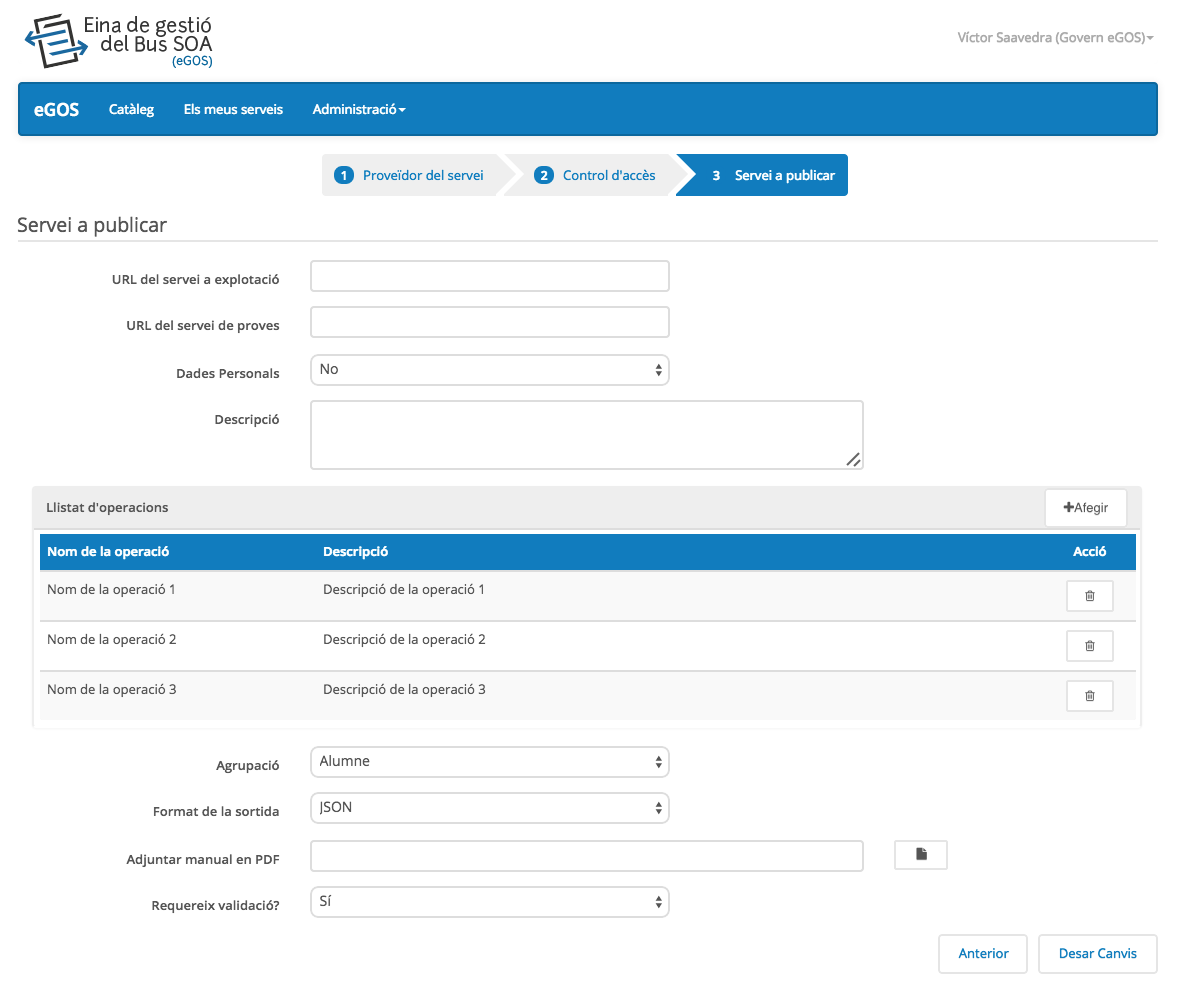
## 5.2 Maqueta

### 5.2.1 Dels Mockups a la maqueta

Un cop acabats els Mockups, el següent pas era fer la maqueta en HTML de l’aplicació, per tal de que el client la veiés en colors i tal qual queda a la versió final. Aquí és on es pot veure la conveniència de primer fer Mockups, que és una feina que no consumeix gaire temps, i a on es poden fer grans canvis (inclús a davant mateix del client, ja que només es arrastrar i col·locar gràficament els diferents elements), cosa que és molt més complicat quan hi ha HTML pel mig. D’aquesta forma, el client accepta el plantejament fet als Mockups, i només s’ha de passar aquest a una versió web, a on com a màxim canviaran els colors dels diferents elements (botons, enllaços, contenidors...).

Una de les peticions del client era que l’aplicació fos *responsive*, és a dir, adaptable per dispositius mòbils. Fa uns anys, això era més difícil d’aconseguir, ja que s’havia de programar tot amb CSS des de zero.

Però amb l’arribada de Twitter Bootstrap, la llibreria de la que parlo al capítol 2.3.2, la programació de l’esquelet d’una aplicació web adaptable a dispositius mòbils es torna molt més senzilla.

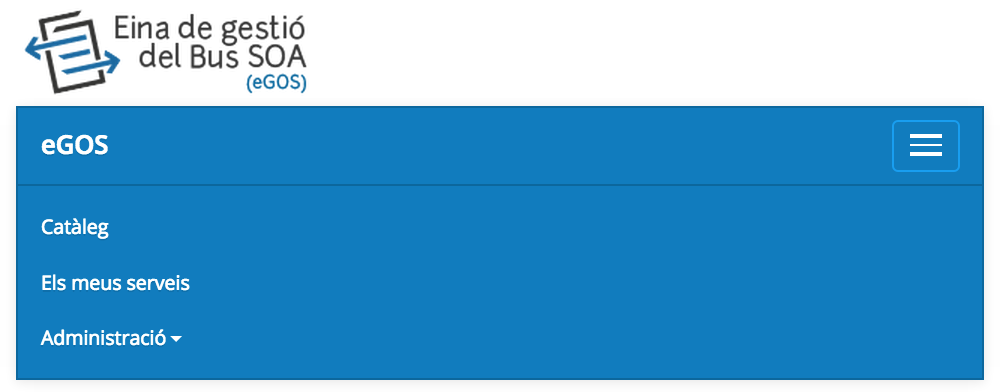


Com podem veure a les imatges de dalt d’aquest text, la imatge de la dreta correspondria a una vista d’escriptori normal. Per contra, la vista de l’esquerra es correspondria amb la vista de l’aplicació a través d’un telèfon mòbil. Es pot veure com els elements s’han anat recol·locant automàticament, i com els camps de text creixen per ocupar tota la pantalla en horitzontal, i que s’assembli més a la idea que tenim d’una aplicació mòbil.

