Títol: Disseny i implementació d'una interfície web amb Symfony

Volum: 1

Alumne: Pablo Prieto Barja

Director/Ponent: Xavier Messeguer Peypoch

Departament: LSI

DADES DEL PROJECTE

Títol del Projecte: Disseny i implementació d'una interfície web amb symphony

Nom de l'estudiant: Pablo Prieto Barja
Titulació: Enginyeria Tècnica en Informàtica de Gestió
Crèdits:22,5
Director/Ponent: Xavier Messeguer Peypoch
Departament: LSI

MEMBRES DEL TRIBUNAL *(nom i signatura)*

*President:*Jorge Castro Rabal

Vocal: Manel Canales Gabriel

Secretari: Xavier Messeguer Peypoch

QUALIFICACIÓ

Qualificació numèrica:
Qualificació descriptiva:

Data:

Índex

1 Introducció al projecte

1.1 Motivació

L'elecció de quin projecte realitzar no va ser senzilla, el primer lloc on vagi buscar va ser la borsa de ofertes de projectes de la FIB. Tot i que hi ha una gran quantitat de projectes disponibles no vaig saber veure un que em motivés especialment al llegir la seva descripció. No penso que fos pels temes que es tractaven en alguns, sinó més bé per la presentació que se'ls dóna. Em vaig plantejar tenir alguna entrevista entre alguna de les propostes que apareixien per informar-me millor sobre aquestes ja que potser amb informació de primera mà i més detallada podria haver-me interessat per algun.

Abans de realitzar cap entrevista amb els responsables dels PFC's de la borsa de la FIB vaig recordar que en una assignatura optativa que havia cursat, Recuperació de la Informació, el professor Xavier Messeguer havia parlat de les col·laboracions que havia tingut amb alumnes per dur a terme projectes de final de carrera, i ens va instar a que quan el volguéssim realitzar ens pudíem posar en contacte amb ell per presentar alguna idea o oferir ella alguna. Tanmateix, a classe va ensenyar algunes aplicacions que havia realitzat amb alumnes de la facultat com a projecte de final de carrera per exemplificar aquestes col·laboracions amb els estudiants.

Com la temàtica de la part de l'assignatura Recuperació de la Informació on fa de professor em va atreure, vaig pensar en posar-me en contacte amb ell per saber si podia tenir alguna proposta que jo pogués realitzar com a projecte. El dia de l'entrevista em va plantejar diverses opcions, una d'elles era posar-me en contacte amb altres persones relacionades amb temàtiques de bioinformàtica per poder realitzar algun projecte que tinguessin, ja que en aquells moments no tenia una oferta pròpia per realitzar un projecte, però si que em va plantejar la possibilitat de unir-me al projecte que estava desenvolupant actualment.

En aquest projecte es trobava alguna de les aplicacions que havia pogut arribar a veure en la classe que ens va parlar dels projectes de final de carrera. Les aplicacions que em va mostrar i que es volien oferir com a serveis dins del projecte em van semblar prou innovadores i atractives. També va influenciar que se m'oferís participar en un projecte que podria tenir el seu reconeixement i amb el que se m'oferia una participació continuada.

1.2 Introducció general

Des de l'aparició de les primeres pàgines web accessibles gràcies a la expansió i reconeixement de l'HTML, els continguts disponibles no han fet més que fer-se cada cop més presents i evolucionar oferint noves capacitats en la navegació i la connectivitat a través de la xarxa.

En un principi la web estava orientada a oferir continguts estàtics que els autors publicaven i s'esperava que els usuaris poguessin accedir i llegir-los. I per part dels usuaris s'esperaven continguts disponibles i sense canvis que es podrien consultar quan sigués necessari gràcies a la seva disponibilitat a la xarxa.

Però des de llavors les aplicacions que tenia la xarxa han evolucionat molt. Els paradigmes en que es basava la web s'han voltejat, fent ús de les característiques bàsiques que es van assentar en els seus inicis enfocats a continguts, es va passar a oferir continguts que poden canviar dinàmicament sense haver d'estar desplaçant-se a través de les pàgines, els usuaris que fins ara s'esperava que accedissin als continguts de les pàgines ara són l'element estrella de les aplicacions web amb més èxit, s'ha produït un canvi enorme en quant a la manera d'utilitzar i interactuar amb l'usuari, possibilitant fer-los actors principals dels continguts i funcionalitats que es presenten.

Un altre aspecte que ha possibilitat aquesta transició és la capacitat de poder portar i posicionar en el núvol aplicacions que fins fa poc només es podien imaginar utilitzades i executades dins dels nostres sistemes operatius sense el suport de la xarxa. Aquest fet es combina avui en dia amb les possibilitats de la web 2.0 que ja s'han descrit oferint entorns per les aplicacions on es poden compartir i enviar informació entre els usuaris eliminant les barreres de connectivitat que podien haver entre aplicacions i la xarxa.

La capacitat per compartir la informació entre usuaris és un altre punt estrella de la web 2.0 d'avui en dia, els usuaris no només poden crear continguts, sinó que utilitzant eines i aplicacions de la web que tenen la motivació d'interactuar amb elles perquè s'obtingui la possibilitat de comunicar-se, organitzar-se, participar i conèixer altres usuaris amb els que compartir informació, gustos, estatus social,...

Perquè s'ha produït aquest canvi? No es que s'hagi produït una renovació en la estructura i tecnologia utilitzades a la xarxa, sinó que ha sigut fruit d'un procés de

renovació i evolució gradual. Ha arribat un moment en el que s'ha arribat a una maduresa per part de les tecnologies, infraestructures i una divulgació prou estesa com per fer possible aportar noves funcionalitats a la xarxa.

1.3 Objectius

Els principals objectius que es buscava complir amb la realització d'aquest projecte són:

- ✔ Masterització del framework pel desenvolupament d'aplicacions web Symfony.
- ✔ Entendre la lògica del mecanisme de serveis de l'aplicació web.
- ✔ Implementació d'un entorn per l'aplicació basat en la relació entre escoles i aules.
- ✔ Implementació d'una política de permisos d'accés pels usuaris.
- ✔ Implementació d'un entorn per la gestió d'escoles i aules.
- ✔ Integració d'aplicacions externes dins de l'aplicació desenvolupada com a serveis.
- ✔ Implementació d'estructura de dades i de gestió que permeti afegir nous continguts als serveis sense la necessitat de realitzar tasques d'adaptació.
- ✔ Implementació d'estructura de dades i de gestió quer permeti diferenciar els serveis en diferents sub-tipus.
- ✔ Corregir els bugs existents.
- ✔ Internacionalització i localització de l'aplicació.
- ✔ Aplicació d'un estil i una identitat pròpia per l'aplicació.
- ✔ Control i estadístiques d'accés.

1.4 Requisits de l'arquitectura

Els requisits necessaris per accedir a l'aplicació poden variar segons els tipus de serveis que hagin disponibles. Per accedir a l'aplicació no caldrà un hardware ni un software potent, però si actualitzat. L'aplicació s'accedeix mitjançant un navegador web, dins dels navegadors que funcionen i visualitzen el contingut de l'aplicació correctament es troben la gran majoria que s'utilitzen avui en dia com Chrome, Firefox, Safari, Internet Explorer. Tot i així s'ha detectat problemes amb alguns navegadors no tan populars com rekonq i konqueror de KDE.

No obstant hi ha un servei que requereix l'ús d'un hardware més potent, concretament la targeta gràfica i la memòria RAM. S'han realitzat proves i les targetes gràfiques integrades i les dedicades amb poca memòria(menys de 128 MB) no estan recomanades, el servei s'executarà però la visualització es veurà afectada sense arribar a veure el contingut correctament o res directament, també es té en compte la potència de processament. En quant a la RAM consumida amb certs serveis de l'aplicació s'ha comprovat que el consum pot arribar al voltant dels 500 MB, amb un giga de RAM hauria de ser suficient (segons el consum del sistema operatiu que s'utilitzi).

L'aplicació i els serveis no precisen d'uns coneixements alts en informàtica ni en cap mena d'aplicació o sistema, l'ús de l'aplicació es basa en esquemes coneguts i estesos com els de les xarxes socials o aules virtuals, només es precisa estar familiaritzat amb la navegació a través de les xarxes. En el cas dels serveis tampoc precisa d'aquesta tipus de requisits, ja que estan pensats per oferir un conjunt d'elements visualment atractius per poder interactuar amb les seves funcions sense dificultat. Com a únic requisit propi de l'usuari, per poder accedir a l'aplicació els usuaris necessitaran un compte de correu electrònic que serà necessari per crear el seu compte i serà la forma amb el que se'ls identificarà com a usuaris.

Els requisits per fer disponible l'aplicació a la xarxa són els següents:

- Servidor Apache on hospedar les aplicacions amb Symfony, amb accés als logs del compte i suport per PHP 5, la versió 4 no proporciona orientació a objectes i és indispensable per l'execució del framework Symfony.

- Servidor MySQL versió 5 per suportar procediments emmagatzemats, i l'accés a diverses bases de dades, amb replicació de les dades en cas de falla.

- Espai en disc dur, la quota d'espai ha de ser prou gran com per emmagatzemar les dades dels serveis que poden arribar a ocupar un quantitat d'espai prou gran (series d'imatges i models 3D), actualment s'ocupen més de 3 GB només amb els models de tres dimensions.

- Domini per proporcionar accés a l'aplicació a través de la xarxa

1.5 Beneficis i impacte

L'aplicació desenvolupada busca apropar eines que avui en dia poden existir però no estan orientades a la web. Mitjançant l'aplicació es poden trobar un conjunt d'eines que només són disponibles dins d'àmbits professionals i no estan obertes al públic ja que el cost que tenen és elevat i no és plausible la seva explotació en altres àmbits.

L'aplicació està orientada especialment a l'àmbit educatiu, ja que l'entorn on es presenta l'aplicació per utilitzar els seus serveis es a través d'aules virtuals, les quals permeten tenir uns gestors dels continguts que seran els interessats en oferir aquests serveis a altres usuaris dins del seu àmbit per tal d'ajudar a formar-se en àrees que estiguin relacionats amb els continguts que s'ofereixen dins dels serveis de les aplicacions, com poden ser medicina, biologia o botànica.

En aquestes àrees que podrien aprofitar-se del potencial educatiu és on pot afectar i arribar a revolucionar més en la forma en que s'imparteixen i es proporcionen els coneixements als usuaris o estudiants. Actualment és relativament senzill trobar informació relacionada i detallada sobre un tema, ja sigui a través d'investigacions publicades, llibres o a la xarxa, però és molt més complicat trobar material multimèdia per complementar aquesta informació com pot ser la imatge d'una cèl·lula, un teixit real, objectes amb els que interactuar, veure un cos humà. Tots aquests elements s'haurien de poder integrar dins de la formació de les persones

interessades i permetrien que els processos d'aprenentatge fossin menys complicats i més efectius, ja que es podria introduir i plasmar la realitat dins dels continguts educatius.

Un projecte d'aquest estil també amplia els horitzons professionals de les TIC. Fins ara els editors de continguts encarregats de proporcionar material estaven bastant limitats a les editorials que recopilen i ofereixen els coneixements que serveixen de base per l'educació en les diferents àrees. Aquest fet pot canviar, ja que amb la popularització d'aquests serveis es pot veure l'àmbit educatiu des d'una altre perspectiva, on també poden tenen entrada tota mena de material, que no ha d'estar limitat als mitjans de distribució convencionals.

Aquest impacte que es pretén aconseguir promocionant aplicacions d'aquest caire és possible gràcies als beneficis que pot aportar el seu ús, hi han molts aspectes positius que s'aporten i poques limitacions en quant a la seva utilització. Es poden destacar les següents aportacions:

- En el núvol. Si es té connexió a la xarxa és accessible des de qualsevol punt, per poder visualitzar un teixit vist des de un microscopi no és només possible fer-ho des del laboratori.

- Material multimèdia. Els continguts de l'aplicació són dinàmics, la qual cosa permet una interacció amb l'usuari possibilitant endinsar-se dins d'un material que d'altre forma es podria visualitzar en un entorn estàtic.

- Hardware. S'han comentat els pocs requisits que necessita complir l'usuari per poder utilitzar certs serveis. La configuració de l'equip que s'utilitzarà no hauria de ser un problema tenint en compte que avui en dia la configuració mitja que es ven en el mercat per un equip ofereix targetes gràfiques amb 256 MB dedicats i una quantitat de RAM que sobrepassa les exigències d'aquests serveis i a un preu molt assequible.

- Divulgació. Un dels motius de la creació d'aules és perquè es puguin accedir i introduir continguts que es complementen amb els que els serveis ofereixen per que siguin accessibles a diferents usuaris. La interacció que es busca

dins de cada servei permet afegir continguts possibilitant la inserció de informació en elements multimèdia com imatges i rotacions.

- Infraestructura. L'aplicació ofereix un conjunt d'entorns per adaptar-se a les necessitats d'estructuració i organització dels usuaris que en vulguin fer-ne ús. En el cas de voler només una aula per utilitzar els serveis o oferir uns continguts es disposa de l'entorn d'aula. Si es vol organitzar conformant un entremat d'aules i un complet entorn de gestió de cada un dels usuaris i aules també hi és disponible aquest entorn.

- Personalització. Els serveis que es poden utilitzar i les dades que s'introdueixen dins d'ells són totalment configurables i personalitzables, cada un dels usuaris apart dels continguts que s'ofereixin per defecte poden editar tota la informació. La disponibilitat i ús dels serveis es pot adaptar a les necessitats de cada usuari.

- Tecnologia a l'educació. Paral·lelament, amb l'ús d'eines d'aquest tipus es pot afegir un valor addicional en fases tempranes de l'educació per veure exemples del que es pot fer amb la tecnologia actual i les múltiples aplicacions que poden tenir i de com es pot innovar.

1.6 Memòria

En aquest document es presenta informació detallada sobre la realització d'un Projecte de Final de Carrera d'una enginyeria Tècnica. El projecte a desenvolupar tractava d'una aplicació web amb serveis amb aplicacions a la bioinformàtica social i de continguts fent ús del framework Symfony pel desenvolupament a la web a partir d'una projecte ja iniciat i parcialment desenvolupat.

El contingut de la memòria explica els motius i objectius que hi han al darrere del desenvolupament de l'aplicació com els aspectes principals de les etapes de desenvolupament del projecte, presentant informació del plantejament i l'estat de l'art en el que es trobava el projecte quan vaig començar a treballar en ell, on s'explicaran els canvis que es van haver de realitzar i el perquè. També hi ha un capítol on es fa una reflexió sobre la repercussió que tindria el desenvolupament

d'una aplicació d'aquest caire en una empresa i es toquen alguns aspectes del projecte management com són la planificació i l'anàlisi econòmic.

Es realitza un anàlisi i una especificació dels elements que es van afegir al projecte per la meva part, per tal de poder complir els objectius i requisits que es descriuen en aquest document. El format d'aquests apartats potser no és el més convencional, la decisió del perquè no s'ha realitzat un anàlisi i descripció exhaustiva abans de procedir al desenvolupament te a veure amb les metodologies que s'explicaran més endavant i el fet de partir d'una implementació tangible, aquests aspectes han influït en l'estructura i les etapes del projecte directament.

Deixant de banda el desenvolupament intern de l'aplicació, gran part del pes del projecte recau sobre l'aprenentatge i ús d'un framework concret amb el que ja havia una aplicació implementada. Aquest framework fa ús d'un conjunt de metodologies i tècniques que ofereix per desenvolupar aplicacions web que s'han utilitzat i han influït en gran manera en la planificació i el format d'aquesta memòria. La lectura que es recomana en el capítol que se li ha dedicat i en el que s'ha posat èmfasi, és una lectura informativa, però a l'hora receptiva.

En el capítol de Symfony he volgut plasmar els aspectes i components que han sigut importants pel desenvolupament, ja que a simple vista l'ús d'un framework es pot veure com una eina per facilitar la vida i a la que se li dona poc reconeixement i mèrit. Amb el contingut exposat però, espero que es pugui entendre l'abast que arriba a tenir Symfony en el desenvolupament de l'aplicació i la dedicació que ha requerit per aprofitar i exprimir al màxim les seves capacitats. Com es podrà veure el format que he utilitzat no es basa en l'exemplificació o indicació de les parts, he preferit condensar els coneixements adquirits amb les meves pròpies paraules per donar una visió tant genèrica, com específica vulgui obtenir el lector.