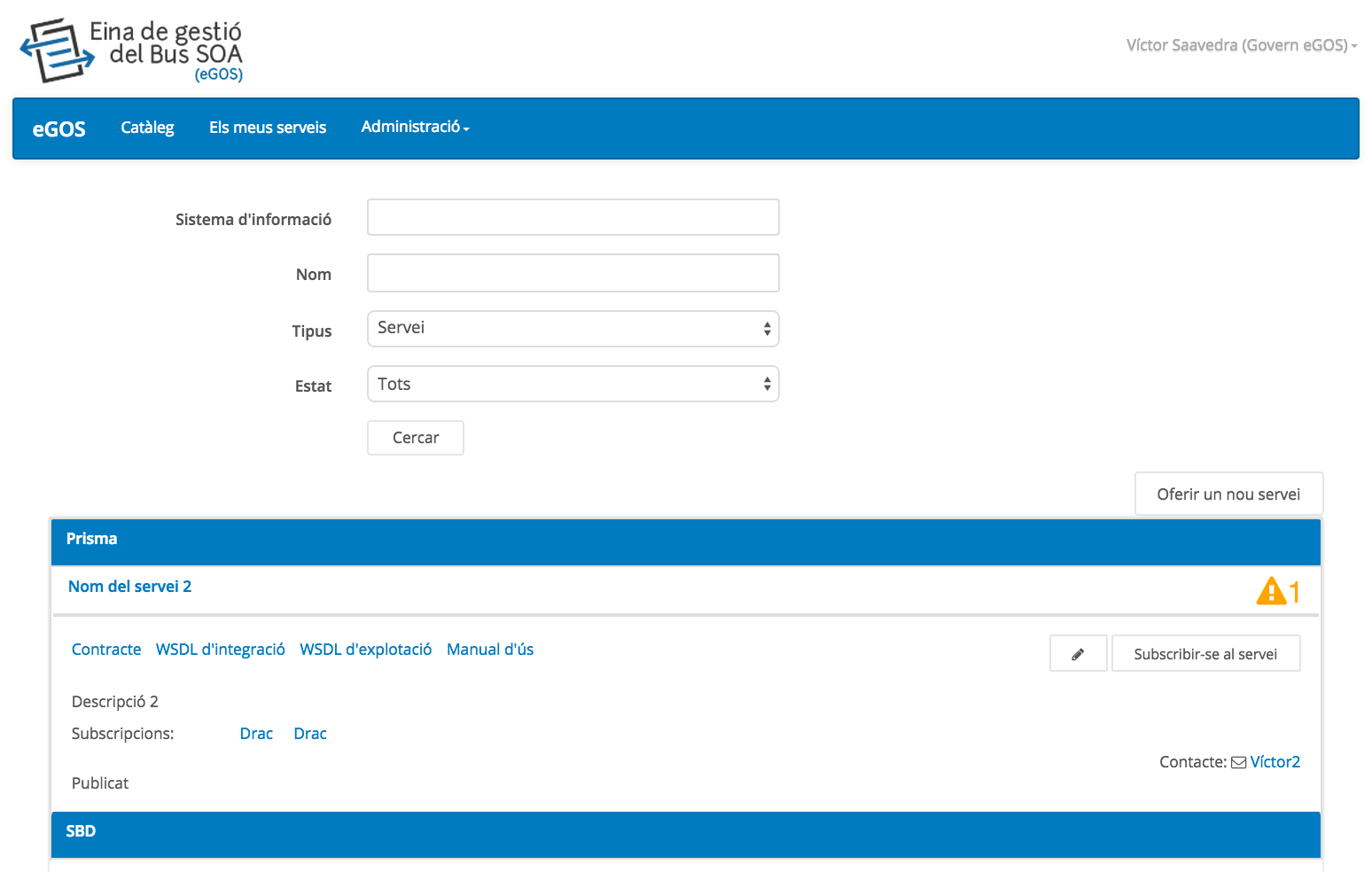
Si fos necessari que l’aplicació hagués de cabre en un espai molt més reduït, per exemple si la veiem des d’un telèfon mòbil amb una pantalla molt petita, el menú també es s’adapta a aquest problema, amagant-se i col·locant un botó per desplegar-lo.



Com podem veure en la foto anterior, i comparant-la amb la imatge de la maqueta, podem veure que el menú ha desaparegut, i en el seu lloc, enganxat a la part dreta, tenim un botó, característic sobretot dels dispositius Android i que fa referència als menús desplegables. Aquesta característica s’aconsegueix mitjançant les classes de bootstrap ‘*navbar navbar-default*’, que afegeixen la barra de navegació a l’aplicació i que la transformen quan és necessari. Aquí podem veure la barra de navegació desplegada quan està en format mòbil:



Pel que fa al traspàs des de Mockups, com podem veure l’aplicació s’assembla molt a allò que el client ja ha acceptat prèviament, i per tant els canvis, com ja hem dit abans, seran mínims. Podem comprovar com ha quedat l’apartat ‘Els meus serveis’ en la maqueta, i comparar-lo amb el Mockup del capítol anterior:

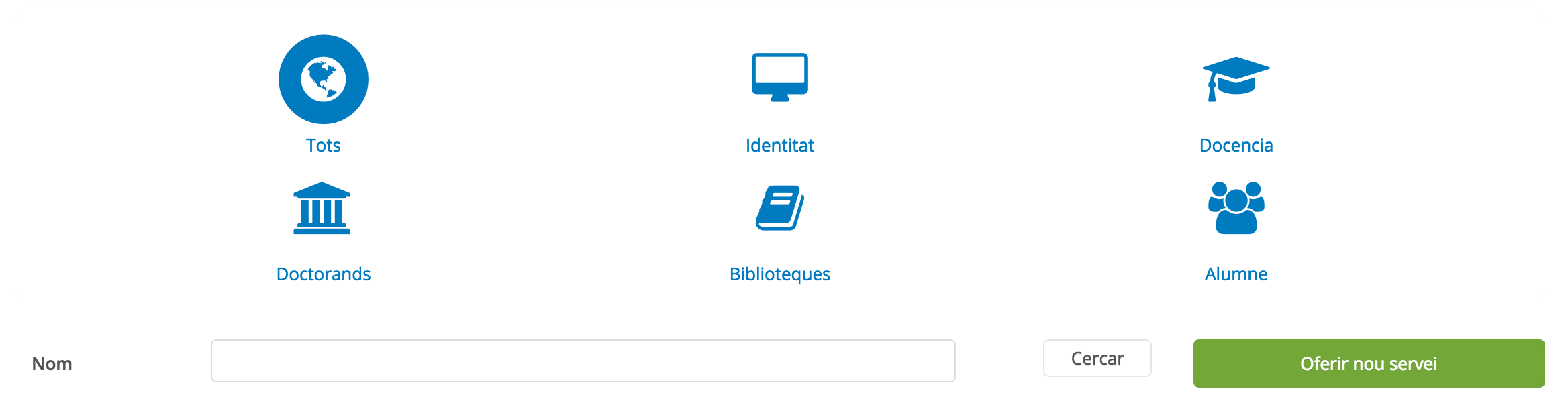


Els serveis estan organitzats pel seu sistema d’Informació, que poden compartir. El client va decidir que la millor opció era organitzar-los d’aquesta manera, en acordió, perquè creia que seria més fàcil per l’usuari final organitzar-se dins d’això. També podem veure al final del nom del servei una senyal d’alerta amb un número: representa els comentaris no llegits per la persona identificada a l’aplicació, i cada servei té la seva pròpia alerta en cas que fos necessari. Podem organitzar també els resultats per subscripcions, i veurem els serveis als que estem subscrits, en comptes dels serveis que proveïm.

### 5.2.2 El Catàleg de serveis

Però la part més interessant de l’aplicació, i la seva raó de ser, és el catàleg. Com ja hem dit abans, la idea de fer aquesta aplicació sorgeix de la necessitat d’automatitzar el catàleg de serveis web de l’empresa, i aprofitant que es realitzava l’aplicació, es volia donar un rentat de cara a aquest. Per tant, el catàleg està dividir en tres apartats molt diferenciats: l’organització per metamodel i el formulari de cerca; les cerques preprogramades i el catàleg de serveis.

Pel que fa a la organització per metamodel i pel formulari de cerca, es va optar per fer una petita organització segons els elements guardats a la taula metamodel, i de forma molt visual podem filtrar els serveis pel metamodel que ens interessi. També, sempre que vulguem, podem utilitzar el formulari de cerca, que consta d’un camp de text des d’on podem buscar per nom de servei o de sistema d’informació.



Com podem veure, a la dreta hi ha un botó que destaca, ja que tota l’aplicació és de color blau i aquest botó és en verd. Això va ser petició del client, que volia que el botó per oferir un nou servei destacarà per sobre de la resta. L’elecció de quin metamodel té entitat suficient per aparèixer de forma destacada en aquest filtre corre a càrrec del client.

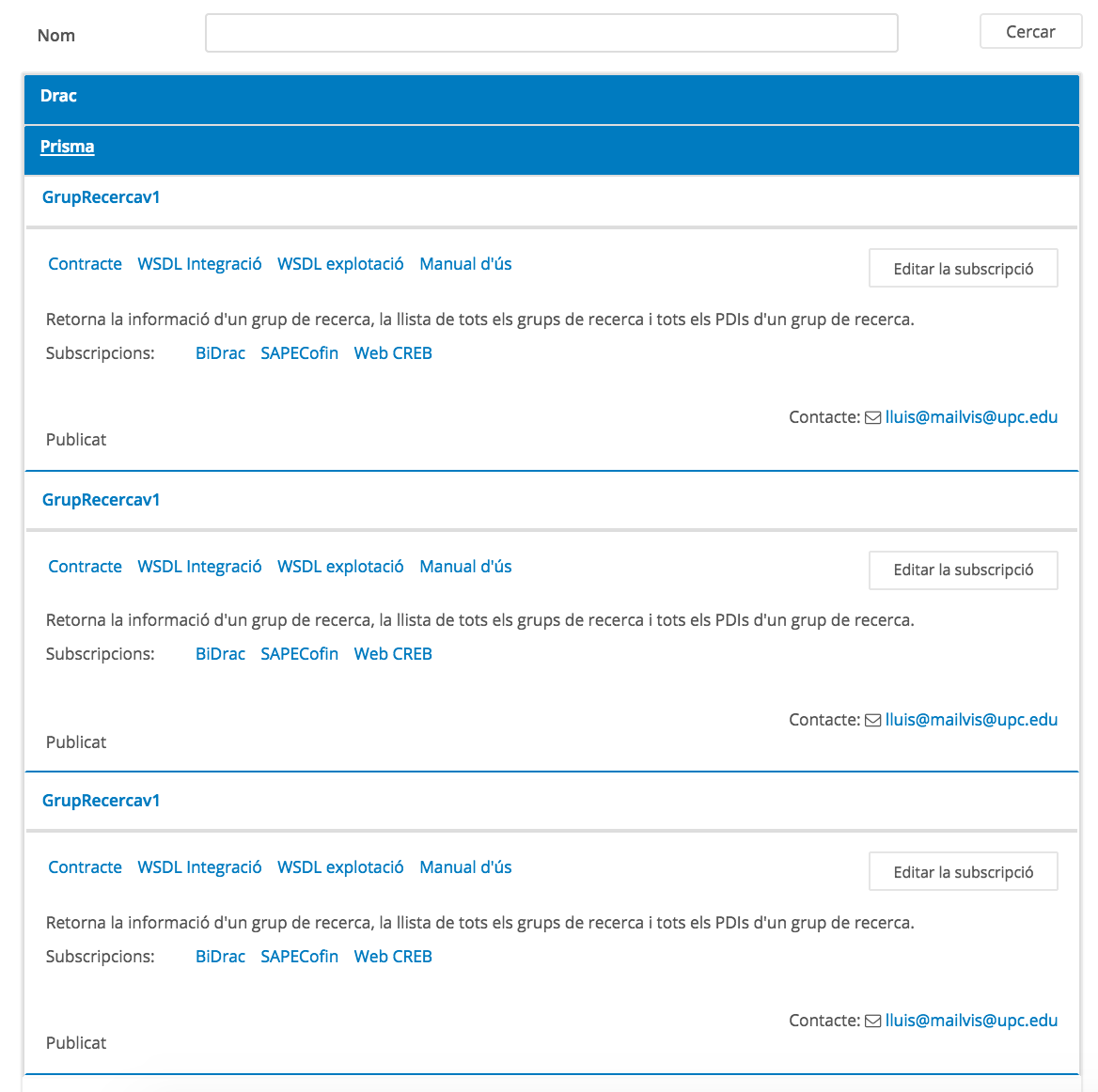
Pel que fa a la part dreta del catàleg, tenim una sèrie d’enllaços, organitzats per categoria, a on trobem diferents opcions per les quals filtrar el catàleg. Podem veure que les etiquetes, a més, tenen entre parèntesis els cops que surten. D’aquesta manera, encara que només n’ensenyem cinc, podem comprovar que son les que més cops estan relacionades amb els serveis. Les etiquetes, en principi, corren completament a càrrec de l’usuari, és a dir, no hi ha un grup d’etiquetes predefinides. L’únic tractament recomanat que es fa és posar-les totes en minúscula, ja que d’aquesta manera podem evitar que hi hagi categories repetides però diferents per les majúscules i minúscules. Com que hi haurà una revisió manual de l’administrador abans de la publicació del servei, es pot evitar etiquetes que no tinguin relació amb el servei que s’ofereix, o malsonants.

El client va trobar interessant ensenyar els últims serveis afegits al catàleg, ja que això fa que sembli que hi hagi molt de moviment. D’aquesta manera, els usuaris poden pensar que sempre s’estan afegint serveis, sense que sigui necessàriament així.

És important també el format de sortida dels serveis, ja que a l’usuari que es subscriu a un servei pot ser-li útil buscar aquells serveis que responen d’una forma determinada i no d’una altra, ja sigui per la tecnologia que tenen instal·lada ells per rebre la resposta, per conveniència...

Per últim, podem veure una classificació dels metamodels, molt semblant a la que hi ha a la part principal del catàleg, però a on surten tots els metamodels del catàleg.

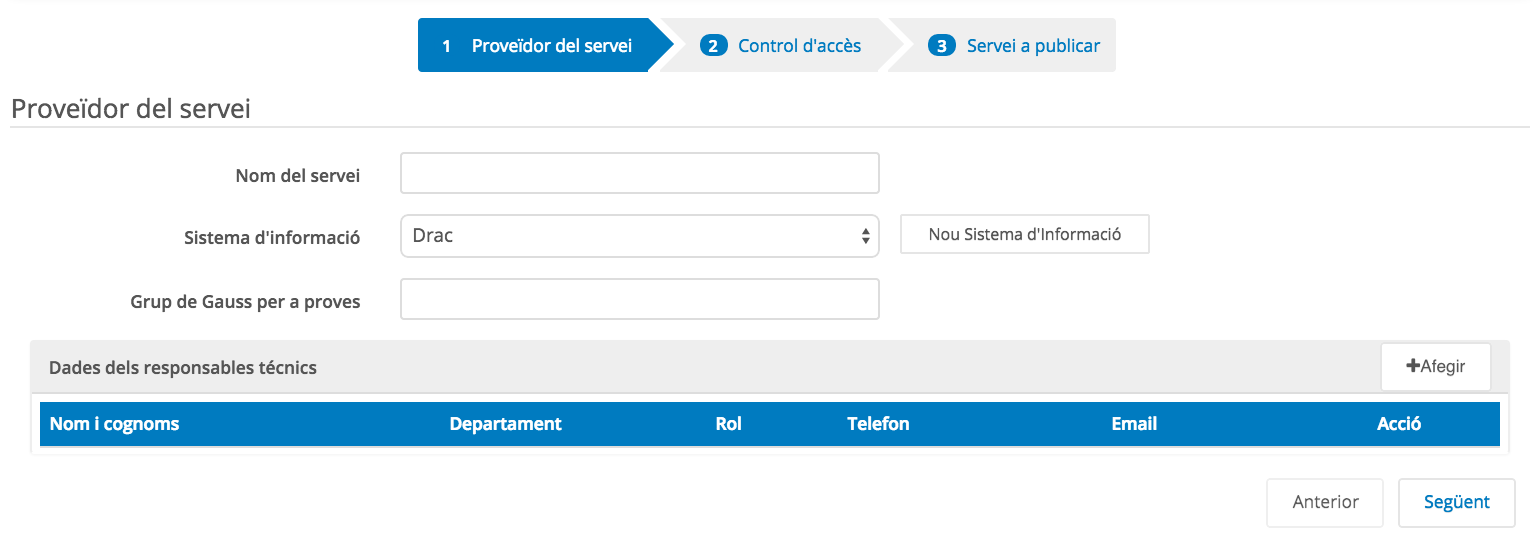
Tant aquests filtres com el filtre de metamodels es comporten de la mateixa manera: només s’ha de fer clic a sobre d’una entitat per a que el catàleg filtri, automàticament, per aquesta entitat. No funciona així, però, la cerca pel nom, situada al camp de text. Com podem veure, al final hi ha un botó que arrenca el procés de cerca, un cop l’usuari ha acabat d’escriure.