Per no fer la lectura més extensa es proposaran alguns annexos en els que es mostrarà algun exemple específic de la implementació realitzada d'algunes de les parts més utilitzades i que tenen un pes important en l'execució de l'aplicació.

Per finalitzar es dedica un apartat per reflexionar sobre els resultats obtinguts i l'experiència. També es fa un anàlisi des de la perspectiva d'un projecte acabat amb les possibles alternatives que es podrien seguir, altres pensaments cap el futur com recomanacions, futures revisions i noves versions que es podrien fer realitat o que que han quedat pendents i els motius que ho han ocasionat.

2 Estat de l'art de l'aplicació

El projecte que he dut a terme no ha estat començat des de zero, quan vaig entrar dins del projecte amb el que realitzaria el Projecte Final em vaig trobar amb una aplicació que contenia la base del que es presenta actualment. Es parla de base ja que tots els elements que hi havien encara segueixen sent-hi presents, tot i així les funcionalitats que hi havien han sigut millorades i s'han afegit mecanismes de control a part de les noves funcionalitats que s'han implementat.

La majoria de les tecnologies utilitzades ja s'estaven fent servir en el desenvolupament del projecte, el framework Symfony, els motors de bases de dades MySQL, SQLite, llenguatge PHP, Javascript, són tecnologies que es podien veure reflectides en la implementació que hi havia en aquell moment. Tot i així hi han algunes funcionalitats com plugins o extensions que s'han hagut d'afegir però per estendre algun comportament i aportar alguna nova funcionalitat, sense afectar a la resta.

Deixant de banda els aspectes implementats durant la realització del projecte, el que més divergeix respecte al seu estat inicial és la seva estructura i organització. Es parla d'organització millorada per diversos motius, el primer són els serveis. Els serveis no aprofitaven les seves característiques, la idea sota els serveis per oferir diferents funcionalitats abstraient-les sobre una capa comuna dins l'aplicació s'evitava utilitzant dins d'aquests enllaços a aplicacions externes que es comunicaven amb els seus propis components que gestionaven informació que estava relacionada amb el funcionament de l'aplicació.

Els problemes que es derivaven d'aquestes pràctiques afectaven a la seguretat i sistema de credencials que es volia implementar dins de l'aplicació, ja que les crides que es realitzaven per executar els serveis es desacoblaven de l'aplicació principal i per tant del framework, els mecanismes que controlen l'execució i es perdia el control de la gestió d'accés dels usuaris i els continguts introduïts.

Un altre aspecte que divergeix molt de l'estat actual del projecte és el model de dades, inicialment el projecte comptava amb diversos models de dades, les dades de l'aplicació principal es trobaven amb una estructura propera a la actual, i s'utilitzaven i gestionaven també dins del framework. Però el model de dades que s'utilitzava pels continguts dels serveis es trobava al igual que els mateixos serveis aïllats de l'aplicació. L'aplicació principal i el framework no tenien coneixement ni

contacte algun amb la connexió al servidor de base de dades dels serveis ni com es tractaven, utilitzaven o si es protegia l'accés.

Dins del model de dades en l'aplicació ens trobem amb una gestió i ús basat en el paradigma d'orientació a objectes com en l'actual, però en el cas dels serveis torna a divergir de l'aplicació. No hi havia cap definició ni cap mètode que es fes servir per manegar les dades que tractaven els serveis, i els accessos a base de dades es realitzaven sense fer cap gestió en l'accés, ja que la lògica que s'encarregava de realitzar aquestes tasques eren scripts PHP, que contenien la informació i configuració de les dades per accedir a la base de dades, i a l'hora d'haver de realitzar algun accés es realitzava directament. Per accedir a les dades s'utilitzava un esquema procedimental de programació per realitzar les operacions, en el qual el codi estava codificat i configurat per utilitzar les APIs específiques del servidor MySQL.

El comportament dels serveis com a elements externes havia de desaparèixer i passar a integrar-se dins de l'aplicació principal per poder oferir totes les funcionalitats dins del conjunt de l'aplicació, sense haver realitzat canvis com aquest no hagués estat possible tenir un sistema de seguretat autònom i únic dins de l'aplicació, s'hagués hagut de distribuir deixant possibilitats a una intrusió o evasió dels filtres que es realitzen.

Aquests aspectes també haguessin afectat com ho feien en aquell moment a la actualització i el manteniment de les aplicacions. Com es podrà comprovar tenint les dades i el codi dins del framework els canvis a realitzar en cas de modificació d'algun component es veu dràsticament reduït, o inclús innecessari, en l'estat que es trobava l'aplicació inicialment els canvis en el codi o el model de dades, implicava haver de refer els procediments que depenien.

Inicialment l'aplicació es basava en la creació d'aules exclusivament, aquestes aules podien estar relacionades amb una entitat, i a l'hora aquesta entitat amb una escola, les dades de les quals no s'utilitzaven. El rol que tenia la entitat s'hauria de veure com informació addicional que s'omple al crear un perfil, però sense cap funció. En la versió actual és una de les funcionalitats importants que ha canviat, afegint la entitat com a instància superior i de la que depenen la resta de serveis, les aules i els seus continguts. Per tant, les úniques accions de gestió a realitzar eren les realitzades amb la informació que definia la pròpia aula i els serveis que es

podien trobar dins, desproveint d'un conjunt de funcionalitats per la gestió de les aules i els seus usuaris.

En quant a la gestió d'usuaris que hi havia, existien els perfils d'usuaris, els credencials que determinaven el rol, però estaven limitats a lectura i escriptura. L'única diferenciació que es permetia era si els usuaris podien accedir només als serveis que hi havien a l'aula, o si podien accedir també a les preferències de les aules i afegir o treure serveis. Com ja s'ha comentat aquest mecanisme de gestió no s'estenia als serveis, per tant un cop accedit a un servei no es discriminaven privilegis ni accions a realitzar. Al no existir entorn de gestió la creació d'usuaris i l'assignació de permisos estava separada, per assignar permisos d'accés a una aula determinada s'havia d'executar manualment a la base de dades que contenia la informació.

Un altre aspecte que afecta als continguts de l'aplicació i que s'ha hagut de refer és la generació d'informació relacionada amb les funcionalitats i opcions dels serveis. Dins dels serveis es molt comú que apareguin diferents opcions i elements als que accedir i utilitzar, es prendrà com a exemple un servei de Zooms, quan s'accedeix a un es mostra s'ha de mostrar una llista de punts que seran accessibles per cada servei, aquesta informació es informació variable, ja que cada tipus de servei oferirà els seus propis elements, i per serveis del mateix tipus poden tenir configurats diferents opcions a les que es podran accedir.

Aquesta informació s'hauria de generar dinàmicament i sota demanda de cada instància de servei. En la versió que em vaig trobar la informació no es guardava dins les dades de l'aplicació, en comptes d'això les pàgines que s'encarregaven de mostrar totes les opcions de cada servei les tenien especificades en el seu contingut, el manteniment d'aquesta informació comportava haver de modificar la part de l'aplicació que s'encarregava de mostrar les dades.

La part de la visualització també ha sofert grans canvis. La interfície que hi havia tenia un disseny molt més senzill i no es trobaven elements per la interacció dinàmica amb l'usuari com els menús. Els elements que s'utilitzen dins la interfície com els menús, i taules per escollir elements no introduïen elements controlats amb javascript. L'aplicació en general no contemplava l'ús de Javascript a la interfície ni es feia ús de crides AJAX per la càrrega de de dades i el control dels formularis. A la versió actual aquest comportament de la interfície ha sigut re-definit, i es potencia l'ús de javascript per afegir efectes a la interfície que abans no estaven presents.

3 Planificació i estudi econòmic

3.1 Planificació

En aquest apartat es presentaran les planificacions que en un primer moment es van dur a terme per confeccionar un calendari aproximat del que s'esperava del projecte. Acte seguit es mostrarà la planificació resultat veient-se les diferències entre l'etapa inicial i la final.

En un principi el projecte es va plantejar de tal manera que s'esperava que acabés la realització a començaments del nou quadrimestre, tenint una durada esperada d'un quadrimestre sencer. Però a mida que el projecte s'ha anat desenvolupant les dates de duració de les tasques s'han anat allargant, denotant una estimació massa optimista, no havent marges d'error per tenir la possibilitat de compensar la demora entre les fases realitzades.

Tasques	Estimació durada(d)	Durada(d)	Diferència(d)
Estudi de Symfony	15	15	0
Anàlisis de l'aplicació	7	20	13
Adaptació de Mapexpl	10	20	10
Nous estils	5	10	5
Mòdul Entitat	15	20	5
Gestió de permisos	5	8	3
Mòdul Bones	10	15	5
Estructura de dades per línies i taules	5	15	10
Subtipus de Mapexpl	2	2	0
Accés a les aules	10	15	5
Logs	2	2	0
I18N	15	20	5
Total(d)	**101**	**162**	**61**
Total(h)	**808**	**1296**	**488**

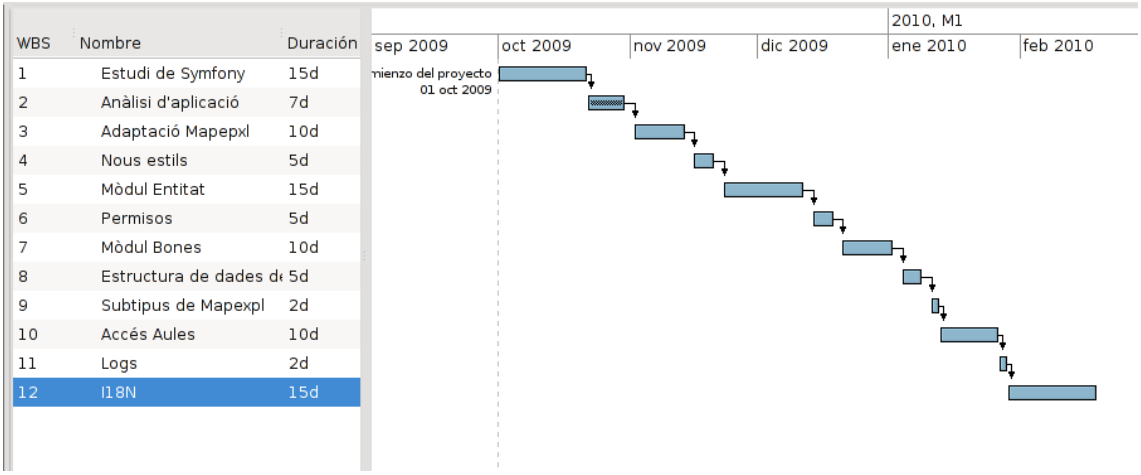
El número d'hores que s'ha acabat dedicant al projecte ha sigut clarament superior al que en un principi s'havia esperat. Aquest augment es pot atribuir a diverses causes. El primer de tots és la fase d'aprenentatge que hi ha hagut per el framework, que tot i haver-se realitzat un estudi i proves sobre el seu funcionament, no es podia arribar fins a un alt nivell de domini fins no s'apliqués en un projecte real de certa envergadura. Els tutorials realitzats que s'ofereixen són senzills i en certs aspectes molts límits, ja que s'utilitzen en casos molt didàctics i en molts casos massa idíl·lics que no cal esperar-se dins d'un projecte real.

Un altre aspecte que ha produït l'endarreriment del plantejament inicial ha sigut la fase d'anàlisi de l'estat de l'art de l'aplicació quan es va començar el projecte. La poca documentació va fer més difícil poder entreveure quines eren les aplicacions i el funcionament de les seves parts. Si a aquest fet li afegim la necessitat de realitzar proves en el projecte amb Symfony per poder veure les seves aplicacions i el funcionament que s'està utilitzant es troba amb una complicació encara major.

La resta d'endarreriments han vingut arrossegats pels aspectes negatius sobre la planificació ja comentats i d'altres més típics de problemes d'implementació. Al començar cada una de les tasques, es plantejaven uns objectius i condicions a anar aconseguint per tal d'implementar les funcionalitats esperades. Durant aquest procés s'anaven realitzant iteracions per tal de revisar i concretar incrementalment el comportament que s'acabaria obtenint en cada pas. No obstant, s'ha produït una situació en la qual la implementació que s'havia realitzar en algunes de les iteracions no resultava ser el que s'esperava, i s'optava per definir altre vies, havent de desfer en alguns casos la feina realitzada, amb el conseqüent allargament de l'etapa en la que es trobava el desenvolupament.

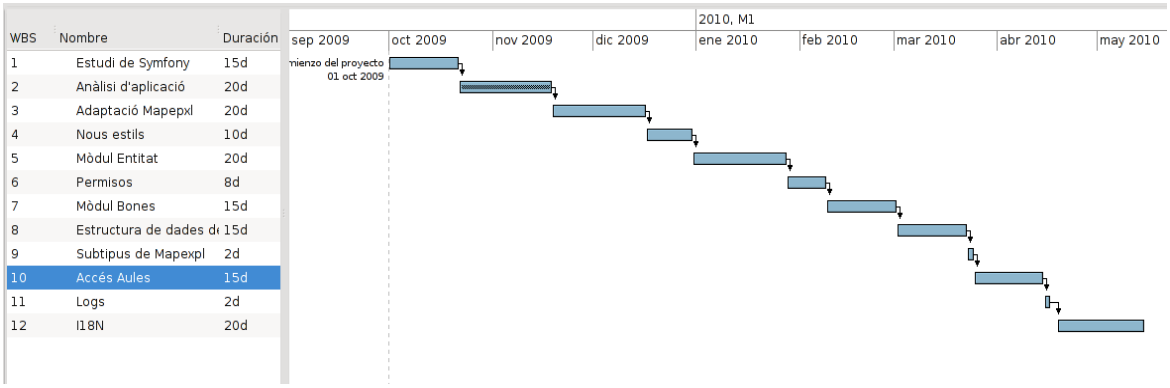
3.1.1 Diagrama de Gantt inicial

A partir de la duració estimada s'ha generat el següent diagrama:



3.1.2 Diagrama de Gantt final

A partir de la duració final s'obté el diagrama següent:



3.2 Estudi econòmic

En aquest apartat es realitza un anàlisis econòmic, fent uns càlculs i unes estimacions per determinar quin cost tindria realitzar el projecte en un àmbit fora del marc educatiu.

3.2.1 Cost hardware

El cost dels recursos hardware utilitzats es basen en l utilització d'un portàtil com el que he emprat que poso com a exemple, durant la realització del projecte, i un servidor que hospedarà l'aplicació que s'ha desenvolupat. A més del hardware, l'hospedament de l'aplicació al servidor i fer-la disponible a la xarxa també comportarà un cost extra per la contractació d'una connexió a la xarxa, i un domini per oferir l'accés.

Producte	Preu
Portàtil Acer Travelmate 5720 Intel Core 2 Duo T7300 2GB DDR RAM 250 GB HDD Intel GMA X3100	500,00 €
Servidor DELL PowerEdge T110 Intel® Xeon® X3430, 4C 2GB Memory, DDR3, 1333MHz 1 TB HDD	1.059,00 €
Domini	35,00 €
Connexió a la xarxa Jazztel PYMES Fins 30 MB/1'5 MB VDSL	54,00 €
Total	**1.648,00 €**

3.2.2 Cost software

Tal i com s'especificarà als requisits no funcionals, el projecte basa les eines de desenvolupament utilitzades en l'ús de software lliure, per tant el cost derivat d'aquest apartat és inexistent, posant com a exemple el software més destacable i que no està cobert en aquest document es destaca:

- Sistema operatiu Arch Linux
- Netbeans i Kdevelop com a entorns de programació
- Gimp per l'edició d'imatges

La resta de tecnologies cobertes en aquesta memòria també són lliures i no han generat cost algun.

3.2.3 Cost recursos humans

En el projecte realitzat, a més d'haver dedicat una mitja jornada de dedicació com a projecte final de carrera, també s'ha dedicat l'altre mitja jornada com a becari per desenvolupar l'aplicació.