La part del catàleg que té a veure amb els serveis, és igual que allò que ja hem vist a l’apartat ‘Els meus serveis’, amb la diferència que en aquest cas, surten o bé tots els serveis sense tenir en compte la persona que està identificada a l’aplicació (realment, el servei és l’única pantalla que es pot veure sense estar identificat), o bé els resultats d’una possible cerca feta pels usuaris del catàleg. No hi ha res més a dir d’aquest apartat. Als annexos es pot veure un exemple de pantalla del catàleg sencer.

### 5.2.3 Els formularis d’alta de servei i de subscripció

Les pantalles amb els formularis de proveïment d’un nou servei i de subscripció a un nou servei també sofreixen canvis considerables. Actualment, hi ha un document en format PDF per a cada formulari que la persona que vol proveir un servei o subscriure’s a un s’ha de descarregar i omplir manualment, i després entregar-ho a mà o enviar-lo per fax a la seu de l’empresa, on s’introduiran les dades manualment. Ara, a més de reduir camps (s’ha fet un rentat de cara també als formularis), el formulari s’ha programat en un *Wizard*, un tipus de formulari molt més user-friendly, ja que en comptes d’haver de veure tot el formulari fent scroll amb el ratolí, es poden anar passant pantalles. A la imatge podem veure la primera pàgina del formulari d’alta d’un nou servei, a on veiem a la part de dalt el menú del *Wizard*, i a la part de sota els botons per navegar per ell:



Com podem veure, el botó ‘Anterior’ té un altre color: aquest color vol dir que està desactivat, ja que al ser la primera pàgina no hi ha pàgina anterior, tot i que el fet de mostrar el botó és important, ja que fa que l’usuari tingui consciència des del primer moment que pot navegar cap endavant i cap endarrere sense problemes.

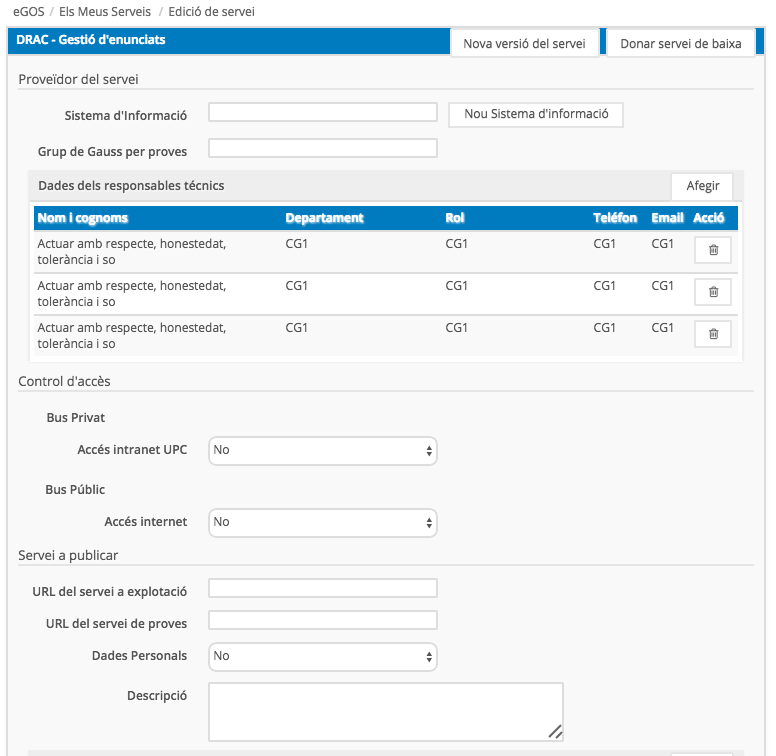
Els botons de l’última pàgina, però, son diferents. L’usuari pot anar cap endarrere o guardar el formulari, ja que no hi ha pàgina següent. Es va decidir amb el client que era millor canviar el botó ‘Següent’ pel botó ‘Guardar’, en comptes de mostrar també el botó ‘Següent’, ja que d’aquesta manera el formulari sempre té dos botons.



Tant la navegació a través del formulari mitjançant els botons de la part de sota del formulari, com el Wizard en sí, estan programats a la part del client amb jQuery.

### 5.2.4 Edició del servei i la subscripció i nova versió del servei

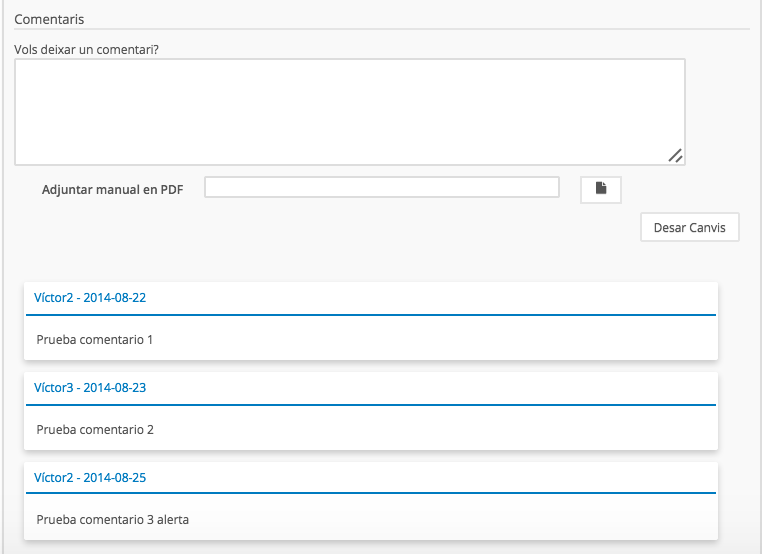
Per últim, tenim les pantalles d’Edició del servei i de la subscripció, i la pantalla de nova versió del servei. Les pantalles d’edició del servei i de la subscripció son iguals, només canvien els camps que hi surten. Aquestes dues pantalles consten de dos parts: la part de les dades, i la part dels comentaris. A la part de les dades podem veure tots els camps del formulari d’alta plens, i un a sota de l’altre, per a que sigui més ràpid veure’ls tots.



Aquests camps estaran bloquejats pels responsables del servei sempre i quan l’estat del servei no sigui Esborrany, ja que el client no veu adient que el responsable pugui canviar sense avisar a ningú dades necessàries pel bon funcionament del servei, un cop aquest ha estat publicat al catàleg. Els canvis es demanaran via comentaris.

Per contra, l’administrador de l’aplicació podrà modificar tots els camps, i actuarà després d’una petició de canvi del responsable als comentaris, sempre i quan aquest canvi no alteri el funcionament del servei.

La part dels comentaris consta primer d’una àrea de text a on l’usuari podrà posar el seu comentari, i un input a on podrà adjuntar un document (per exemple, un Manual d’ús, necessari pel servei, per si el vol canviar). També hi haurà, per suposat, el botó per guardar el comentari. Tot seguit, hi trobarem els comentaris previs, a on podrem veure informació com per exemple la data del comentari, l’usuari que l’ha penjat, i el comentari mateix.

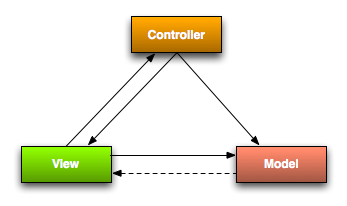


A la pantalla de nova versió d’un servei hi podem accedir des de la part de dalt d’aquesta mateixa pantalla, clicant al botó de ‘Nova versió d’un servei’. La pantalla és la mateixa que la d’edició de servei, però amb els camps omplerts i modificables pel responsable, i sense l’apartat de comentaris. D’aquesta manera, com que el servei tindrà molts dels valors iguals que el servei anterior, facilitem al client la posada en marxa de la nova versió.

Aquestes son totes les pantalles que hi ha a l’aplicació. Als annexos es pot trobar una captura de pantalla de totes aquestes, per si hi ha qualsevol dubte.