Si es té en compte el preu/hora que es marca com a becari al voltant dels 6,25 €, i es contempla el nombre total d'hora que s'han realitzat durant el projecte entre la mitja jornada del projecte final i la beca, prenent les 1296 hores, s'obtindria un cost total pels recursos humans de 8100 €.

3.2.4 Cost total

Fent un càlcul amb els costos presentats anteriorment, el cost global del projecte seria aquest:

Producte	Cost
Hardware	1.648,00 €
Software	0,00 €
Recursos humans	8.100,00 €
Total	**9.748,00 €**

4 Anàlisi de requeriments

A continuació es detallaran els requeriments que s'han especificat pel desenvolupament del projecte i que representen les necessitats i les condicions que havia de complir l'aplicació. Aquests requeriments són els que han determinat els dissenys i implementacions que s'han realitzat. Aquests poden ser de dos tipus, els funcionals són els que determinen les accions que es permetran realitzar a l'usuari. I els requisits no funcionals tracten els aspectes que no es veuen reflectits en accions a realitzar per l'usuari però que igualment afecten al comportament de l'aplicació i a funcionalitats que s'han d'introduir.

Els requeriments que l'aplicació havia de complir s'han anat específicament a mida que es realitzava el projecte. Els requisits poden passar per diferents etapes des de la seva especificació i alguns poden acabar descartats o aparèixer de nous. Les fases per les que pot passar la determinació d'aquests requisits impliquen des de fases anteriors al disseny i implementació com les entrevistes en les que es proposen i apareixen les necessitats que es volen cobrir, fins a les fases de implementació i proves per veure si els requeriments compleixen les expectatives.

S'ha de destacar que les metodologies i per tant fases realitzades durant el projecte per definir aquests requeriments no han estat metodologies clàssiques de d'enginyeria de software, en comptes d'això i seguint la filosofia del framework i del projecte utilitzada s'ha preferit seguir metodologies per obtenir resultats encara no siguin del tot complets en períodes de temps curts i sense invertir molt temps en els processos previs de documentació i d'anàlisis, d'aquestes metodologies es parlarà més endavant quan s'analitzi la filosofia i mètodes de treball que pretén inculcar el desenvolupament d'aplicacions web amb Symfony.

Finalment els requeriments es complementaran amb la especificació dels models de casos d'us i les seves especificacions que reflectiran el curs dels esdeveniments de les funcions realitzades pels usuaris de l'aplicació.

4.1 Requeriments funcionals

Els requeriments funcionals són els que demanen de funcionalitats al sistema, aquest requisits deriven en la generació d'un subsistema o conjunt de codi que haurà de realitzar les accions relacionades amb aquestes funcionalitats. Aquests requeriments són els que determinen el comportament i funcionament de l'aplicació, ja que aporten les funcionalitats desitjades que es voldran utilitzar en ella. A continuació es detallen els requisits funcionals que s'han introduït a l'aplicació:

- Permetre la creació d'entitats que representin escoles o centres que vulguin tenir les seves pròpies aules connectades i gestionades.
- Assignació de rols professor / administrador a usuaris d'una escola.
- Editar els rols dels usuaris que pertanyen a una escola.
- Creació d'aules amb un password per l'accés d'altres usuaris.
- Cercador d'aules obert, sense haver d'accedir dins del sistema per poder buscar aules.
- Possibilitat de donar diferents tipus de permisos a altres usuaris per accedir a una aula pròpia.
- Edició d'aules pròpies (eliminar/desactivar/activar)
- Seleccionar serveis segons el seu tipus / sub-tipus.
- Selecció dels continguts disponibles per un servei.

- Editar les dades de la entitat.

- Canvi d'idioma

- Enregistrar informació del funcionament de l'aplicació.

- Introducció de textos amb imatges a través d'un editor.

4.2 Requeriments no funcionals

Els requisits no funcionals són aquells aspectes que igualment són tan importants per l'aplicació com els funcionals, perquè d'ells pot dependre l'èxit.

- Multi-plataforma. L'aplicació web i els seus serveis s'havien de poder executar en una gran varietat de navegadors, principalment els de més populars, sigués la plataforma que sigués.

- Interfície intuïtiva. L'usuari ha de trobar la interfície de l'aplicació còmode durant el seu ús, les opcions i funcionalitats han de ser clares, i explicatives, els elements visibles i ordenats, les interaccions no han de necessitar un alts nivells de coneixements en informàtica.

- Interfície atractiva. L'aplicació ha de veure's adaptada als nous corrents i tendències del disseny de la web, no s'ha de poder veure com una aplicació deixada, o desfasada.

- Accessibilitat. Dins l'aplicació s'ha de permetre accedir als diferents entorns i serveis de manera ràpida (quants menys clics millor), i amb enllaços fàcilment accessibles i clars.

- Manteniment. El manteniment de l'aplicació durant el seu funcionament hauria de ser mínim i no generar problemes, ni haver d'estar realitzant tasques al servidor per garantir el seu correcte funcionament.

- Extensibilitat. La possibilitat d'afegir nous serveis i continguts als serveis existents sense haver de modificar el codi de l'aplicació existent.

- Cost. Els requeriments del projecte s'haurien de garantir amb eines i processos de qualitat però sense haver de recórrer a solucions costoses, ja que actualment al mercat i han tecnologies molt potents, lliures i que no necessiten d'uns grans requisits pel seu ús, es tracta d'aprofitar les ofertes i coneixements per tal d'obtenir un bona relació entre cost-qualitat-funcionalitats.

- Seguretat. Oferir un sistema segur tant pels usuaris protegint-los d'atacs que puguin rebre per debilitats del sistema de seguretat de l'aplicació, i assegurant l'accés i autenticació a l'aplicació i als seus serveis, com poder garantir la integritat per les dades que es contenen dins l'aplicació.

- Disponibilitat. Els serveis han de poder estar sempre disponibles i el mateix pels seus continguts.

- Aspectes legals i llicencies. Ús de software amb llicències lliures i no de restrictives, que permetin l'accés i coneixement del seu codi i per tant saber com funcionen, i poder-los adaptar a les necessitats del projecte.

5 Plataforma tecnològica

Per dur a terme el projecte s'ha hagut d'utilitzar una sèrie de tecnologies, la majoria estan enfocades al desenvolupament específic de la web, però també d'altres amb un propòsit més general.

Durant la descripció d'aquestes faré èmfasi en alguns aspectes de les tecnologies que han sigut importants o si més no interessants pel desenvolupament de la aplicació, com també descriuré els possibles aspectes negatius que m'hagin repercutit en la implementació.

5.1 HTML

5.1.1 Historia

L'HTML (Hyper Text Markup Language), no està considerat pròpiament un llenguatge de programació tradicional, sinó que es defineix com un llenguatge de marcatge tal i com indica el seu nom, amb la funció d'estructurar documents que es volen visualitzar amb qualsevol navegador de pàgines web.

El llenguatge ha anat evolucionant des dels seus inicis, en la seva primera versió 1.0, ens trobem amb llenguatge basat en l'SGMLguid que a l'hora és un llenguatge basat en SGML que es feia servir al CERN (European Organization for Nuclear Research) per tal de compartir documents entre els científics, d'aquest llenguatge provenen tags com <title>, <p>, <ol>, l'única pròpia del nou llenguatge HTML va ser l'hiperenllaç <a>.

El SGML va ser creat per ser l'únic metallenguatge de marcat amb el qual es podria crear qualsevol altre element de marcatge d'un document. El problema del SGML és que és tant ampli i global que els simples mortals no el poden fer servir. Per poder utilitzar SGML efectiu requereix eines molt complexes i cares que no es troben a l'abast de la gran majoria dels usuaris que només necessiten i volen visualitzar documents HTML. És per aquests motius que l'HTML adquireix alguns, però no tots els estàndards del SGML, eliminant moltes de les funcionalitats que oferia el seu marcatge, com podien ser funcionalitats per minimitzar el marcatge i així oferir un grau de especificació més relaxat perquè la introducció del marcatge fos més senzill; fun-

cionalitats per enllaçar, per definir el tipus d'enllaç a un element que es farà servir, definir si serà simple, implícit per no processar el resultat del l'enllaç, o sí és explicit, en el qual es defineix que una altre document SGML és el resultat del l'enllaç.
Des de llavors l'HTML ha continuat millorant i ampliant les seves capacitats fins a convertir-se en l'estàndard per al desenvolupament web, el primer estàndard en publicar-se va ser amb la versió 2.0, però no va ser fins l'any 2000 que va ser reconegut com a estàndard internacional (ISO/IEC 15445:2000).

Actualment s'està treballant en els esborranys de la versió 5.0 que ja no es basarà en SGML, però tot i així no s'oblida de la compatibilitat amb versions antigues. Aquesta nova versió començada pel WHATWG (The Web Hypertext Application Technology Working Group) degut a la tardança del W3C (World Wide Web Consortium) i per la adopció com a estàndard a continuar del XHTML. Promet portar una revolució al llenguatge i apropar-se a la filosofia web 2.0 introduint elements com <video>,<audio> per poder eliminar la necessitat de plug-ins extres pels navegadors, també parts estructurals clàssiques com els blocs i , per elements com <nav>,<footer>, també introduirà nous elements als formularis amb lo que es denominarà webforms 2.0.

The World Wide Web Consortium(W3C) va ser format amb el propòsit de definir els estàndards per HTML, i més tard l'XHTML. Els membres són els responsables dels esborranys, les revisions, i modificacions dels estàndards basant-se en el feedback que es rep de la xarxa per acomplir les necessitats de la major part de la comunitat. Apart del HTML i el XHTML el W3C té la responsabilitat d'estandarditzar qualsevol tecnologia relacionada amb la Web, per tant, s'ocupen també dels estàndards de HTTP, CSS, XML.

Tot i que amb HTML i XHTML es poden mostrar multitud de continguts multimèdia i suporten i conviuen amb moltes altres tecnologies a la web amb les que es complementa és important entendre les limitacions del llenguatge. No són eines per processar textos, ni aplicacions d'escriptori, o llenguatges de programació com ja he esmentat. El seu objectiu es definir la estructura de documents i famílies de documents perquè puguin ser enviats ràpidament i fàcilment als usuaris a través de la xarxa i ser renderitzats. HTML i el seu descendent XHTML proveeix de diferents vies per definir la aparença dels documents, però estan centrats en la estructura. La aparença pot ser molt important segons la aplicació, per això s'han buscat solucions amb les que complementar aquest aspecte i els seus beneficis per l'usuari com els estàndards CSS. Deixant de banda la manera en la que es visualitza, el contingut es

lo primordial, la forma en que es visualitza pot variar per la varietat de navegadors i de formatjat de textos. Explicar limitacions que es poden trobar al intentar establir els estils per mètodes arcaics i que les seves funcions no són d'un editor de text.

5.1.2 Estàndards i extensions

Els desenvolupadors de navegadors i aplicacions relacionades amb HTML segueixen les especificacions i convencions per tal que els seus productes puguin treballar amb documents HTML i XHTML vàlids. I per una altre banda els autors creen documents basant-se en els estàndards per tal d'obtenir obres efectives i correctament escrites per a la visualització en els diversos navegadors. Però els estàndards no són sempre explícits, els desenvolupadors sempre tenen certa llibertat d'acció en la forma en que el seu software tractarà i mostrarà la informació. Però aquesta no són les úniques diferencies que es poden trobar entre navegadors, ja que en alguns casos s'arriben a afegir extensions no estàndards per millorar o afegir noves funcionalitats.

Un exemple del primer cas pot ser el tractament del valor que té un atribut com maxlenght d'un input, en el cas de Firefox i Opera ignoren la “o” i el valor queda com a “2”, mentre que Internet Explorer ignora el camp sencer. Per una altra banda estan les extensions afegides per altres, per oferir noves funcionalitats, i així també poder-se diferenciar dels navegadors estandaritzats. Això pot arribar a ser un horror pels autors, ja que es volen incloure les últimes i més noves funcionalitats per estar al dia en la web, però resulta que no totes estan suportades pels navegadors, o en altre cas, es pot trobar en que diferents navegadors poden tenir una manera diferent de fer la mateixa cosa, o mostrar el mateix.

Aquestes extensions poden arribar a estar tant utilitzades i reconegudes que fins i tot acaben sent introduïdes com a estàndards en noves i futures versions, moltes d'aquestes extensions han sigut introduïdes per empreses i corporacions com poden ser Sun Microsystems(RIP), Netscape(RIP too) o Microsoft. Existeixen etiquetes com marquee, que crea una marquesina per el contingut que engloba, o bgsound per tenir un so de fons mentre es visualitza el document, que només tenen suport amb navegador microsoft. O ni han d'altres que han sigut adoptades com poden ser la etiqueta <applet> (ara obsolets fent-se ús de <object>), o la extensió de frames, script, etc... o d'altres etiquetes i extensions que oferia el navegador Netscape que són alguns exemples de extensions pròpies que amb el temps la fama, i el bon fun-

cionament van ser afegides en les noves revisions d'HTML com: frame, javascript, afegint nous atributs a elements HTML ja existents així estenent-los i ampliant les seves opcions de visualització.

Així doncs et pots trobar amb la disjuntiva de incorporar noves funcionalitats o aplicacions en els teus documents, o bé perdre una part dels possibles usuaris que per no utilitzar un navegador amb suport per les extensions no podrà visualitzar el document correctament o fer-lo anar. Si et trobes en el cas que es mandatari utilitzar alguna d'aquestes extensions no estàndards, una solució pot passar per realitzar diferents versions o alternatives que compleixin amb els estàndards i que permetin la seva visualització encara que sigui amb un comportament diferent. Sol ser una bona pràctica.

5.1.3 Etiquetes

En l'HTML, perquè els navegadors puguin renderitzar els documents s'interpreten unes etiquetes que són les que indican l'estructuració de cada un dels elements del document, per tant podem veure un document html com un arbre d'etiquetes, ja que de cada etiqueta poden descendir d'altres, i totes tenen una etiqueta pare de la que provenen. Aquestes etiquetes normalment comencen i acaben, però segons l'element que es tracti i la versió aquest comportament és més o menys restrictiu. La gran diferencia entre l'HTML i el XHTML radica en la necessitat obligatòria d'haver de tancar totes les etiquetes, aquest comportament ve donat per l'XML i que s'ha fusionat per donar lloc al XHTML. Una altra diferència es la sensibilitat a les majúscules, en el cas de html no discrimina si ho són o no, però en xhtml les etiquetes han de ser escrites en minúscules.

Les etiquetes HTML estan encastades dins del document, i son les que guien el contingut del document, i marquen el que es mostrarà. Els navegadors utilitzen la informació dins d'aquestes etiquetes per decidir com mostrar, o bé com tractar el subconjunt d'elements del document. La primera paraula del tag és el seu nom formal, normalment sol ser bastant descriptiu amb la funció que realitza dins dels documents. Les paraules addicionals que es poden trobar dins d'una etiqueta són atributs especials, alguns poden tenir assignats uns valors tot seguit, que acabaran de definir o modificar l'acció del tag.

La majoria de les tags afecten una part concreta del document. La regió afectada comença on per primer cop apareix la etiqueta i els seus atributs, i continua fins que arriba a una etiqueta d'acabament. Aquesta etiqueta es el mateix nom amb la que s'ha obert precedida per "/", a més aquestes etiquetes mai porten atributs. En HTML la majoria d'etiquetes tenen una de tancament, però hi ha algunes que no, com la etiqueta <link> (enllaç a un fitxer CSS extern). També s'ha de tenir en compte que el fet de deixar d'afegir segons quines etiquetes no ha de comportar necessàriament en una incorrecta visualització, els navegadors pot introduir etiquetes obvies al voltant d'un text al interpretar el document html, això passa amb la etiqueta <p> (tag que defineix un paràgraf).