## 5.3 Back-End

A l’hora de programar el BackEnd de l’aplicació, hem de tenir en compte el patró de disseny que s’ha decidit utilitzar per a l’aplicació, i pel qual Spring Framework està preparat: el Model Vista Controlador (MVC a partir d’ara). Aquest patró de disseny, que podem veure en el següent esquema:



Té tres parts plenament diferenciades:

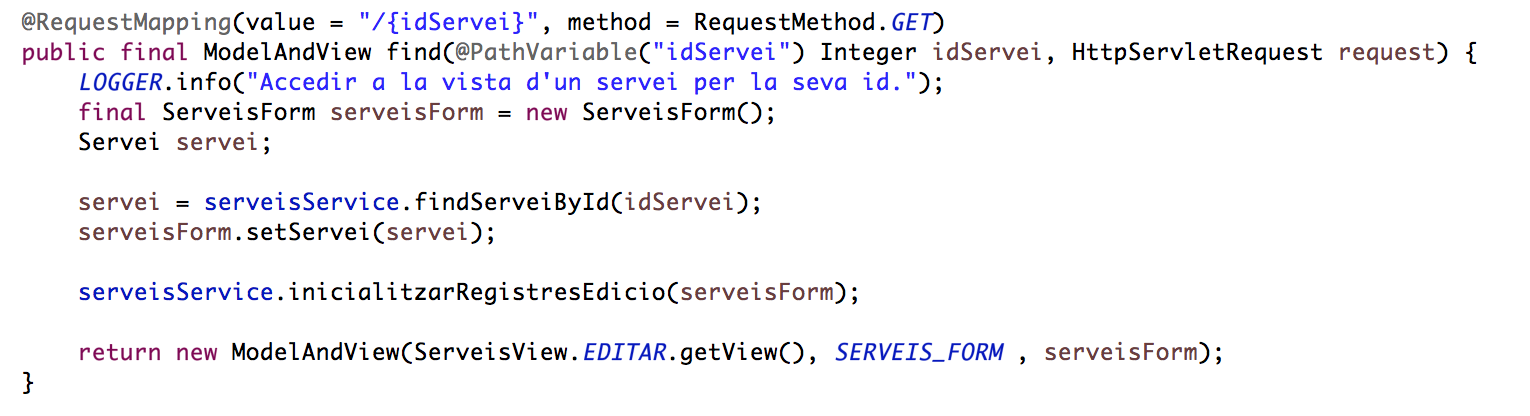
* El **model**: El model de dades, la representació de la informació que tenim guardada a la nostra base de dades. Com veiem a l’esquema, envia la informació que la vista demana per ensenyar-la a l’usuari, però en pot enviar al controlador i rebre’n d’aquest (per actualitzar informació, per exemple).
* La **vista**: Allò que, finalment, ha de veure l’usuari.
* El **controlador**: la lògica de negoci de l’aplicació. Rep la petició web des de la vista per organitzar el contingut utilitzant les dades del model, empaquetant-lo, i enviar-lo a la nova vista. També pot enviar dades al model per actualitzar la base de dades.

Durant aquest apartat, parlarem del controlador i del model, i la relació d’aquests amb el format acceptat per Spring Framework.

### 5.3.1 Controlador

El Controlador en l’aplicació està dividit en dues parts, per tal que sigui més senzill implementar i mantenir la lògica de negoci. Tenim les classes anomenades ‘Controller’, una per interacció principal amb la base de dades (per tant, tindrem un controller pel catàleg, un pel servei, un per la subscripció i un per l’usuari).

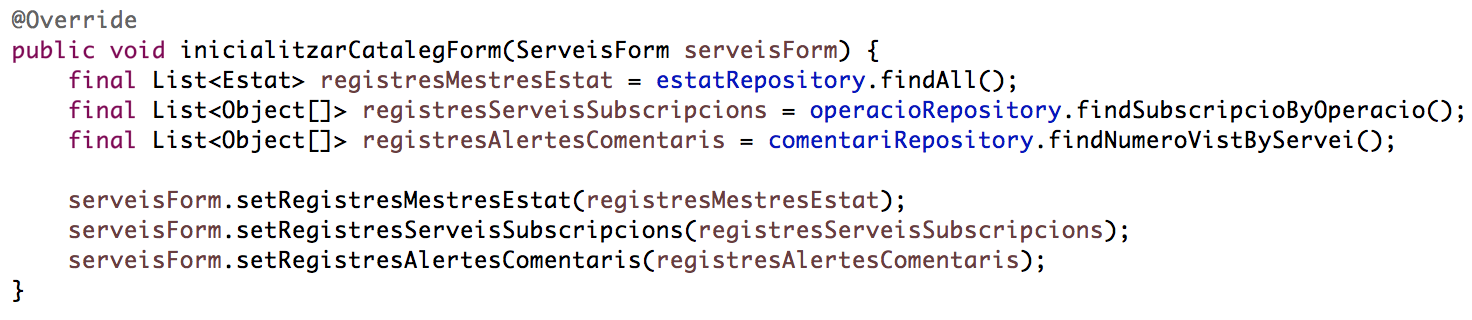
Aquests controllers reben de la vista les peticions webs que l’usuari demana gràcies a una sèrie d’etiquetes d’Spring:



Gràcies a l’etiqueta RequestMapping, podem veure que quan entra una URL amb un valor de tipus integer corresponent a l’id del Servei, sempre i quan aquesta petició sigui del tipus GET, passarà per aquesta funció. Amb aquesta funció, omplirem la vista d’edició d’un servei amb les dades del servei amb la id que ens arriba per URL.

Podem veure un objecte del tipus ServeisForm, que conté totes les dades d’un servei, tan per rebre-les de la vista com per enviar-les a aquesta. Aquest mateix objecte, doncs, l’utilitzarem tant per omplir la vista de les dades del model de dades, com per rebre els valors dels formularis.

Per últim, tenim els serveis. Els serveis son les classes que interactuen directament contra el repositoris, és a dir, dins de l’objecte ServeisService és on trobarem els mètodes per obtenir les dades de la base de dades. Podem veure un exemple de mètode d’un servei aquí:

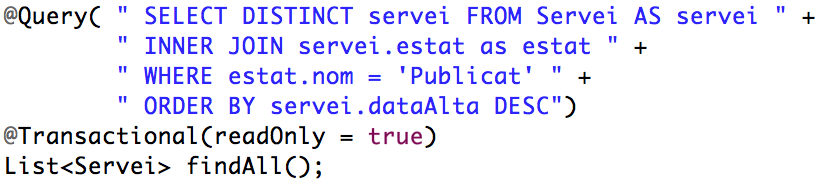


Aquest mètode d’exemple serveix per inicialitzar els registres mestres del catàleg. Com podem veure, utilitzem els repositoris per realitzar les consultes a la base de dades, i allò que ens retorna, ho guardem a l’objecte ServeisForm, que després enviem a la vista.

Spring permet accedir a la vegada a diferents tipus de base de dades (Oracle, PostgreSQL, MongoDB...) a on no necessàriament han de coincidir els mètodes. Per tant, en aquesta aplicació creem una interfície (anomenada, seguint l’exemple, ServeisServiceBD) a on declarem tots els mètodes. Però només és la declaració, ja que la implementació d’aquests es fa als serveis (i per això l’etiqueta Override, que sobrecarrega la declaració del mètode).

### 5.3.2 Model

Gràcies a Hibernate, treballar amb el model de dades és molt fàcil. En aquesta aplicació, utilitzarem una interfície amb un repositori de consultes que necessitarem per a l’aplicació (per exemple, ServeisRepository). Com ja s’ha explicat a les eines que s’utilitzaran durant aquest projecte, Hibernate ens permet fer consultes senzilles sense necessitat de programar les consultes a la base de dades completament. Però si volem fer consultes una miqueta més complicades, podem utilitzar el llenguatge HQL que ens ofereix Hibernate. També podem sobreescriure consultes que Hibernate ens proporcioni, com al següent exemple:

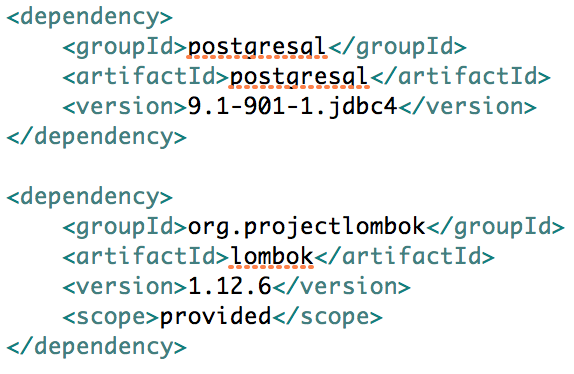


Aquesta consulta, com podem veure, es diu findAll(). Hibernate ja ens proporciona aquesta consulta, sense necessitat de programar-la. Però nosaltres realment no volem tots els serveis, volem tots els serveis que tinguin com estat ‘Publicat’, a més de tenir-los ordenats d’una forma molt específica. Per tant, la millor opció per nosaltres era realitzar de zero la consulta.