L'estructura de l'esquelet d'un document html el defineix el tag <html></html>. L'estàndard d'html i xhtml per tal que el compleixin requereixen d'aquesta etiqueta, però en aquest cas la majoria de navegadors poden detectar i mostrar informació codificada en html en documents que falta la etiqueta. Dins dels documents html trobem dos estructures principals, <head><body>. Dins de la primera es col·loca informació sobre el propi document, i dins la segona hi ha el contingut que es voldrà mostrar en la finestra del navegador, aquesta és la que porta casi tota la càrrega del document en la gran majoria dels casos. Un altre element estructural que l'estàndard obliga a introduir, però que un altre cop els navegadors no obliguen als seus autors a ser introduïts tot i així és <title>. El contingut d'aquesta es mostra generalment en la part superior de la finestra que fa el marc del navegador.

Exceptuant les etiquetes comentades i que estan especificades com a obligatòries pels estàndards, no hi han altres elements estructurals que siguin s'hagin d'inserir dins d'un document. La resta d'elements per introduir són de lliure elecció i opcionals. Tot i la gran quantitat d'elements estructurals per introduir dins, el contingut que s'afegirà és pot classificar en tres grans categories: tags (ja comentades), text i comentaris. Al veure només aquestes tres grans categories es pot arribar a pensar erròniament que s'està oblidant d'un altre gran grup, el que contingui elements multimèdia, però el cas és que una imatge, sons, vídeos, o altres elements multimèdia no són més que referències inserides als documents cap a uns fitxers externs, i el navegador és qui s'encarrega d'utilitzar aquestes referències per carregar i integrar aquests continguts en el propi document html.

5.1.3.1 Principals etiquetes

La majoria de funcionalitats venen donades per un conjunt d'etiquetes de les quals després n'hi dependran d'altres i acaben de definir la seva estructura i propietats. Amb aquest subconjunt d'etiquetes es poden definir la majoria d'elements que s'utilitzen i que es comú trobar dins dels documents (X)HTML.

Les etiquetes de marcatge relacionades amb text comprenen un gran conjunt dins d'aquest llenguatge, això ve donat per l'origen del llenguatge, que es va fer servir per poder enriquir, estructurar i organitzar millor el text i per la seva orientació acadèmica, per poder ser escanejats i distribuïts a través d'Internet i ser mostrats. Per aquest motiu el disseny s'ha considerat secundari respecte a la estructura, ja que encara que nosaltres podem analitzar textos i estructurar-los segons com es veu, les màquines acostumen a llegir per elements marcats.

A l'hora de modificar l'aparença del text tenim diferents opcions, els estils de text basat en els seus continguts, són etiquetes que poden modificar l'aparença del text, però que a més volen donar a aquella part del text un ús o sentit especial. Aquestes etiquetes poden ser <cite>, per denominar una cita, , per fer èmfasi,o <var>, per afegir codi de programació, deixant de banda el possible canvi a la visualització, pot tenir aplicacions a la cerca de continguts a través d'aquestes etiquetes, per això és important fer-les servir quan sigui necessari i pel seu context, no per l'aparença que pot donar.

Una altre opció és fer servir els estils físics, aquesta és una forma d'atorgar una aparença específica a text, sense haver de donar-li un significat especial. Comunament es pot confondre aquesta pràctica amb la descrita anteriorment, aquesta opció s'hauria d'utilitzar quan hagi de prevaldre el fons per sobre les funcions.

HTML també treballa amb uns elements que són els denominats caràcters especials, ens podem trobar amb problemes per introduir certs caràcters, com poden ser "<" que poden fer confondre el navegador al interpretar el document, així doncs es dona una via alternativa per poder introduir qualsevol caràcter dins del conjunt de caràcters Unicode amb la codificació de la seva entitat de caràcter. També ens podem trobar en la situació de voler introduir algun altre caràcter que no és possible introduir via teclat, com pot ser un símbol de copyright, la lletra pi, o bé

una fletxa, això i lo descrit abans es pot aconseguir en dos formes, en les dues es comença amb “&” i s’acaba amb “;”, entremig es pot fer servir el número de la posició del caràcter dins la taula Unicode precedit amb “#”, o bé directament el nom que es fa servir per la entitat.

La forma en la que es visualitza un document en pantalla és deixant fluir el text que conté, els salts de línia i espais que es van trobant no es tenen en compte i no es mostren, el navegador omplirà tant com pugui cada línia de text dins la pantalla fins al final i seguint des de el començament en una nova línia. Per tal de poder controlar i organitzar el text d’una forma més llegible i estructurada html ofereix certes formes, una d’elles consisteix en agrupar el contingut dins d’uns elements estructurals com poden ser una divisió , paràgraf <p>, una altre opció seria estructurar les parts en seccions amb <h1> - <h6>, aquests elements separen i trenquen el fluir del text i el separa en noves línies. En els primer es poden fer servir amb atributs per modificar la visualització del contingut que delimiten a part de estructurar-lo. Uns altres elements comuns seria el salt de línia
, les regles horitzontals <hr>. També hi ha la opció de mostrar el text tal i com està escrit, amb salts de línia, sangria, espais... A més obliga a que encara que la finestra del navegador sigui redimensionada el contingut d’aquest bloc no es vegi afectat com passa en els altres casos.

Un element també molt important i vital pels documents HTML són el links, ja hem comentat algun dels seus usos, com poden ser les imatges i qualsevol altre mitjà multimèdia, hem dit que no són més que links que el propi navegador llegeix, obté i introdueix dins del document, i aquest element és tant important perquè així podem enllaçar qualsevol mena de localització, no obstant, que podem enllaçar no significa que tots els mitjans que afegim es puguin integrar dins dels nostres documents HTML. En el cas de que no es puguin integrar directament en el contingut del document no vol dir que el navegador no sàpiga manegar-lo, ja que es pot donar que un plugin o un programa extern sigui l’encarregat d’obtenir l’enllaç i executar-lo.

També té una altra funció molt important i interessant pels autors de documents, i un altre cop és la de estructurar. Segons el contingut que tingui un document pot arribar a ser molt llarg i pesat, així doncs el fet de poder enllaçar a altres recursos com poden ser altres documents HTML ens ofereix la possibilitat de separar continguts i estructurar-los segons ens convingui per fer la visualització menys pesada i més agradable, organitzada i accessible pels lectors, i no només a això,

fent ús d'atributs d'elements com "id" i "name" es pot fer que s'enllaci a un element del mateix document. La forma d'introduir aquests enllaços als nostres documents és mitjançant l'etiqueta <a>, (aquesta etiqueta s'anomena anchor, àncora en anglès, va rebre el nom perquè té la funció d'"ancorar" els enllaços i elements dins del document), la part de text i imatges que estigui continguda dins el bloc de la etiqueta serà la que activarà l'enllaç si l'usuari interactua amb ella i desencadenarà l'acció d'anar al recurs donat.

Un altre element molt comú en els documents HTML són les imatges, com ja s'ha comentat es tracta d'enllaços que el navegador llegeix i obté la imatge, fent ús dels descodificadors amb els que ja ve proveït, processa i la introdueix. Les imatges no només es fan servir com a il·lustracions sinó també com a icones, punts clau de llistes i altres petits elements decoratius per formatjar el text. Una altre aplicació és com a image map, fent servir una imatge com a mapa en la que quan el lector faci clic en una part de la imatge es farà una crida a la url especificada en la que s'enviaran les coordenades del clic de l'usuari al servidor i calcularà la posició per executar la funcionalitat que tingui designada el mapa. Actualment com que autors no tenen accés al servidor es fa servir mitjançant javascript aquesta tasca del costat del client. També es pot combinar amb les etiquetes <map> i <area> per no haver de dependre del servidor web, introduint tota la informació en HTML.

Els que s'utilitza sovint per estructurar el contingut són les llistes, n'hi han de tres tipus, ordenades <ol> , desordenades <ul> i de definicions <dt></dt>. La diferencia entre el primer tipus i el segon és ben clara, en el cas de les llistes de definicions per cada element hi ha el nom o títol, i en la següent línia la definició corresponent. El punt fort a remarcar de les llistes és la capacitat per inserir una gran varietat d'elements de l'HTML en el seu interior, fins i tot es suporta llistes dins d'una llista.

L'element perquè els usuaris puguin interaccionar dins del document són els formularis <form></form>. Dins d'un formulari es poden introduir una gran quantitat d'elements com entrades per text, etiquetes, valors per defecte, comboboxes, checkboxes, botons, el formulari interactuarà segons com estigui definit el seu comportament, podent enviar la informació al propi servidor perquè processi aquesta, o bé es pot processar des de el costat del client amb codi javascript entre d'altres. Una de les funcionalitats important de la que no fa gala és la verificació automàtica dels camps del formulari, això però es pot salvar fent ús

d’algun script si no es vol esperar a enviar la informació al servidor i evitar tornar a carregar per informar de l’error.

Un altre subconjunt d’etiquetes per formatjar la informació són les taules, es poden definir fila per fila <tr></tr>, amb capçaleres <th></th>, o amb la informació de la taula <td></td> s’ha de mencionar que hi ha un número d’atributs que permeten adaptar la aparença, format i dimensions d’aquestes a les nostres necessitats. És habitual utilitzar les taules no només per mostrar dades o com a típiques taules, sinó també per estructurar el contingut gràcies a la seva varietat en el format en que es poden visualitzar.

Un altre element important a l’hora d’estructurar el document HTML són els frames<frame>. Els frames provoquen la divisió de la finestra en finestres independents amb els seus propis continguts, que no han de ser necessàriament altres documents HTML. Així, si un frame conté enllaços es pot controlar el comportament, fent que reemplaci el contingut del mateix frame, el d’un altre, o tota la pàgina del navegador. Els frames comuns s’han de definir dins d’etiquetes <frameset> que en aquest cas a més substitueixen l’etiqueta <body> i són les encarregades de definir com dividir la finestra del navegador.

Hi ha un altre tipus de frame però, molt pràctic i usable, es tracta del <iframe> inline frame, a diferencia dels altres no necessita estar contingut dins d’un <frameset>, i es comporten com una imatge, definint una àrea rectangular que mostra un altre document podent tenir els elements propis d’una finestra, scroll i vores com els altres frames; poden ser introduïts com qualsevol imatge dins del document, aquest és el cas que farem servir nosaltres per visualitzar alguna de les nostres aplicacions,

5.1.3.2 Continguts executables

Una de les capacitats més útils que ens podem trobar per estendre els nous documents amb noves capacitats/funcionalitats és la d'afegir aplicacions que és complementin amb el nostre document. Les formes més comunes de fer-ho són amb scripts i amb applets. Els applets són petites aplicacions basades en java, independents de la plataforma. Durant l'execució aquests programes poden executar contingut dinàmicament en el client, distribuint així una part o la totalitat de la càrrega computacional del servidor a l'ordinador del client i al seu navegador.

També pot interactuar amb l'usuari, validar informació, crear finestres, executar aplicacions independents a la pàgina. És una forma de poder facilitar aplicacions i altres funcionalitats al usuari sense que hagi d'actualitzar el seu navegador o software, ja que no es necessita de cap extensió ni software específic per executar-lo, mentre que el navegador utilitzat suporti els applets, i actualment tots els navegadors populars tenen suport. Els applets es poden introduir de diverses maneres dins d'un document HTML, la primera es fent ús de la etiqueta <applet>.

Aquesta etiqueta actualment està obsoleta, i ha sigut substituïda per la etiqueta recomanada <object>, la qual es fa servir per incloure dins del document applets i altres elements com podrien ser imatges, i també d'altre contingut no-HTML/XHTML, com pot ser contingut multimèdia. En el cas d'utilitzar el paràmetre data dins de <object>, el navegador descodificarà el tipus de contingut i si el pot manegar el renderitzarà directament, però en el cas que no invocarà el plugin associat a la aplicació perquè s'encarregui d'executar-ho.

L'etiqueta <object> va ser originalment implementada per Microsoft per suportar els controls ActiveX, més tard però afegiria suport per Java, de manera similar, Netscape inicialment suportava la alternativa <embed> i <applet> per la inclusió d'objectes, i més tard afegirien també suport per <object>. Tot i així hi ha una resistència a suportar la etiqueta, per una banda està el perdre una part de navegadors que no arriben a oferir suport per antiguitat, i per una altra banda està relacionat amb queixes sobre una especificació vaga de com els navegadors haurien d'implementar el suport a applets de Java i el suport actual dels navegadors és inconsistent, per aquest motiu alguns com Sun recomanen que es segueixi utilitzant applet per incloure'ls.

En conclusió, els punts forts d'un applet davant una aplicació els trobem en que aquest pot obtenir la pàgina del navegador, accedir i renderitzar html dins per nosaltres, també pot ocasionar l'execució de codi sense connexió, com hem comentat està l'avantatge de que l'usuari no ha d'instal·lar res, sinó que automàticament s'obté l'última versió del software. Però també ni han punts negatius: per poder executar funcionalitats importants dins del codi s'ha d'utilitzar applets signats que l'usuari ha d'acceptar.

L'applet consumeix els seus propis recursos, però ha de coexistir amb el navegador, això pot semblar que avui en dia no és important, però en els inicis quan les quantitats de memòria ram no eren tan grandioses podia arribar a ser un problema

la falta de recursos, per això, tenint em compte el consum d'ample de banda que pot generar, s'han de dissenyar i utilitzar tenint molt en compte com pot afectar als usuaris, i administrar els recursos per satisfer a la majoria de la audiència(balanç entre servidor i client). També ens podem trobar bugs en el suport d'alguns navegadors, i intentar salvar-los per tota una varietat de navegadors pot ser complicat, no obstant, aquest és un problema que ens trobem en la gran majoria de tecnologies.

La velocitat a la que s'executarà el codi serà lleugerament inferior a la d'una altra aplicació escrita en c++ per exemple, ja que no es poden utilitzar compiladors de codi natiu, sinó que estàs limitat a la màquina virtual que suportà el navegador. Respecte al control i l'accés que té un applet, els navegadors assignen una part de l'espai del document on l'applet es podrà executar.

Com a aplicació la seguretat que oferirà al client dependrà de la versió que es s'executi, si és el cas d'un applet sense firmar, el seu accés al sistema de fitxers i permisos de la maquina del client està limitada, i també té l'accés web limitat al host de l'applet això és important, ja que d'aquesta forma s'evita que es puguin incloure applets a documents html que quan executats pel client provoquin un atac DOS a una màquina (un atac a la mateixa web que conté l'applet es torna un sense sentit, ja que normalment els que administren la web, també administren el seu contingut). Un applet signat però, com és el nostre cas, comporta la verificació de la signatura mitjançant certificats, i que l'usuari en el que s'executi l'applet accepti la seva execució, amb això l'applet pot obtenir més permisos i així evitar les limitacions establertes als applets no signats.

5.1.4 Tipus de contingut

La declaració del DOCTYPE hauria de ser la primera sentència en aparèixer en el document html abans de la etiqueta <html>. No es considera una etiqueta pròpiament d'html, és una instrucció que diu al navegador en quina versió del llenguatge està escrit el document. El DTD defineix les etiquetes i la sintaxis que s'utilitzen per crear el document html.

En el cas d'assignar un DTD que no correspongui amb el de la versió que s'utilitza pot causar que el navegador no entengui correctament el document al visualitzar-lo. Això es dona perquè els navegadors poden renderitzar els documents en dos mo-

des, el mode estàndard en el que treballen segons els estàndards i el document es renderitzarà com s'espera. Però en l'altre mode, quirk, que s'activa quan es troba amb un document que no compleix els estàndards especificats, el navegador el mostrarà com si ho fes un navegador antic que no respectava en tots els casos els estàndards. El cas és que s'ha detectat que hi han diferencies entre els navegadors en el mode quirks, i moltes propietats de CSS es veuen afectades com: padding, margin de les imatges es veuen afectats, el centrat d'un bloc CSS amb margin:0 auto no funciona, la herència de fonts des de body o taules, la declaració dels tamanys de fonts en una cel·la d'una taula es relatiu respecte la font del navegador i no al seu element pare...són alguns elements entre d'altres que poden afectar greuement a la visualització esperada.

Entre XHTML Strict, Transitional y Frameset, en els tres casos el document xhtml ha d'estar correctament escrit i ben format. En el primer no es suporten les etiquetes antigues i no es permet l'ús de framesets, és el més estricte dels tres. El segon és com el primer però treient la restricció de les etiquetes antigues. I en l'últim es treu la restricció d'ús dels framesets.

5.1.5 Atributs

Uns altres elements importants de l'HTML són els atributs, s'utilitzen per estendre les propietats d'una etiqueta, quan el navegador interpreta les etiquetes, també cerca parells d'atributs/valors dins de cada etiqueta si en conté, i el que acaba renderitzant és l'etiqueta amb unes característiques donades per aquests valors.
Algunes etiquetes tenen uns atributs per defecte que poden ser modificats pels que són especificats directament, com els borders de les taules que son nuls per defecte.

Els atributs poden ser de molts tipus, poden afectar a la aparença de l'element en el que estan definits com el color, el tipus de font, també poden afectar el seu comportament, com per exemple els atributs d'esdeveniments: onmouseover, onmouseout, que fan que si es compleix l'esdeveniment pel que estan definits executaran la tasca declarada, que pot ser codi en un llenguatge client-side com el Javascript. També poden aportar informació per definir un element, com pot ser el camp "id" o "class", els quals respectivament declaren com a identificador de l'element el valor donat, i afegeixen una classe a l'element, a més aquests elements també poden servir per discriminar sobre els estils que els hi afectaran.

5.2 XML

Sabent que SGML era massa pesat per descriure HTML d'una manera útil, i que hi havia una necessitat de crear un llenguatge de marcatge similar al HTML el W3C va crear el XML. XML és un metallenguatge com el SGML, però que utilitza unes funcionalitats del SGML per definir llenguatges de marcatge d'una manera similar al HTML. Elimina doncs, molts elements SGML que no són aplicables a llenguatges com el HTML, i simplifica d'altres per fer-les més fàcil d'entendre i d'utilitzar. La seva importància i presencia a la web va augmentant, a l'hora d'intercanviar informació entre persones, màquines i sistemes.

A diferència de HTML/XHTML, XML és un metallenguatge que va crear el W3C i es utilitzat pels desenvolupadors per definir altres llenguatges de marcatge com el XHTML. Es diu que és un metallenguatge perquè, XML no especifica ni cap semàntica, ni cap conjunt d'etiquetes o marcadors. Proveeix d'una facilitat per definir etiquetes i la relació estructural que hi ha entre elles. Donat que no existeix cap conjunt d'etiquetes pre-definides, no hi haurà cap tipus de semàntica establerta i per tant, serà establida per la aplicació encarrega de processar la informació. Per això els desenvolupadors de navegadors confien en les regles de XML per crear processos automatitzats que llegeixen la definició del llenguatge XHTML i implementen els processos que mostren o processen els continguts d'un document XHTML.

El motiu pel que es va desenvolupar aquest metallenguatge és perquè d'aquesta manera hi ha un estàndard per poder definir llenguatges de marcatge per qualsevol tipus de necessita en comptes d'haver de dependre del HTML i les extensions i/o noves funcionalitats que vagi apareixent. Així utilitzant XML s'obté una forma perquè matemàtics, físics, músics puguin expressar diferents tipus d'informació, puguin definir el seu propi llenguatge de marcatge com els hi convingui, i els desenvolupadors s'hauran d'ocupar de crear parsers tals com els estan en els navegadors per l'HTML, per tal que llegeixin les noves definicions dels llenguatges i permetin al servidor processar els documents en aquests llenguatges.

Donada aquesta capacitat del XML de definir una gran varietat de llenguatges de marcatge, mostrar els documents és més complicat. Un dels objectius amb la creació de l'XML, és desenvolupar uns navegadors, tals com els que fem servir avui en dia per poder visualitzar els continguts de documents HTML, per poder visualitzar

el contingut d'un document XML, encara s'estan definint. Tot i haver una especificació existent per les fulles d'estil per l'XML, anomenada XSL, no és suficient, ja que deguda la gran diversitat de llenguatge i per tant continguts, ens podem trobar en moltes situacions que no es puguin ni manegar amb fulles d'estil.

XML es composa d'un conjunt de regles relacionades amb els elements i entitats que pot contindre el llenguatge. Un document XML es pot considerar ben format i vàlid. Per ser un document ben format, ha de complir obligatòriament aquestes normes:

- Ha de contenir com a mínim un element el document.
- Només pot contenir una etiqueta per marcar el inici i el final del document, coneguda com a etiqueta arrel.
- Les etiquetes han d'estar niades correctament. Això es tradueix a que sempre que hi hagi una etiqueta d'inici i haurà una de tancament. I entre les etiquetes niades no es poden encavalcar, s'han de tancar en el mateix ordre d'obertura. En el cas especial d'etiquetes HTML que no tenen etiqueta de tancament s'afegeix dins d'aquestes al final.
- Les etiquetes son sensibles a las majúscules, dues etiquetes una en majúscules i l'altre en minúscules es processaran com a etiquetes diferents.
- Els valors dels atributs han d'estar sempre entre cometes, a diferència d'HTML que poden anar sense.
- Atributs no es repeteixen dins d'una etiqueta
- Sense caràcters il·legals
- Les referències a entitats estan declarades.
- S'utilitza una declaració d'XML.

Tots els documents XML són documents ben formats, però opcionalment també poden ser vàlids. Perquè siguin vàlids han de complir algun DTD. Al ser un metallenguatge, per definir les regles dels llenguatges que es definiran a partir de l'XML, s'utilitza el que s'anomena DTD, en aquest cas XML DTD (XML Document Type Definition).

Els DTD són un conjunt de regles que defineixen que etiquetes apareixien en un document XML, els atributs que poden contenir les etiquetes, i la relació que hi ha entre les etiquetes. Quan un document XML és processat es compara amb el DTD per veure si està estructurat correctament i si els seus elements són utilitzats de manera correcta. Aquest és el procés de validació i el duu a terme un

analitzador(parser). Els DTD poden ser interns, els quals són inserits dins de la mateixa declaració de tipus de document. També poden ser externs, ja que poden arribar a ser bastant complexos, es guarden com a fitxers de text amb la extensió ".dtd" en fitxers separats i són referenciats des de la declaració de tipus de document. És comú que segons el tipus de document que estiguem creant ja existeixi un DTD publicat obert per la seva utilització, així es facilita la tasca de no haver de re-escriure una declaració per un llenguatge ja creat i especificat. Els DTD poden contenir 4 tipus de declaracions XML al seu interior: d'elements, llista d'atributs, d'entitats, i de notacions. Cadascun del tipus de declaració pot contenir les regles que definiran aquest subconjunt d'elements i que s'aplicaran al document XML amb el que es vulgui validar.

L'èxit de l'XML i la seva acceptació deixant de banda la seva estandardització, ve donada per la seva gran capacitat a l'hora d'intercanviar informació. Amb l'aparició d'XML apareix s'evita el haver de crear i/o utilitzar un format per transferir informació creat pels propis usos i així dificultar la transferència entre diferents plataformes, amb XML ens trobem amb una plataforma neutre i genèrica en la que es pot emmagatzemar qualsevol tipus d'informació.

Està a la disposició de qualsevol i es pot integrar amb facilitat dins de les aplicacions, l'aparició de noves eines per crear i validar documents faciliten encara més la tasca de crear correctament els documents per poder guardar i intercanviar la informació que vulguem. Si a més tenim en compte, que s'evita la problemàtica de la visualització correcta de la informació és un punt afegit, normalment es voldrà transferir la informació i la mateixa aplicació que utilitzi XML per la transferència de les dades serà la encarregada de mostrar la informació com li sigui convenient, així doncs només hi ha un cost mínim per tal d'incloure capacitats de transferència de dades basades en XML sense cap afegit extra.

Per tal d'intercanviar informació ens estalvia el fet d'haver de definir un protocol per aquestes tasques, o bé adaptar-se a un protocol encarregat de la transferència de dades que no estigui tan orientat a informació estructurada i fàcil d'extreure. D'aquesta manera només és necessari definir un DTD i integrar el analitzador("parser") dins de la nostra aplicació. D'aquesta manera es pot utilitzar XML amb la finalitat de transferir, es pot estalviar l'emmagatzematge de les dades fent servir el DTD com a protocol per la transferència per tal de complir les necessitats i ser enviat a través de la xarxa en una connexió entre diversos sistemes.

5.2.1 Aplicacions XML

Un dels usos més estesos per l'XML a la xarxa són els serveis web (Web Services) en els que s'utilitza XML per definir protocols per intercanviar informació sobre les funcionalitats dels serveis(SOAP per exemple), com es comunicaran amb els serveis (WDSL), i com es poden trobar(UDDI).

5.2.1.1 XLIFF

Una de les altres grans aplicacions i més esteses de l'XML i que es fa servir durant el projecte és el format XLIFF(XML Localization Interchange File Format), aquest format que va ser creat basat en XML i és un estàndard de localització proposat per OASIS(Organization for the Advancement of Structured Information Standards, la organització sense ànim de lucre independent més gran del món dedicada a la estandardització d'especificacions eBusiness, treballen molt amb l'XML i eBusiness) al 2002 i que actualment està a la versió 1.2 (2008), està dissenyada per ser una especificació per l'intercanvi de dades localitzades i la seva informació relacionada sent neutre per la plataforma i eina que s'utilitzi. També facilita l'emmagatzematge de informació útil en un procés de localització, així també es podran utilitzar múltiples eines que treballin amb la mateixa informació i puguin interactuar i/o actualitzar-la.

En els fitxers XLIFF trobarem dades que necessiten ser modificades per tal de localitzar els continguts originals amb els que els documents han estat creats, poden contenir per tant: text traduïble, informació de les fonts, coordenades de controls de interfícies, gràfics, etc... I junt a aquesta informació també es poden facilitar meta dades com: identificadors, longitud màxima del text, tipus del recurs, notes pels traductors,... Per tal d'obtenir com a resultat uns continguts localitzats, a partir dels continguts originals es poden generar uns continguts localitzats dins dels fitxers XLIFF i aquests fitxers junt amb l'esquelet en el que es contenia els originals es poden fusionar mitjançant una eina per tal de generar el resultat esperat.

Dins d'un document XLIFF ens trobem dos parts diferenciades, una és la capçalera on es poden definir informació del projecte com informació de contacte, fases del projecte, referencies a material i informació de l'esquelet del fitxer. L'altre part, el cos, conté elements <trans-unit> que són els elements principals dels fitxers XLIFF. Aquests elements són els que contenen text localitzable i les seves respectives

traduccions. A l'hora, un element trans-unit conté principalment elements source, target, alt-trans, entre d'altres.

Un element pseudo-traduit conté un identificador utilitzat per determinar on està ubicat dins del document original. L'element target representa la traducció acceptada del source després que s'hagi produït la validació de l'idioma. També es pot trobar l'element alt-trans, aquests elements s'utilitzen però per afegir traduccions alternatives que puguin ser útils per aplicacions o per traductors, dins podem trobar definits altres idiomes diferents de l'origen i el destí, ja que dins d'un document XLIFF es declara l'idioma origen i el final en l'etiqueta <file>, i ens podem trobar varies dins d'un document XLIFF.

També poden haver atributs per indicar la qualitat de la traducció alternativa, o la eina amb la que s'ha afegit aquesta traducció. S'ha de tenir en compte que el procés de traducció tot i que existeixin estàndards, frameworks i eines per poder facilitar la feina, sempre ha de ser revisat i supervisat per una persona, els desenvolupadors o traductors, ja que les persones són les que tenen un coneixement global i complet de les llengües, i en aquesta àrea com en moltes es poden produir aproximacions, errors, així que la última fase sempre passa per una d'aquestes persones.

5.2.2 Estandarditzación de l'HTML

L'XML també ha sigut utilitzat pel W3C per definir una versió estàndard de l'HTML coneguda com a XHTML. L'XHTML reté la majoria de funcionalitats que es poden apreciar dins de l'HTML 4.01, però també afegeix un petit nombre de diferències. El que es pretén utilitzant XML és disciplinar i netejar HTML per tal que vagi en consonància amb la família de llenguatges de marcatge.

Aquest primer estàndard nascut va ser el XHTML 1.0, i el seu naixement s'ha produït arrel que en les últimes especificacions s'ha posat poc empeny en que l'HTML complís amb l'SGML. Així doncs, tal i com l'HTML venia amb tres variants, amb l'XHTML passa el mateix, amb els corresponents XML DTDs. El primer DTD correspon al estricte de l'HTML, aquesta definició no suporta els elements obsolets de l'HTML 4.01. Com molts autors troben la versió estricta massa restrictiva perquè molts elements i atributs obsolets encara són àmpliament utilitzats a la web i es pot trobar molt contingut que utilitza aquestes funcionalitats, existeix doncs la versió

transicional. L'únic avantatge respecte a aquest cas, es que amb la versió estricta un autor estarà segur que les següents versions XHTML suportaran el seu document plenament. (Això es pot veure afectat per la nova versió d'HTML 5 i l'impacte que tindrà en les següents versions d'XHTML). L'últim dels tres XML DTD és pels frames. Aquesta versió és idèntica a la versió transicional en els altres aspectes, l'única diferència és la substitució del cos del document pels apropiats elements frames, en comptes de crear una versió de frames per cada una de les altres dues, es va decidir que si es volia utilitzar la extensió obsoleta dels frames segurament es voldrien utilitzar la resta de funcionalitats obsoletes d'HTML.

Una de les diferències que es troba a simple vista amb l'inici d'un document HTML, és la declaració de la versió utilitzada d'XML i la seva codificació. Tot seguit es procedeix a la definició del tipus de DTD escollit pel nostre document HTML, que en aquest cas la diferència ve donada pel fet de si s'utilitzarà un DTD XML.

Un altre element nou dins de l'XHTML i del que ens podrem aprofitar són els espais de noms(namespaces),a l'hora de definir els elements i atributs en el DTD, ho fan dins d'un espai de noms del DTD. Així doncs tenim etiquetes típiques com <table> i els seus atributs, i quan s'hagin de buscar es trobaran al corresponent DTD. Però amb XML es poden utilitzar més d'un espai de noms dins dels documents. Això ens proporciona la capacitat per tenir un document XHTML que validi amb un DTD estricte però que a la mateixa vegada poder utilitzar un mateix element que en un altre DTD fa una altra funció, com una taula de dades per gràfics matemàtics amb unes opcions i format especial.

Aquest fet s'aconsegueix definint quin espai de noms s'utilitza amb l'atribut xmlns, o definint una abreviatura a la part en que comença el document i fent-la servir dins les etiquetes. Les resta de diferències que hi han entre un document HTML i un XHTML les hem comentat anteriorment al parlar dels documents XML ben formats, es tracta de l'anidament correcte de les etiquetes, el tancament de les etiquetes, la sensibilitat a les majúscules i minúscules, els valors dels atributs entre cometes, atributs sense valors, el tractament de caràcters especials. En el cas dels elements que no tenen etiquetes de tancament el que es fa es afegir una barra oblícua(“/”) dins la etiqueta al final, encara que hi hagin atributs, s'afegeix al final.

En el cas dels atributs, en l'HTML es permet utilitzar valors sense estar entre cometes, en aquest cas es obligatori, i també en HTML es veuen afectats els atributs que no tenien cap mena de valor assignat, per aquests casos la solució és

assignar com a valor el mateix nom de l'atribut. I com a diferència no comentada fins ara, ens trobem amb la preferència per l'atribut id que va ser introduït amb la versió 4 d'HTML. En XHTML l'atribut preferit és aquest, i l'atribut name es veu com a obsolet en el seu ús pels elements a, applet, form, frame, iframe i map, però no per button, textarea, select, input, object, param i els meta elements (En la versió 1.1 és obsolet per tots els elements).

Actualment existeix la versió XHTML 1.1, amb l'estàndard aprovat al 2001. El món de la web està girant entorn a la propera versió d'HTML 5, i les properes versions d'XHTML que també s'estan preparant, s'han vist afectades per aquest fet, i arrel de la preparació de l'HTML5 el W3C va decidir acabar amb la preparació del nou estàndard XHTML 2.0, ja que s'havia de preparar l'equivalència de l'HTML5, l'XHTML5. Havent-hi una controvèrsia entre quina de les vessants s'hauria de continuar o si era necessària la especificació de dues versions d'XHTML es va decidir apostar per les característiques de l'HTML5, que resulta ser una versió avançada per part d'un grup que creia que les pròximes versions d'XHTML estaven massa enfocades a l'ús dels documents, i que per tant no encaixaven dins de les necessitats de la xarxa per obtenir noves funcionalitats multimèdia. La nova versió d'XHTML5 al haver desaparegut dels plans del W3C la recomanació de l'XHTML 2 que portaria un gran renovació sobre el llenguatge comparat amb les versions anteriors, sembla ser que l'XHTML queda bastant menyscabat, ja que al haver d'adaptar-se a les noves versions d'HTML s'ha hagut de deixar de banda aquesta renovació ja que no entra dins d'html, i per tant ens trobem amb que la propera versió d'XHTML no serà més que la versió d'HTML 5 escrita en format d'XML.