Pel que fa a la persistència, Hibernate ens ofereix el mètode save(). Aquest mètode, existent a tots els repositoris, ens permet guardar dades d’una entitat (fent una actualització si la clau primària existeix, creant un nou camp si no) utilitzant les declaracions d’aquestes entitats per saber quins camps hi han d’haver, el seu tipus, i quins són obligatoris i quins no. En cas que el fet de guardar les dades no hagi funcionat bé, se’ns retornarà una excepció que podrem gestionar, tal i com veurem a l’apartat seguretat d’aquesta memòria.

### 5.3.3 Gestió de les llibreries i dependències

Com que aquest és un projecte basat en Maven, la llista de dependències i llibreries que necessita el projecte les trobarem al fitxer pom.xml. Dins d’aquest fitxer hi trobarem entrades bàsiques, com el nucli d’Spring, i dependències de tercers, com poden ser Lombock o Postgresql. El format per afegir aquestes dependències és el següent:



D’aquesta manera, quan Jenkins (o nosaltres mateixos) obrim el projecte en una altra màquina (Jenkins l’obre al servidor, per exemple), només ha de descarregar les dependències seguint l’estructura de Maven (en aquest cas, es tractaria d’utilitzar la instrucció ‘maven clean package install’ en una terminal de Linux) i després es podrà executar l’aplicació sense problemes i sense haver de tenir sempre descarregat a la màquina local totes les dependències de l’aplicació.

## 5.4 Front-End

A l’hora de programar el FrontEnd de l’aplicació ( és a dir, la vista del MVC ), es van haver de prendre algunes decisions, sobretot pel que respecta al format de les pàgines, i a la gestió dels formularis. Pel que fa al format de les pàgines, es va decidir utilitzar Thymeleaf, ja que és molt més natural a l’hora de programar que no pas JSP, que és el que s’utilitza normalment per a aplicacions basades en Java. Pels formularis, com ja s’ha explicat en capítols anteriors, es va decidir muntar un formulari de tipus Wizard, que tingués tres o quatre pàgines per tal que a l’usuari final tot li semblés molt més curt del que li sembla en la situació actual, on ha d’imprimir 6 o 7 pàgines d’un PDF i omplir-les a mà. Per últim, a l’hora de gestionar dinàmicament aquests formularis, s’ha utilitzat la llibreria Handlebars, que no és una altra cosa que gestionar els elements que s’afegeixen al DOM de forma dinàmica a través de jQuery d’una forma elegant. Tot seguit, s’entrarà una miqueta més en detall de com s’han utilitzat aquestes eines.

### 5.4.1 Thymeleaf o JSP?

En el seu moment, vam haver de decidir de quina manera es programava el FrontEnd de l’aplicació: o bé es feia utilitzant el que es venia utilitzant a l’empresa, les JSP (Java Server Pages), o bé canviàvem i començàvem a utilitzat Thymeleaf. Hi ha una sèrie d’avantatges en utilitzar Thymeleaf, que explicarem ara mateix, i per això al final ens vam decidir per aquesta nova llibreria.

Primer de tot, una aplicació en Thymeleaf consta de pàgines amb extensió HTML, i no JSP. Això és possible gràcies a que utilitzant Thymeleaf tota la gestió del contingut es fa a través de comandes acceptades per HTML, i no s’han de fer servir etiquetes estranyes JSP o sintaxi que no correspon a HTML.

També s’ha de tenir en compte el sistema de gestió dels layouts que ofereix Thymeleaf. El sistema de gestió dels layouts consta d’una pàgina esquelet estàndard (layouts.html, en aquest cas), a on es programarà tot el que és comú a totes les pantalles de l’aplicació (normalment, la icona de l’aplicació, el menú d’usuari, la barra de navegació i el footer) i permet afegir el contingut específic de la pàgina a on volem estar de forma dinàmica. També permet afegir scripts Javascript o pàgines CSS comunes a tota l’aplicació una sola vegada.

La part principal del nostre layouts.html té la següent forma:



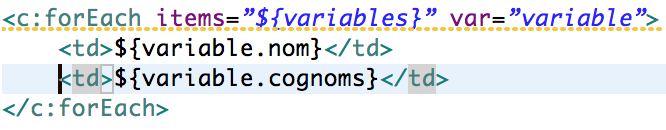
Com podem observar, totes les pàgines constaran de 4 elements: el header, el menú, el contingut i el footer. Tant el header, com el menú, com el footer, estan inclosos a la carpeta ‘fragments’ del nostre sistema de fitxers. Per posar un exemple, la pàgina amb el menú de l’aplicació és d’aquesta manera:



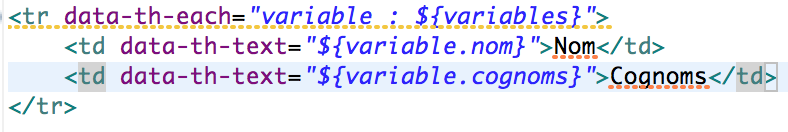
A la segona línia, veiem que ja s’identifica aquesta pàgina com un fragment de nom menú. Per tant, les pàgines més interessants, aquelles que son del tipus container, hauran de començar totes amb un data-th-fragment=’container’, que identificarà aquesta pàgina amb el que volem que sigui.

Aquest exemple ens serveix també per ensenyar alguns dels elements més típics de Thymeleaf: les expressions amb coixinet, que representen textos amb internacionalització, i les variables amb arrova, que representen canvis a la URL. També podem veure que, tot i que tenim atributs data-th-text, mantenim el text hardcoded a on en principi ha d’estar si no tinguéssim aquest atribut de Thymeleaf. Això passa perquè utilitzem una altra de les funcionalitats de Thymeleaf: la capacitat d’aquest de funcionar com a maqueta tot i no ser ja una maqueta. Si en un moment donat ens quedéssim sense connexió a Internet, o sense connexió amb la base de dades, en comptes de mostrar un error, Thymeleaf ens ensenyarà el text que es veu ara mateix.

Una altra de les característiques de Thymeleaf és la facilitat per controlar el flux de l’execució de l’aplicació, i lo ben integrat que queda en el codi. Mentre que utilitzant JSP hem d’utilitzar etiquetes com aquestes per recórrer una array i, per exemple, ensenyar totes les dades d’aquesta array:



A on ‘variables’ son les dades que obtenim del nostre controlador, i ‘variable’ és la variable que utilitzem per operar Utilitzant Thymeleaf, la cosa és molt més senzilla:

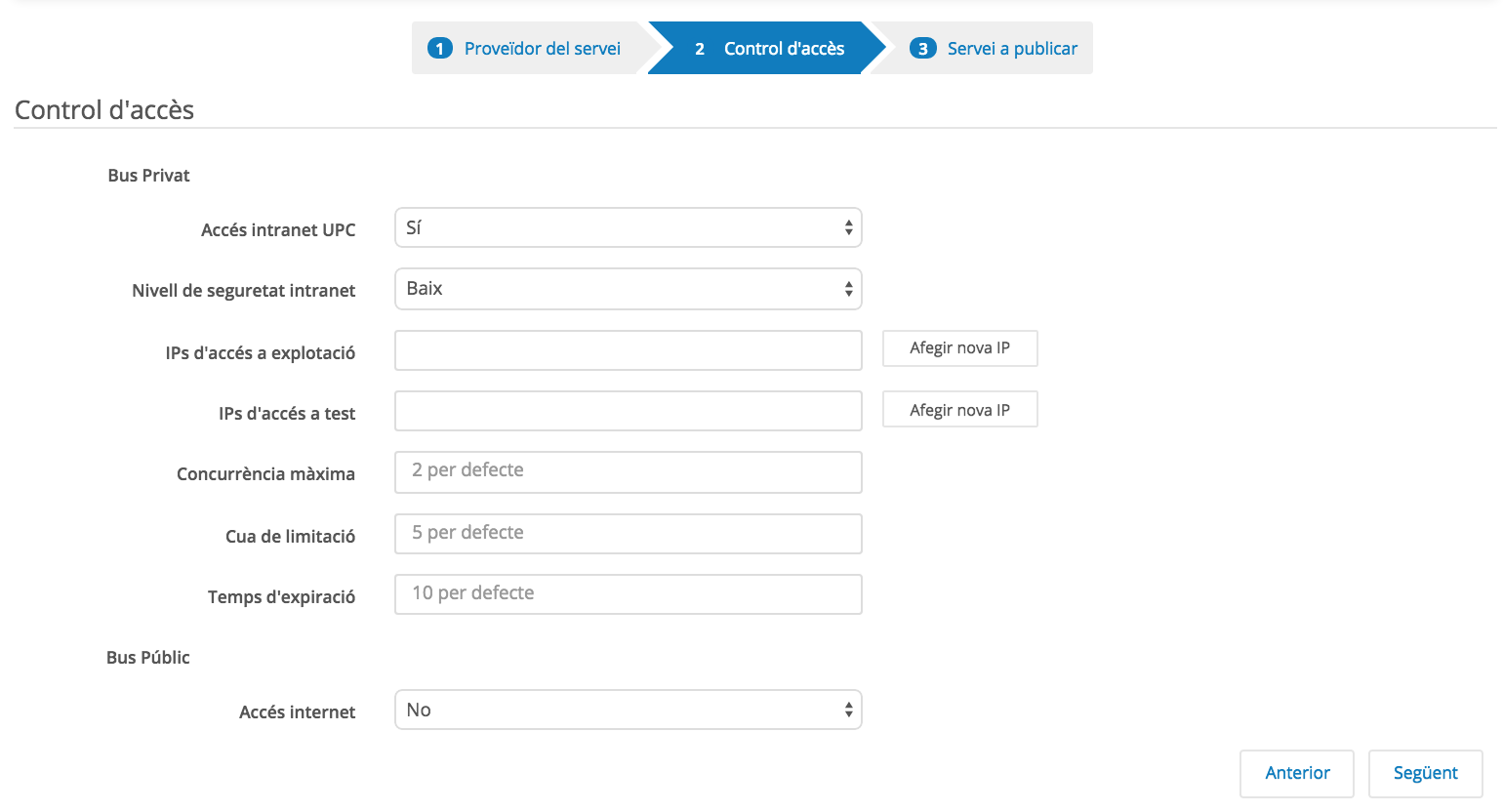


Com podem veure, tots els atributs de Thymeleaf van precedits per l’atribut d’HTML5 ‘data-‘. En aquest exemple, podem veure una nova forma de tractament de variables Thymeleaf: ‘${}’, molt semblant al tractament de les variables procedents del servidor de JSP. També podem tornar a veure que hem escrit ‘Nom’ i ‘Cognoms’ quan el que volem és ensenyar aquests atributs des de la variable. Això, com ja hem dit abans, serveix com a maqueta per a quan no hi hagi una connexió que funciona.

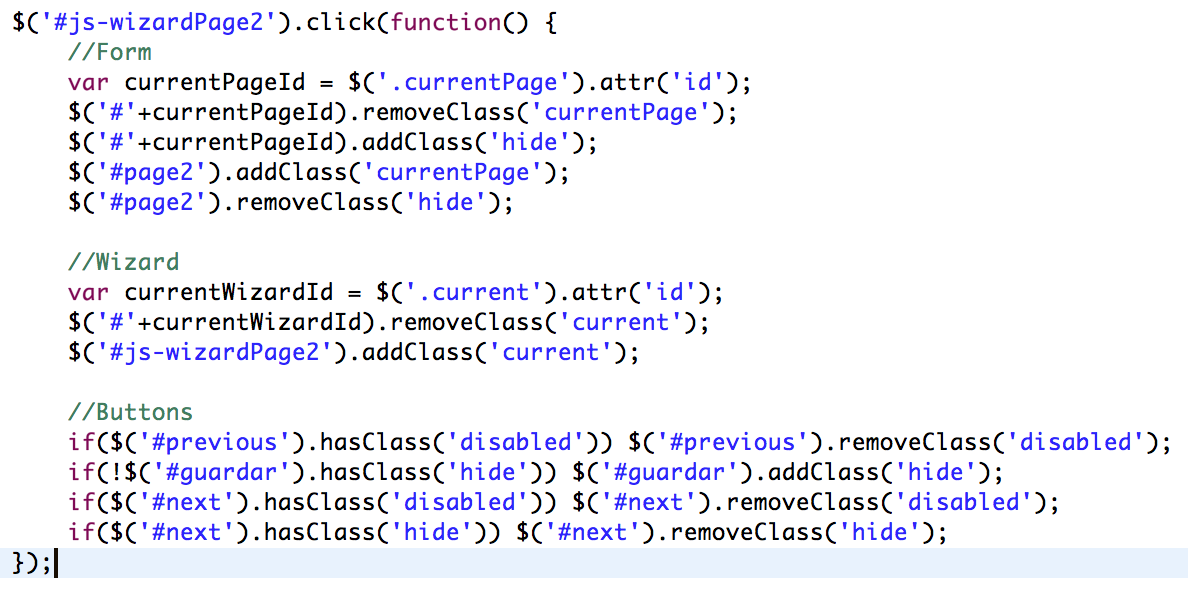
A Thymeleaf també podem utilitzar elements de control del flux de l’aplicació condicionals (if/else), d’una manera similar a com s’utilitzen els bucles.

### 5.4.2 Formularis i Handlebars

Els formularis, tal com s’ha dit en capítols anteriors, estaran formats per Wizards que permetran amagar una mica d’informació de cop per no espantar l’usuari que haurà d’omplir-los tant per donar d’alta un servei com per subscriure’s a un. Gràcies a l’ajuda de jQuery, és molt fàcil gestionar aquests Wizards. Per exemple, per passar de la segona pàgina a la tercera del formulari d’alta d’un nou servei:



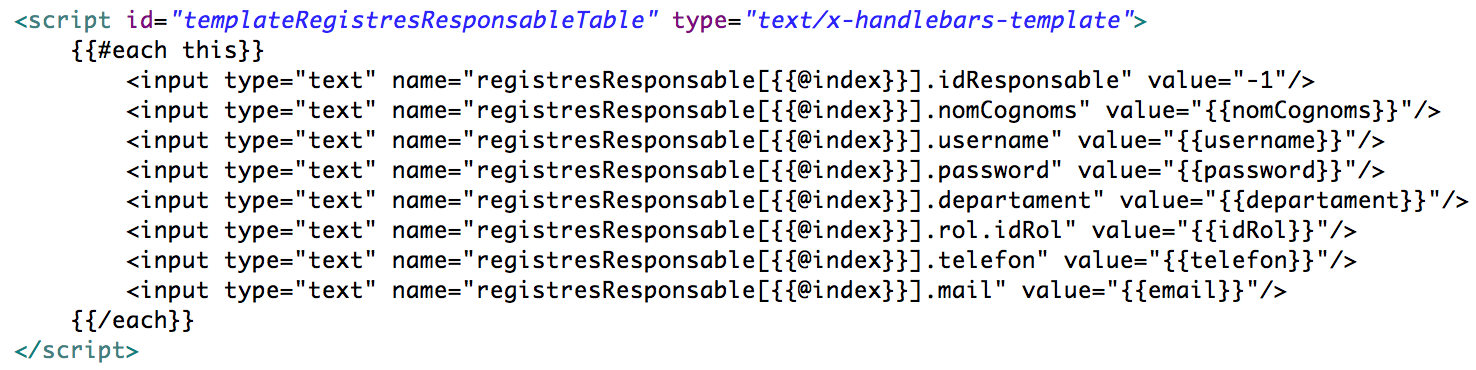
Podem veure que tenim els dos botons a la part de sota, ‘Anterior’ i ‘Següent’, i les categories del formulari a la part de dalt, des d’on podem també navegar. Al nostre jQuery, prèviament haurem d’haver definit el comportament d’aquestes interaccions, ja que haurem d’amagar o mostrar una part o unes altres del formulari, fent que quedi la resta amagada i que l’usuari no se n’adoni de res.