Els estàndards estan per marcar una guia, però en aquests moments no hi ha cap guia establerta en quant a quin camí seguir per escollir a adoptar-se a les properes versions. Està clar es que les versions d'HTML i XHTML fins ara estan cobertes per una quantitat gegant de documents a la xarxa i això implica que el suporta per part dels navegadors encara persistirà durant molts anys, però per tal d'aprofitar les noves funcionalitats ja s'està començant parts en estat de prova de les futures especificacions, això implica que els autors de nous continguts tot i que han de saber mirar cap al futur també poden quedar-se atrapats u en el passat per tal de satisfer les seves necessitats degut a la incertesa de quin camí es recorrerà.

5.3 CSS (Cascade Style Sheets)

Després de l'aparició de l'HTML i el seu èxit, van començar a crear-se documents amb diferents temàtiques i àmbits a qualsevol lloc, arrel d'aquest continu increment i utilització, també van començar a aparèixer queixes i augmentar les demandes de noves funcionalitats que permetessin als autors dels documents un major control sobre l'aspecte que tindrien els seus documents. Aquestes demandes van ser les que van proporcionar l'aparició d'elements en les futures versions 3.2 i 4 que anirien en endarreriment de la filosofia inicial del llenguatge de marcatge de poder marcar i estructurar els elements, com <font> i <big>. Així doncs els documents van començar a omplir-se de contingut que feia referència a aspectes de presentació del document, això comporta certs aspectes negatius:

- Documents desestructurats fan més difícil i més complexa la indexació. Eines de cerca permetran buscar a través del contingut i de la seva importància dins del document, per aquests casos es necessita de cert marcatge estructural per saber que són els elements i la importància dins d'un document, un exemple d'aquest ús és el cercador Google, que es fixa a la estructuració del contingut entre d'altres aspectes.
- La falta d'estructuració redueix l'accessibilitat, una de les futures aplicacions buscada amb els nous llenguatges i nous navegadors serà l'accessibilitat a usuaris amb dificultats com pot ser la ceguera, a partir d'elements estructurats per la lectura segons el seu contingut i la disposició i importància dels documents serà important, i no un document inestructurat que no es pugui tractar modularment i d'una forma ben organitzada.
- L'estructuració també es necessària per una presentació avançada en la que es necessiti tractar els elements i disposar-los de diferent manera segons el seu significat i disposició dins del document, aquest és un dels punts importants dels que s'aprofita CSS.
- Mantenibilitat. El fet de tenir un document ben estructurat amb el que poder interactuar després per modificar les seves propietats de visualització és molt més usable, llegible i mantenible que no pas haver d'estar modificant tots els elements que estiguin relacionats dins dels documents amb aquestes propietats. El resultat són documents amb codis nets i amb facilitat per mantenir-los, actualitzar-los o cercar dins d'ells.

No es pot negar que la presentació s'ha tornat important en els últims anys i és un aspecte a tenir en compte a l'hora de publicar un document, per això i per intentar minimitzar aquesta problemàtica es va buscar una forma per combinar un marcatge estructurat amb una presentació atractiva de les pàgines. Així doncs el W3C passa a preparar un altre estàndard, el CSS, l'estàndard per definir presentacions de documents escrits en HTML, XHTML, i qualsevol llenguatge que hagi sigut creat amb XML.

Amb CSS els llenguatges (X)HTML poden tornar a encarregar-se de les tasques d'estructuració i de definició tal i com van ser definits. CSS és un llenguatge separat amb la seva pròpia sintaxis, aquesta sintaxis es queda i tracta amb la estructura lògica de la web per transformar i proporcionar unes funcionalitats visibles d'una manera fàcil de manegar, gestionar i a la vegada amb uns efectes molt potents sobre les transformacions que es realitzen. Així amb CSS ens podem minimitzar i/o oblidar de la introducció de nous elements físics per formatjar el document i la presentació per part dels navegadors com va passar anteriorment, aportant les funcionalitats existents i d'altres noves amb CSS.

5.3.1 Beneficis del CSS

- Millor escriptura i control de la presentació. La presentació a través d'elements físics no oferia un control total d'aquests aspectes, CSS ofereix millors eines per administrar-los.
- Menys feina. Es pot canviar la aparença sencera d'una pàgina editant només una fulla d'estil. Fer petits o grans canvis en el disseny d'una pagina amb fulles d'estil és molt més senzill que manegar una barreja entre el marcat i les propietats de visualització en un mateix document.
- Documents més petits i descàrregues més llargues. Amb les pràctiques antigues s'aconseguia tenir elements redundants repetint les mateixes propietas i esquemes que es volien aplicar. Així doncs es poden tenir definides sense haver-les de re-escriure on es vulguin aplicar. Només amb una fulla d'estil es podrien realitzar tots els canvis necessaris en la presentació d'un conjunt de pàgines.
- Tenint tots els aspectes de la presentació dins de les fulles d'estil, el contingut torna a ser ben estructurat i a tenir un significat i així ser més accessibles per altres plataformes o formes d'accedir als continguts.

– Millor suport. El suport dels navegadors pels estàndards és ben conegut, i per CSS no és una excepció, la majoria de navegadors suporten les diverses versions que han anat apareixent.

5.3.2 Els estàndards

La primer versió oficial de CSS, la CSS Level 1 Recommendation, CSS1, va ser lliurada oficialment al 1996, incloïa propietats per afegir fonts, colors, instruccions d'espai als elements de la pagina. Per desgràcia, la falta de navegadors dependents de l'estàndard va fer que la adopció majoritària trigués més en arribar. CSS Level 2 (CSS2) va ser lliurat al 1998, va afegir propietats per el posicionament que permetia que CSS fos utilitzat per la determinació de la disposició dels elements. També afegeix estils per altres tipus de recursos multimèdia com aural(per veus), print (per les impressions), handheld(per dispositius mòbils) i més mètodes per seleccionar els elements a estilitzar. CSS Level 2, Revision 1 (2002) soluciona errors de CSS2 i elimina funcionalitats poc suportades i afegeix algunes extensions implementades ja pels navegadors a les especificacions.

Actualment CSS3 s'esta redactant, però en aquest cas se sap que les especificacions seran modulars, així es diu que serà més fàcil manegar-les, i actualitzar-les que sent una especificació monolítica. Com a funcionalitats a destacar, vindran noves funcions per colors, fons, efectes de text com ombres, interacció amb l'usuari com ajustar el tamany, navegació, i tal i com especifica, nous mòduls com speech, fonts web o media queries amb les que depenent de certes condicions com una alçada i una amplada específica, es podrà canviar les fulles d'estil a unes altres automàticament.

Dins dels nostres documents tenim diverses opcions a l'hora d'afegir fulles d'estils en cascada:

– Poden ser introduïdes en la capçalera dels documents utilitzant, s'utilitza l'element <style>. Dins d'aquest element es poden trobar atributs tots ells opcionals com media que ja hem comentat.
– Dins del cos del document amb els elements , . Permeten aplicar CSS específics a algunes seccions. Aquests elements només tenen coma atributs style, per definir dins del mateix element l'estil, o bé id i class, els quals es poden fer servir per relacionar-los amb estils ja definits a la capçalera o en un CSS extern.

– Utilitzant documents externs que són referenciats des de la capçalera i que contenen els CSS. Afavoreix la gestió de una col·lecció de documents, així com ja s'ha comentat si es vol fer un canvi mínim o global per tal de que es reflecteixi no s'ha de modificar el contingut, solament s'hauria de modificar aquests fitxers que són referenciats.

Tots els estils tenen una estructura comuna, i es basen en el mateix concepte i sintaxis per dir que es el que es fa i a que es fa. Així doncs trobem dos principals elements, la declaració de les propietats que es voldran efectuar, i els elements destinataris que rebran aquesta transformació. Els elements que estan dins d'aquesta estructura de la que es parla són:

– Selector. És l'encarregat de dir que element o elements han de ser seleccionats per veure's afectats. Els selectors són una part fonamental que es defineix en el llenguatge CSS ja que permet seleccionar els elements que volem de diferents maneres, així tenim al nostre abast vies molt flexibles per tal de seleccionar i especificar quins elements concretament volem que siguin afectats i no uns altres. Es poden utilitzar des de d'un grup d'elements típics (el que s'anomena agrupació) com poden ser els headers, a fins i tot utilitzar els atributs com id i class, o també definir un element dins d'un anidament en concret, o bé pel seu estat si estar marcat, seleccionat...les opcions són molt amplies.
– Bloc de declaració. És com es coneix al codi que va després del selector. Comprèn totes les opcions de formatjat que es volen aplicar als elements designats pel selector. Aquesta part és la que està continguda dins dels braces d'obertura i tancament({}).
– Declaració. Són els elements que trobem dins del bloc de declaració, cada una d'aquestes indican una instrucció a aplicar a l'element en el que està continguda. Cada una de les declaracions a la vegada conté un parell de propietat i valor acabat en punt i coma.
– Propietat. El conjunt de opcions o instruccions que es poden aplicar als elements mitjançant CSS s'anomenen propietats. Són noms (normalment bastant auto-descriptius) que indican a que afectaran de l'element en qüestió.
– Valor. Aquest camp conté la opció escollida dins d'un rang de valors acceptats per la propietat a la que es refereix, acaba de completar i determinar quina acció específica es durà a terme per la propietat donada.

N'hi han també d'altres aspectes importants que s'han de tenir en compte i que són de gran ajuda a l'hora de com aplicar els estils per facilitar, alleugerar i fer la tasca menys redundant.

Un aspecte es l'herència, així com s'aplica en altres aspectes de les tecnologies i llenguatges de programació, CSS permet que els elements html puguin heretar les propietats CSS dels seus antecessors. En CSS això s'aplica fent que les propietats descrites per cert element puguin anar passant cap els seus elements fills. I com sol ser habitual la herència no està limitada a un sol nivell, en aquest cas parlem a través de generacions, i per tant si tenim un element que hereta dels seus antecessors, però a la vegada els seus antecessors també han heretat algunes propietats, aquestes propietats heretades pels antecessors també arriben als elements en qüestió. Un altre aspecte important és el concepte de cascada que conté el mateix nom de CSS. És molt probable que les CSS estiguin ubicades en diferents llocs, tal com diferents fulles externes, o tenir definit un estil intern i un un altre referenciat externament, aquests casos poden provocar col·lisions entre propietats que han d'afectar a un mateix element. Per aquest possible problema es va dissenyar sistema jeràrquic per resoldre el problema. El terme cascada fa referència al que passa en aquests casos, van passant la diferent informació d'estils per sobre i es queda la propietat amb més pes de totes les que poden colisionar.

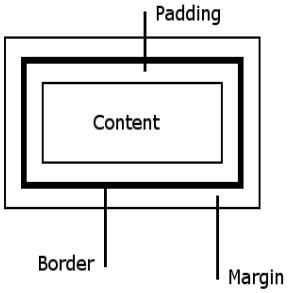
El pes que es dóna a les regles en aquest sistema ve donat per certes regles:

- Les regles aplicades als pares si no entren en conflicte amb regles més específiques per l'element seran heretades.
- Les regles CSS són ordenades pel seu pes, i qualsevol marcada amb !important te precedència sobre la resta.
- Qualsevol regla importada per un CSS extern amb <link> serà sobreescrit per qualsevol regla que estigui dins del document.
- Regles més específiques sobreescriuen a les més generals.
- Si dues regles sembla que tenen el mateix pes guanya l'última en ser especificada.

El resultat d'aquests esquemes de cascada és que es poden tenir múltiples estils definits sense haver d'estar concentrats en tots els aspectes i cadascun dels elements, ja que mitjançant cascada i herència es poden obtenir resultats desitjats tenint CSS menys complexos i més flexibles.

I per últim, dins del CSS també hi han propietats que donen la funcionalitat per poder disposar els elements de la forma que ens sigui més convenient. El més interessant i important per entendre com es es produeixen aquestes transformacions i com s'han de fer, és el model que hi ha per tractar els elements

dins dels documents, el model de caixa. Podem simplificar-ho dient que els navegadors veuen cada element dins del document com si estigués contingut dins d'una capsa rectangular. I és en aquesta capsa a la que se li poden aplicar aquelles propietats anomenades box propierties. Entre elles trobem borders, marges, espaiat, fons o posicionament d'aquestes.



Com podem veure el model de capsa està ben estructurat i diferenciat i és fàcil veure com afecta a l'element cadascuna de les propietats relacionades. Així doncs podem diferenciar diverses àrees:

- Àrea del contingut. Es la part principal de l'element, ja que és on va el contingut de l'element que representa.
- Marges interiors. Es refereixen a la frontera entre el nucli de l'element i la resta d'àrees de la capsa, i tenen l'amplada i alçada determinada per camps width i heigh.
- Padding. És l'àrea que hi ha entre l'àrea del nucli on hi ha el contingut, i el marge exterior, i és opcional.
- Margin. És una quantitat extra d'espai que es pot afegir opcionalment fora del marge exterior de la capsa.
- Marges exteriors. Són els marges que rodegen tot el conjunt de la capsa, no hi ha cap part fora d'ell. Aquesta és l'àrea total que ocupa l'element en la pàgina.

Totes les àrees comentades existeixen dins de cada element de la pàgina, però depenent de les propietats que siguin utilitzades es modificarà la seva mida i/o aparença, ja que totes aquestes àrees tenen dedicades un conjunt de propietats per tal de poder personalitzar la capsa del propi element i poder així situar-lo i deixar-lo com vulguem.

5.3.3 YUI CSS Reset (Yahoo! User Interface)

Es tracta d'un framework CSS que elimina i neutralitza les inconsistències d'estils d'HTML que proveeixen els propis navegadors, que propicien que segons l'utilitzat es podran observar diferències en l'aparença a causa d'els atribut predefinits que pot tenir cada navegador i que divergeixi entre ells, com passa amb el tipus de font, la grandària,... A diferència d'altres frameworks CSS yui reset deixa tots els elements de text amb el mateix aspecte, ja sigui <p>, , <h1>. Per aquesta raó les pàgines a les que s'aplica únicament reset.css no estan llestes per ser mostrades a l'usuari, a diferència d'altres frameworks que acte següent de neutralitzar apliquen uns estils correctes per la visualització. Però opcionalment també es proveeix d'un altre component en el framework YUI nomenat base.css per aplicar aquests estils als elements que han sigut neutralitzats. El motiu pel qual aquest comportament no ve per defecte és que així un cop neutralitzats un desenvolupador pot explicitar els seus propis estils sense haver de preocupar-se per cap altres, ja que Reset s'haurà encarregat d'ells.

5.4 Javascript

HTML encara que tingui avantatges, també té de desavantatges pels desenvolupadors quan es tracta de proveir un major control sobre els elements del document a l'hora de crear pàgines i aplicacions. Aquestes limitacions van desencadenar en l'aparició i utilització de programes o scripts des del cantó del servidor per manegar alguns aspectes dinàmics del comportament de les pàgines que els desenvolupadors necessitaven. Aquests programes ofereixen la possibilitat d'obtenir més funcionalitats a l'hora d'interactuar amb els usuaris i tractar la informació que es derivava.

Però també hi havia un desavantatge en el tractament de les dades, en el cas que hi hagués un error o no es complissin certes condicions es produïa un tràfic innecessari tal que si abans d'enviar al servidor les dades s'hagués pogut comprovar, s'hauria estalviat doncs aquesta comunicació. Així va ser com es va fer present que era necessària una tecnologia que oferís la capacitat per executar aquestes tasques als navegadors dins les màquines dels usuaris, i que a més oferís la possibilitat de cedir part de la càrrega de feina que es localitzava tota al servidor fins al client (sempre dins d'uns límits) per balancejar i obtenir un millor rendiment global.

Llavors va ser al desembre del 2005 que Sun i Netscape van introduir Javascript 1.0, en aquells temps nomenat com Livescript. Es tractava d'un llenguatge que podia ser interpretat pel navegador Netscape Navigator 2. Així es va obrir la veda al fet de no només formatjar informació com es feia amb HTML, sinó que com a complement de Java es podria programar aplicacions per la web. Degut al bon acolliment que va tenir Javascript, els desenvolupadors van demanar que hi hagués la implementació equivalent de Javascript també en el cantó del servidor, així podrien programar en el servidor de la mateixa manera que es feia pel cantó del client i no haver de utilitzar tecnologies diferents per les mateixes aplicacions. El nivell d'acceptació no va arribar a ser global fins que Microsoft va afegir la seva implementació del llenguatge en el seu Internet Explorer 3, anomenada Jscript i Jscript 2.0 amb el suport pel cantó del servidor. Els principals aspectes que han afectat al desenvolupament i que s'han hagut de dominar dins de JavaScript, poden arribar a divergir molt en certs aspectes de Java, llenguatge amb el que en un principi es pot confondre i pensar que es tractarà d'una versió lleugera amb funcionalitats mes dirigides a la web. Aquests aspectes han sigut:

- Comparació amb Java. Degut a la relació amb Sun, i el seu nom, en un principi es pensa en una versió lleugera de Java a la web, però això no és cert, hi han diversos aspectes que divergeixen bastant l'un de l'altre, no obstant hi han aspectes, com la sintaxis, i les estructures de control, les quals la majoria són suportades en Javascript, però per una altra banda és un llenguatge molt més lliure.
- Sintaxis. La majoria de la sintaxis que ens trobarem, operadors i les estructures de control són prou semblants a les de Java.
- Orientació a objectes vs Prototipatge. A diferencia de Java que és un llenguatge orientat a objectes per excel·lència, Javascript es basa en un altre paradigma de la programació, bastant proper però més limitat en algunes aspectes. Això ajuda a que el llenguatge sigui menys complex i més ràpid de aprendre i mantenir. També gràcies a que es permet crear elements semblants a objectes i estendre els del nucli prototipant noves propietats ajuda a salvar aquestes limitacions que ens podríem trobar. Així a diferencia de Java que hi han les classes, ens trobem objectes, i el que es fa és tenir instàncies que serveixen de prototipus que es clonen per poder tenir aquesta sensació de classe amb els mateixos atributs i comportament, a aquest paradigma se li diu programació basada en prototipus. Hi ha certa critica davant aquest paradigma, en el que es parla de predictibilitat, rendiment, i robustesa, però com tot es sacrifiquen certs aspectes en detriment d'un altre.
- Tipatge dinàmic i dèbil. A diferencia de Java el tipatge és dinàmic ja que es realitza en temps d'execució a diferencia de ser definit durant la compilació. També es diu que és un tipatge dèbil ja que permet l'ús d'un tipus d'objecte en l'àmbit d'un altre, en el cas de Javascript el tipus de dades es va convertint automàticament quan es necessari durant l'execució del codi sense provocar errors.
- Objectes Javascript. Un altre aspecte que resulta un dels pilars en els que es basa la programació de Javascript són els conjunts d'objectes que hi ha representats. Els trobem agrupats en diferents conjunts i la majoria depenen del context pel qual s'està desenvolupant.
- Core. Són els objectes que formen el nucli dels tipus d'objectes bàsics de Javascript, n'hi han alguns que són estàndards a Javascript, però també n'existeixen uns altres implementats per cada navegador com a resultat de les extensions que ofereix cadascú del seu producte.
- Client-Side:

- DOM (Document Object Model). Abans de suportar-se DOM els navegadors més antics treballaven amb altres models, amb DOM s'obté un model d'objectes dels elements dels documents que és neutre en quant a llenguatge i plataforma, així s'obté una base comuna en la que desenvolupar per tots els navegadors. Aquest model permet l'accés i modificació del contingut, estructura i estils de documents, que no necessàriament han de ser XML o (X)HTML, tot i que el seu ús està aplicat majoritàriament a aquests casos. El model té una estructura jeràrquica dels elements dels document, a través del qual es pot accedir a qualsevol element i tots els seus atributs. A més conté una API amb la que es pot creat, eliminar, modificar elements i continguts. DOM es suportat per la gran majoria de navegadors actuals. El model es basa en la idea de que els documents estan composats per 3 components, els nodes que representen els continguts siguin els que siguin, elements que són les unitats que poden contenir atributs, i els atributs tals com l'adreça d'origen d'una imatge. DOM ens proporciona diferents formes per accedir a aquests components, però totes són seguint la estructura jeràrquica del document, primer a través del node arrel que és el propi document, i després es van seleccionant nodes, les possibilitats són moltes, des de utilitzar els identificadors, atributs, fent ús de les relacions entre nodes dins del document...També es pot utilitzar conjuntament amb CSS, ja que es proveeix de funcions per accedir a les pròpies regles i modificar-les si és necessari.

– Server-Side: Hi han moltes diferencies entre les plataformes que es poden trobar, com a exemple molt estès està la solució de Microsoft IIS + ASP.
– Ajax(Asynchronous Javascript And XML). Ens proporciona la capacitat per comunicar-nos entre el client dins de la mateixa pàgina, i el servidor. Funciona de tal manera que s'envia una crida al servidor i el servidor després de rebre i processar-la adequadament l'envia de retorn amb un conjunt de dades. L'avantatge d'aquesta tecnologia és que aquesta feina es realitza transparentment, així quan s'envia la petició el client no s'atura i s'espera a rebre la resposta, d'aquí ve el terme Asynchronous. Dins el client s'observa el resultat, i es crida a una funció quan el resultat de la petició canvia, un cop es detecta es realitza la validació per tal de comprovar el resultat i decidir que es fa. El conjunt de tecnologies que s'utilitzen : XHTML, XML, HTTP i Javascript són tecnologies que existeixen des de fa molt de temps i per amb una maduresa suficient com per ser utilitzades en aquest context. Junt amb DOM s'obre la veda a una nova visió del funcionament dels documents dins la xarxa, podent passar

de documents completament estàtics, a documents i continguts completament dinàmics en els que mitjançant comunicació asíncrona amb el servidor es pot crear una aplicació basada en la interacció del client per modificar i actualitzar els continguts segons li convingui eliminant la sensació d'estar navegant entre documents separats. Amb aquesta tecnologia també s'ha possibilitat que amb facilitat es puguin fer ús de llibreries i serveis webs ja existents per altres empreses o desenvolupadors, mitjançant crides es pot obtenir el codi i establir la comunicació necessària per afegir noves funcionalitats dins d'un document propi.

5.4.1 jQuery

Tal i com acabo de comentar al descriure les aplicacions d'AJAX, en els últims anys per tal de facilitar i a l'hora donar més aplicacions als desenvolupadors han aparegut una serie de frameworks com és jQuery que he fet servir durant el projecte, que ve amb uns models ja preparats per realitzar un conjunt de tasques més o menys comunes com poden ser un conjunt d'efectes visuals que avui en dia ens trobem en pàgines i aplicacions dinàmiques (auto-completar, drag & drop, animacions de desplegament, col·lapsar elements).

Com a aspectes favorables per jQuery trobem que és tracta de una llibreria de Javascript molt poc pesada (24 kb comprimida), preparada per treballar amb els últims estàndards CSS, inclosa la futura tercera versió que està en esborranys. També és multi-plataforma i està suportat per la majoria de navegadors actuals, aspecte molt important a tenir en compte ja que gràcies a la utilització d'un framework testejat i que funciona correctament amb diversos navegadors s'estalvia la feina extra de comprovar un bon funcionament en les diferents plataformes i crear un codi o fins i tot codis alternatius per segons quines versions o navegadors.

Un altre punt a favor es la seva senzillesa i reduïda grandària. I per si no hi hagués suficient amb les funcionalitats que proporciona out-of-the-box, ens permet estendre les nostres aplicacions fent us de plugins desenvolupats per utilitzar amb jQuery, de fet també proporciona la possibilitat de que un mateix pugui fer aquests plugins i afegir-los a la comunitat.

5.4.2 jQuery UI

També s'utilitza jQuery UI (User Interface), el qual proporciona abstracció per interaccions de baix nivell i animacions, efectes i una gran capacitat per fer el teu propi tema amb els elements de la interfície que s'utilitzaran personalitzats i tenir una interfície pròpia i adaptada al teu disseny, tot això utilitzant la llibreria de jQuery per ajudar a desenvolupar aplicacions webs interactives amb l'usuari.

Un dels punts positius d'utilitzar jQuery UI és la modularitat, carregant el nucli per separat només has de seleccionar en cada cas la part de llibreries de UI que necessites en cada moment per tal de carregar sempre només les llibreries a utilitzar i així evitar carregar parts de les llibreries innecessàries perquè no seran utilitzades. Si només volem utilitzar botons amb UI, carreguem la llibreria jquery.ui.buttons.js i ja podrem utilitzar les funcionalitats relacionades amb els botons de UI, però no les d'altres com els diàlegs per exemple.

5.5 PHP

PHP és un llenguatge de programació per crear pàgines webs interactivament des de la computadora que les servirà, els servidors web. Però a diferencia de l'(X)HTML que utilitza etiquetes perquè el navegador pugui construir la pàgina PHP executa codi entre la pàgina que s'ha sol·licitat i el servidor web, afegint així i modificant el contingut de la sortida HTML que es produirà. Llavors tenim un llenguatge amb el que podem realitzar tasques que podran variar i extreure contingut variable, que és la idea clau, poder generar codi HTML per visualitzar dins d'un document.

La principal diferència entre accedir a una pàgina sense PHP o amb PHP és que, en el primer cas al servidor web li arriba la crida del recurs demanat, s'accedeix a disc, s'obté el recurs i es retorna el seu contigu; però en el segon cas, el servidor web en comptes de retornar el contingut passa el fitxer al interpret PHP que s'executa sobre el fitxer i es passa el resultat obtingut al servidor web que ho retornarà al navegador del client.

Els inicis de PHP van ser curiosos, tant en essència com en taxonomia no s'assemblaven en res al llenguatge que avui en dia coneixem. La primera aparició de PHP, amb la versió I, va aparèixer a l'any 1995, i va ser anunciat com a un

conjunt d'eines, un framework que oferia funcions per enregistrar dades d'accés en logs i visualitzar-los, i també funcions per gestionar l'accés d'usuaris als dominis, tant per protegir la seva informació d'accés com per gestionar els formularis que hi havia d'haver entre mig. La lògica però estava escrita en C i s'encarrega de analitzar el codi HTML i cridar a les funcions.

Al 1996 va ser quan PHP va girar en direcció contrària amb l'aparició de la seva segona versió. Va passar a ser un llenguatge d'scripting, i el parser va ser reemplaçat per un altre de més sofisticat. El motiu principal d'aquest canvi va ser que els pocs usuaris que van utilitzar la primera versió demanaven poder incloure lògica de control dins de les mateixes pàgines per generar el contingut HTML en un llenguatge semblant a C. Una de les principals aplicacions que volien utilitzar era poder enviar diferents blocs HTML sota diferents condicions. Aquest fet va ser com va néixer l'if a PHP, i un cop tenint if, van anar apareixent la resta d'estructures de control i a partir d'aquesta base es va acabant fent un llenguatge d'scripting sencer.

Per la versió 3 de PHP ja es van interessar més desenvolupadors i va deixar de ser un projecte d'una persona, després amb la versió 4 i afegir una capa d'abstracció entre el llenguatge i el servidor web, nous mecanismes de seguretat i un nou parser. La versió 4 de PHP va ser amb la que es va fer més famós i va aconseguir obtenir una quota de mercat molt gran entre els llenguatges de programació web. Una de les grans millores i aportacions de la versió 5, que és la que ens ocupa dins del projecte, va ser la millora significativa del suport a la orientació a objectes.

La sintaxis es bastant semblant a altres llenguatges semblants com poden ser C i Java, hi ha la declaració de la classe amb els seus camps, sense el tipus que es determina durant l'execució, i les funcions que en aquest cas es tracten de funcions creadora, getters i setters per manipular els atributs de les instancies de les classes.

5.6 Symfony

Symfony es un framework que ens vol oferir un marc més eficient pel desenvolupament d'aplicacions utilitzant i automatitzant molts patrons que s'utilitzen durant el desenvolupament d'una aplicació. Un framework com Symfony també proporciona una estructura pre-establerta en el codi per obtenir-ne de millor qualitat, més llegible i mantenible. I com ve sent habitual aquest framework també proveeix d'un conjunt de funcionalitats i operacions accessibles des de simples crides per fer el procés d'implementació més senzill.

El framework està dissenyat per optimitzar el desenvolupament d'aplicacions web a partir d'un conjunt de funcionalitats. Primerament separa les regles de negoci, la lògica del servidor i la presentació. Conté un gran número de classes i eines presents per escurçar el temps que es triga en desenvolupar una aplicació web. També automatitza certes tasques de gestió perquè el desenvolupar es pugui centrar en els aspectes específics de desenvolupament de l'aplicació. El resultat es que a l'hora de desenvolupar diverses aplicacions no has de preparar i configurar un entorn per cada una.

5.6.1 Visió general

Tot el seu conjunt està programat completament en PHP5. És utilitzat en nombrosos projectes reals arreu del món i actualment s'utilitza per desenvolupar aplicacions e-business d'alta demanda. És compatible amb la majoria dels motors de bases de dades com MySQL, PostgreSQL, Oracle, SQL Server, SQLite, i funciona tant en plataformes UNIX com en Windows.

Característiques principals de Symfony:

- Fàcil d'instal·lar i configurar en la majoria de les plataformes (provat i garantit per UNIX i Windows)
- És independent del motor de base de dades que s'utilitza
- Senzill d'utilitzar, però tot i així s'adapta a necessitats complexes (amb més o menys extensions)
- Basat en la premissa de convencions per sobre configuracions, el desenvolupador només hauria de configurar el que se sortís del normal
- Segueix la majoria de bones practiques de programació i patrons de disseny

- Preparat per empreses, i suficientment estable per projectes de llarg termini (no perquè el desenvolupament sigui lent, sinó perquè les versions son robustes)
- Codi molt ben estructurat i llegible, fàcil de mantenir i actualitzar
- Fàcilment extensible utilitzant llibreries d'altres desenvolupadors

També n'hi han moltes funcionalitats que podem trobar a la web que estan automatitzades dins de Symfony:

- La internacionalització i localització té una capa interna dins de Symfony que permet tant la localització de les dades com de la interfície, també té funcionalitats relacionades amb la internacionalització de les interfícies.
- La part de la presentació està separada i poden ser fetes per dissenyadors webs sense que hagin de tenir cap mena de coneixement del framework. Però, en la part de la presentació l'ús de Helpers pot comportar un estalvi en redundància a l'hora d'escriure porcions de codi tant PHP com HTML important.
- Suport excepcional de formularis. Permet automatitzar les tasques que involucren la gestió dels formularis, validació, gestió d'errors, gestió de les relacions entre els formularis i les dades que representen en la base de dades, que milloren la interacció amb l'usuari utilitzant un esquema i procés de gestió preparat i controlat.
- El mecanisme de decoració de la sortida protegeix a les aplicacions de ser atacades per tal d'evitar que les dades es puguin tractar i executar algun codi maliciós.
- Les funcionalitats de la caché redueix l'ample de banda i la càrrega del servidor.
- Funcionalitats de credencials i autenticació faciliten la creació de restriccions segons el context i instaurar una gestió dels aspectes de seguretat.
- El sistema d'adreçament fa que les adreces de les aplicacions puguin formar part de la interfície i utilitzar-se per cerques i accedir fàcilment a l'aplicació.
- Funcionalitats de correu electrònic i d'altres APIs que proporcionen interaccions dins l'aplicació amb aspectes típics de la web.
- Llistat d'elements funcional, àgil i senzill de preparar gràcies a la automatització de la paginació, ordenació i filtratge.
- Factories i plugins aporten una alta capacitat per la extensió de les funcionalitats de les aplicacions.
- Fàcil implementació d'interaccions AJAX mitjançant Helpers que encapsulen funcionalitats compatibles en la majoria de navegadors.