Aquest és el codi per gestionar la segona pàgina via jQuery. Com que la tercera pàgina és l’última, amaguem el botó ‘next’ i ensenyem el botó ‘guardar’. També, a les parts de Form i Wizard (que es veuen com a comentari), gestionem la part del Wizard, perquè en cap moment es perdi a quina pàgina estem, per marcar d’uns colors o uns altres el fil del Wizard, o per a que si fem clic a ‘Anterior’ o ‘Següent’ mostri la informació correcte.

Un cop s’han omplert tots els camps, podem veure que hi ha taules que s’han d’omplir dinàmicament. Si volem guardar després aquestes taules, com s’omplen de dades que encara no tenim a la base de dades, no es poden guardar com els altres camps. Per guardar-ho tot correctament, haurem d’utilitzar el sistema de plantilles JavaScript anomenat Handlebars.

Els Handlebars permeten afegir blocs de codi HTML des d’una plantilla, escrits en HTML i que s’ompliran de dades que enviarem mitjançant jQuery en format JSON. Una plantilla d’exemple Handlebars, per omplir per exemple la taula de dades dels responsables, seria aquesta:



Com podem veure a la imatge, cada entrada de la taula (variable ‘this’) conté un índex, que correspondria a l’índex de l’array que després voldrem recuperar al controlador, i s’afegeixen els camps que s’han anat omplint.

## 5.5 Seguretat

Quan es parla de seguretat, en aquest cas, es parla de dues coses diferents:

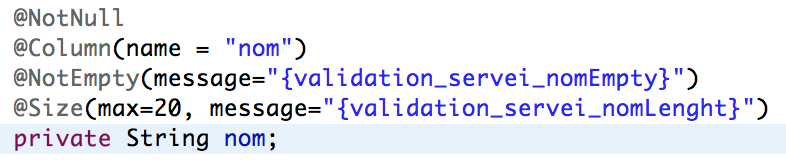
* La seguretat referent al manteniment de les dades, és a dir, a que la introducció de les dades a la base de dades es faci de la forma correcte.
* La seguretat referent a la identificació dels usuaris a l’aplicació.

Durant aquest apartat, es parlarà dels dos casos, ja que els dos s’han tingut en compte.

### 5.5.1 Gestió de les dades

Pel que fa a la gestió de les dades, tenim 3 tipus de nivells de seguretat, que actuen com una piràmide: a la base, la seguretat més elemental de totes, és la que es fa a la base de dades mateixa. És aquí on podem decidir quins camps poden estar repetits i quins no, quins camps poden ser null i quins no, i també podem escollit el tamany de, per exemple, els camps de text. Gràcies a Hibernate, qualsevol error dins d’aquesta capa de seguretat es fàcil de gestionar gràcies a les excepcions de Java.

En un segon nivell de seguretat, tenim el que es fa al servidor abans d’enviar les dades a la base de dades. Aquí ja es fan tota una sèrie de comprovacions de forma molt senzilla, i en cas de detectar qualsevol cosa que no sigui el que ha de ser, enviarem a la vista un missatge d’error. Aquí podem comprovar, com es pot veure a la imatge a continuació d’aquest paràgraf, la llargada dels camps de text, els camps que poden ser nuls.



Aquests tipus de comprovacions, com podem veure a la foto, no son difícils d’implementar, i tot i que son molt semblants (per no dir pràcticament iguals) que les comprovacions que es fan a base de dades, sempre és millor fer-les primer al servidor, i no haver d’arribar a la base de dades. També podem comprovar que un email estigui escrit de forma correcte gràcies a l’anotació @Email de Hibernate, o comprovar que els camps de tipus Date tinguin un format adequat amb l’anotació @DateTimeFormat.

Per últim, i a la part de dalt de la piràmide, tenim les comprovacions que hi ha a la part del client. Aquestes comprovacions les farem a dos llocs diferents: als JavaScript i a l’HTML. Dins del JavaScript, podem comprovar coses com per exemple que tots els camps que han d’estar obligatòriament omplerts ho estiguin o que les dates i els emails tinguin el format correcte.

A nivell HTML, podem comprovar poques coses, com per exemple la llargada dels camps de text. Gràcies, però a HTML5, podem validar ja a HTML camps com els emails.

El principal problema de la seguretat a la part del client és que aquestes poden ser desactivades fàcilment per un usuari malintencionat. Es per això que son necessaris els altres dos nivells de seguretat. Combinant els tres nivells de seguretat d’aquesta piràmide, ens podem assegurar que les dades que introduïm a la base de dades siguin tot lo correctes que ens permeti un sistema obert als usuaris.

### 5.5.2 Identificació d’usuaris

[[ TODO: FALTA PARLAR DE LA IDENTIFICACIÓ D’USUARIS ]]

# 6 Qualitat del codi

En una aplicació com aquesta, mantenir la qualitat del codi és molt important. El fet de fer-la en Java, ens permet utilitzar una sèrie d’eines per a mantenir la qualitat del codi es mantingui, tot i que la mateixa persona no estigui al mateix projecte. Tenim dos nivells a on utilitzar les eines que ens ofereix tant Eclipse com Java: els tests unitaris, que ens permeten mantenir els resultats de les funcions tot i haver de canviar aquestes per noves peticions del client en el futur, i les proves d’integració amb Jenkins, que amb l’ajuda de Maven ens permet després veure els resultats de la nostra aplicació a l’eina SonarQube.

## 6.1 Tests Unitaris

Hi ha diferents tipus de tests que es poden fer dins d’una aplicació Java. Per a aquest projecte, però, s’ha decidit realitzar tests unitaris. Aquest tipus de proves es diuen unitàries perquè proven una funcionalitat única de l’aplicació. Normalment, com és aquest cas, es fan en un entorn controlat (amb una base de dades fictícia, però que s’assembla a la de veritat), i amb la llibreria Mockito podem fins i tot simular el retorn d’algunes funcions que no volem provar però que, de totes formes, necessitem per comprovar el funcionament d’una altra funció.

[[ TODO: FALTEN EXEMPLES DE TESTS UNITARIS ]]

## 6.2 Integració (Jenkins)

Jenkins és un software d’integració contínua, escrit en Java, i que es té molts plugins que el doten d’una gran quantitat de funcionalitats. Aquest servei agafa el nostre projecte directament des de GitHub (podem dir-li, fins i tot, la branca d’on l’ha d’agafar, així podem comprovar la branca que vulguem, o diferents branques a la vegada. Un cop despleguem aquest serveis, que també ens executa els tests unitaris, si el resultat es que tot està correcte, podrem dir que tenim una versió ‘estable’.

Després, Jenkins envia els resultats a SonarQube, a on podrem comprovar realment si podem millorar la qualitat del nostre codi, i fer que aquest tingui un manteniment molt més assequible per a la nostra empresa.

[[ TODO: FALTA FER LES PROVES D’INTEGRACIÓ CONTÍNUA AMB TESTS ]]

## 6.3 Anàlisi del codi (SonarQube)

Un cop feta la integració contínua, Jenkins envia els resultats a aquesta eina, SonarQube, a on podrem realitzar una sèrie de millores al nostre codi, i després veure una nota final (purament informativa, però que serveix per comparar aplicacions entre si) desglossada per diferents apartats (arquitectura, tests, errors...) que ens pot donar indicacions per veure a on la podem millorar.

[[ TODO: FALTA PARLAR DE SONARQUBE ]]

# 7 Conclusions

[[ TODO: FALTEN LES CONCLUSIONS ]]

# 8 Fonts d’informació

[[ TODO: FALTEN LES FONTS D’INFORMACIÓ ]]

# 9 Annexos

[[ TODO: FALTA AFEGIR ELS ANNEXOS ]]