Symfony per encaixar dins de projectes empresarials aporta un entorn completament configurable i adaptable en tots els seus aspectes, però per defecte

ofereix uns entorns preparats i un conjunt d'eines per automatitzar tasques relacionades amb la enginyeria de software:

- Eines de generació de codi basades en prototips i generació de entorns de administració a l'instant.
- Framework pel desenvolupament de proves que proporciona eines i funcions per desenvolupar basant-se en proves.
- Un panell de depuració per facilitar les tasques de depuració mostrant i informant dades i estadístiques sobre el funcionament de les parts involucrades en la tasca executada.
- Els logs ofereixen un alt nivell d'informació amb informació detallada sobre les activitats que es produeixen en la aplicació.
- Permet la modificació en calent de la configuració i la execució de l'aplicació.
- La interfície de la línia de comandes automatitza el traspàs d'aplicacions a altres servidors.

5.6.2 Orígens de Symfony

La primera versió de Symfony va sortir al 2005 pel fundador del projecte Fabien Potencier. Fabien és el CEO de Sensio, una agència web francesa pionera en el desenvolupament web. Symfony va néixer arrel de l'interes de Fabien per les eines de software de codi obert en el desenvolupament web en PHP. Al trobar que cap de les existents complia amb els requisits que ell necessitava com són els que s'han descrit anteriorment, i amb l'aparició de PHP 5 va decidir que es podria desenvolupar un framework amb totes les funcionalitats necessàries. Basant-se en el framework Mojavi també amb patró model-vista-controlador(MVC), la eina Propel pel mapatge d'objectes a la base de dades (ORM Object-relational-mapping) i els Helpers utilitzats per Ruby on Rails en les plantilles, es va desenvolupar el nucli de Symfony.

La motivació personal per desenvolupar el framework va ser poder utilitzar-lo dins de la empresa Sensio. Després d'utilitzar-lo en diversos projectes Fabien va decidir lliurar-lo perquè la comunitat pogués aprofitar la seva feina, i així també poder aprofitar la resposta de la comunitat d'usuaris i desenvolupadors per potenciar el propi framework (també diu que ho va fer perquè és divertit). Per tal que el projecte tingués una bona acollida entre els usuaris es va encarregar de crear una documentació suficient com per incloure el funcionament, configuració i la utilització del framework, és per això que en general la majoria dels aspectes que es

poden trobar a Symfony es poden investigar, entendre i dominar al nivell que es necessiti, des de saber lo just per utilitzar-lo com per aprofundir en els mecanismes de funcionament i personalitzar-los.

La comunitat es va posar del cantó del framework degut que en aquells moments la necessitat d'un framework complet pel desenvolupament d'aplicacions web en PHP estava molt demandat. Symfony a diferencia de la competència oferia una qualitat de codi i una quantitat de documentació que no es trobava en altres. Com molts grans projectes de software de codi obert ofereix la possibilitat de participar dins del projecte gràcies al sistema de tiquets, l'accés al projecte mitjançant un repositori public, i múltiples altres formes de contribuir tant com a col·laboradors, tenint sempre al creador com a principal mantenidor i encarregat dels canvis finals, per tal d'assegurar un codi amb una garantia de qualitat.

A més de la documentació oficial es pot trobar dins de la comunitat una gran quantitat d'informació i d'ajuda tenint el fòrum oficial en el que tant el creador com els col·laboradors i moderadors són prou actius com per resoldre la majoria dels dubtes en poc temps als usuaris. Llistes de correu, canals IRC, wiki, suggestions de fragments de codi, tot el conjunt de la comunitat es una font inesgotable de coneixements i recursos pel desenvolupament d'un projecte amb Symfony.

Per saber si es té la necessitat d'utilitzar un framework com Symfony s'ha d'analitzar la grandària i complexitat del projecte que es vol portar a terme. Si es vol desenvolupar un lloc web que consti de poques pàgines, que la estructura de dades no comporti un gran model conceptual ni moltes relacions entre els seus components, que no ens preocupi el rendiment general, o un control, manteniment i administració important, llavors serà millor fer-ho únicament en PHP. En general utilitzar un framework quan no es cal comporta una ralentització i complicació extra en el procés de desenvolupament sense obtenir res a canvi.

Si no es troba en els casos anteriors i es vol desenvolupar una aplicació web més complexa, amb un component de lògica de negoci important PHP no acostuma a ser suficient. Si el projecte necessita d'un manteniment, o poder estendre les funcionalitats i capacitats, si es volen utilitzar els avantatges en la interacció amb usuaris o amb altres tecnologies de la web de manera intuïtiva, si es vol desenvolupar d'una manera àgil i efectiva, amb només PHP el procés es pot tornar molt desestructurat i caòtic.

5.6.3 Conceptes fonamentals de Symfony

PHP5

Symfony esta desenvolupat completament en php5 i s'utilitza per generar aplicacions webs en el mateix llenguatge. Un dels requisits per poder utilitzar-lo es el coneixement del llenguatge, tot i que no es necessita un gran grau o nivell de domini del llenguatge per poder-lo utilitzar, per aprofundir dins del seu funcionament i buscar modificar el seu comportament o afegir altres funcionalitats no disponibles amb plugins o altres tecnologies serà molt convenient tenir uns coneixements de programació en PHP i la seva orientació a objectes més avançats.

Programació orientada a objectes(OOP)

És un prerequisit per poder entendre com funciona Symfony, ja que fa un ús exhaustiu dels mecanismes que aporte la orientació a objectes de PHP5. I per tal de poder aprofitar aquests mecanisme es necessari entendre conceptes com classe, objecte, mètodes, herència, auto-carrega, sobrecàrrega, serialització, etc...

Mètodes màgics

Una altre de les funcionalitats que ofereix PHP 5 relacionada amb els objectes és la capacitat de definir mètodes per modificar el comportament sense haver d'afectar el codi exterior. Així el comportament d'una classe per una funcionalitat comuna en el llenguatge es pot definir dins de cada classe que es vulgui perquè el seu funcionament s'adapti i compleixi amb les necessitats de la classe per la que es defineix, així s'aconsegueix obtenir una sintaxi més comuna amb un comportament personalitzable i configurable segons les necessitats. El cas més famós i a la vegada important dins de Symfony es el mètode __toString, aquesta funció s'utilitza per convertir un objecte a una cadena de caràcters que mostrar per la sortida corresponent, aquesta funció és molt utilitzada pels Helpers i els formularis:

També n'hi han d'altres mètodes màgics molt utilitzats com són __construct(), funció constructora d'una classe, __call(), utilitzada quan es crida un mètode que no existeix dins la classe i es pot fer servir per mostrar un error personalitzat per exemple, o __autoload() que defineix el comportament en una classe quan s'intenta carregar un objecte d'una classe que no ha sigut inclosa mitjançant alguna de les funcions include, require, o require_once.

PEAR(PHP Extension and Application Repository)

És un framework i sistema de distribució per components PHP que permet descarregar, instal·lar, actualitzar i treure scripts PHP. Les bonances d'utilitzar els paquets de PEAR és el fet de no haver de preocupar-se on posar aquests scripts, com fer-los disponibles en cas de voler-los compartir, o com afegir les noves funcionalitats a la línia de comandes. El propi framework Symfony es recomana instal·lar-lo via PEAR per aquests motius, i a l'hora de voler afegir plugins que estiguin llistats dins la wiki de Symfony(el recurs principal on es poden cercar plugins de tots tipus) es poden instal·lar via PEAR ja que són disponibles en format de paquets PEAR. Symfony ja proveeix de les funcionalitats necessàries per poder-los instal·lar, ja sigui localment o remotament.

ORM(Object-Relational Mapping)

Al treballar amb bases de dades ens trobem amb el problema de que les dades que es guarden a la base de dades es guarden de forma relacional (si la base de dades es relacional, com ve a ser la tendència actualment per representar les relacions que s'estableixen entre els objectes) però les dades de PHP5 i de Symfony estan orientades a objectes, llavors hi ha d'haver un procés que faci d'intermediari i s'encarregui de traduir la lògica de l'objecte a la lògica relacional, i aquest intermediari es la interfície que s'anomena ORM.

La interfície ORM mateixa està feta a base de objectes que són els que proporcionen les funcionalitats encarregades d'accedir a la base de dades per obtenir la informació. Com a conseqüència s'obté una capa d'abstracció entre la base de dades i l'aplicació. Aquesta capa afavoreix en el sentit que l'aplicació no estarà desenvolupada per treballar específicament amb una base de dades concreta, ja que la pròpia lògica de l'ORM s'encarrega de traduir les crides en el nostre llenguatge a consultes en SQL que estaran adaptades i optimitzades per la base de dades que estiguem utilitzem en tot moment(sempre i quan la suportin, però aquestes eines solen tenir suport per totes les grans solucions de base de dades del mercat), de manera que en qualsevol moment serà possible fer una migració a un altre motor de base de dades sense que el codi de l'aplicació es vegi afectat (no el que nosaltres desenvolupem).

Una de les aplicacions que ha tingut durant el projecte és que durant el desenvolupament al no estar lligat a una base de dades es podia tenir per fer proves locals l'aplicació funcionant sense cap servidor de base dades engegat fent ús de bases de dades SQLite, i a l'hora, es podien fer proves amb el mateix codi en el servidor utilitzant el servidor MySQL modificant només una petita configuració de les bases de dades a utilitzar, i obtenint el mateix funcionament. Com a limitacions es poden trobar algunes a l'hora d'executar segons quines consultes, com pot ser la utilització de la clàusula DISTINCT ja que en el cas de SQLite no està suportada, aquestes limitacions venen donades per les diferencies i limitacions que pot haver en cada motor.

Aquesta interfície facilita la tasca de transport entre una banda i l'altre de l'abstracció de la capa de dades i poder concentrar-se en la utilització dels models de dades transformats. i així es poden estendre les capacitats d'accés a la informació que hi haurà a la base de dades, ja que és molt probable que per cada classe representada a la base de dades, dins l'aplicació s'utilitzi s'hagi de definir comportaments relacionats amb la forma de realitzar operacions a la lògica de negoci ja que l'ORM s'encarrega de generar una base i un entorn amb el que poder utilitzar fàcilment les relacions que hi han entre els objectes i les taules mapejades a la base de dades. D'aquest tema es parlarà més endavant en l'apartat de l'ORM utilitzat, Propel.

5.6.4 Metodologies de desenvolupament

Els cicles de vida de l'enginyeria de software usuals com els d'una metodologia com Procés Unificat de Rational comprèn moltes fases fins que no es compleixen certs requisits, i especificacions. Altres metodologies han aparegut que difereixen de les metodologies de desenvolupament software clàssiques amb l'aparició d'eines i uns llenguatges amb una versatilitat i facilitat per començar un projecte. Aquestes metodologies intenten obtenir resultats amb els que poder analitzar i veure la progressió del procés amb els que anar dissenyant segons es vagi obtenint resultats, de tal manera que la falta d'aquesta planificació i preparació excessiva es transforma en un desenvolupament de funcionalitats més ràpid i sense haver d'estar marcats per un esquema rígid poc sensible als canvis que s'hagin de realitzar durant el procés de desenvolupament que modifiquin els plans previstos inicialment. Les metodologies que s'utilitzen amb Symfony i durant la realització del projecte són:

- Rapid Application Development(RAD). Aquesta metodologia conté tècniques com el desenvolupament iteratiu i prototipatge de software, en el que les funcionalitats es van dissenyant per iteracions en les que s'obtenen funcionalitats parcials fins arribar a la completa. Aquest esquema es combina amb l'ús de models incomplets que serveixen de base per començar a desenvolupar i que a mida que s'avança es van completant i estenent.
- Keep It Simple Stupid(KISS). Es busca evitar la complexitat innecessària, intentant buscar la solució simple al problema per tal d'evitar perdre el temps.
- Don't Repeat Yourself(DRY). És comú que al afegir codi s'utilitzi una part ja hagi sigut escrita en una altre part degut a que hi han funcionalitats comunes i comportaments que es repeteixen. Aquests ser útils en diferents elements del codi, és per aquest motiu que és molt recomanable aquesta pràctica, que permet escriure codi més ràpidament, molt menys rígid i fràgil respecte als canvis, i que proporcionarà una lectura més intuïtiva i en menys extensió.
- Test-Driven Development(TDD). La base són els jocs de proves, on primer es plantegen unes proves específiques en les que es busca que l'aplicació falli o passi, i a partir d'aquestes proves es genera el codi necessari per complir les situacions desitjades. El que s'intenta buscar és poder provar l'aplicació per tal que sempre que es faci un canvi es pugui comprovar que compleix amb els resultats esperats.

5.6.5 Patró MVC(Model-Vista Controlador)

Symfony es basa en el patró de disseny conegut com a arquitectura MVC de 3 capes:

- El model representa la informació amb la que treballa i funciona la aplicació, és lo que es coneix també com a lògica de negoci.
- La vista mostra el model en una pàgina web on l'usuari pot interactuar amb ella.
- El controlador s'encarrega de respondre a les interaccions de l'usuari i invoca canvis en el model o la vista segons convingui.

La arquitectura MVC separa la lògica de negoci (model) i la presentació(vista), fent que entre aquests no s'hagi d'accedir a l'altre, com a intermediari està la capa del controlador que és la que permet la comunicació. Aquest patró de disseny està provat que aplicant-lo correctament es guanya en termes de canviabilitat, re-usabilitat, portabilitat i d'obtenir un sistema que sigui provable. L'abstracció de

l'aplicació en diversos nivells o capes ens permet realitzar canvis que no es propaguen a la resta de capes, per exemple ens facilita poder tenir una gestió de dades, o de la vista independent entre ells, també té algun inconvenient l'ús d'aquest patró ja que pot afectar al rendiment en general i a donar una feina extra per tal d'establir una organització i seguir les pautes que marca l'ús del patró.

Separació de la presentació

Per tal de saber si la capa de la vista està ben determinada i es lo suficientment independent, s'ha de tenir en compte el codi PHP que s'executa dins d'ella. El que s'hauria de trobar és l'estructuració del que serà mostrat a l'usuari, i no s'hauria de executar codi que impliqui accés a la capa del del model, el més comú seria trobar elements típics com echo(per mostrar la informació que ha de ser visualitzada), condicionals i altres estructures de control per processar conjunts de dades. Una convenció dins la capa de la vista és que el codi PHP no sigui l'encarregat de mostrar etiquetes HTML, una bona pràctica dins la vista és que els blocs de codi que s'hagin d'executar en PHP siguin mínims, a poder ser només d'una sentència.

Separació de la manipulació de les dades

El fet de tenir agrupat aquesta part del codi de l'aplicació i ser independent de la resta ens dona la possibilitat de poder tenir accés des d'una altre part de l'aplicació i així evitar la repetició i redundància del codi. Així quan vulguem realitzar algun canvi en la forma en la que es realitzen les consultes a la base de dades, o canviar el mateix motor la resta de l'aplicació no es veurà afectada la resta de l'aplicació, ja que seran funcionalitats encapsulades que en l'exterior es seguiran utilitzant de la mateixa forma i mentre els resultats que es retornin siguin els mateixos que s'esperen no es notaran els canvis. El que canviarà serà l'execució que podrà seguir un curs completament diferent sense haver de demanar cap permís. Un altre aspecte positiu és que d'aquesta forma la part del controlador es torna molt més fàcil d'entendre i senzilla, ja que les tasques que es podran observar només s'encarregaran de cridar a les funcions del model per tal d'obtenir les dades necessàries, i realitzar les gestions necessàries per passar-les a la vista.

En alguns casos el controlador també haurà de tractar i dur a terme les gestions necessàries i executar els mecanismes necessaris que estiguin relacionats amb les interaccions de l'usuari com pot ser la autenticació, peticions i d'altres. Per tal que es tingui la independència entre les capes, la del controlador també s'haurà

d'encarregar de rebre i donar la informació que requereixi la capa del model per executar les seves tasques, d'aquesta manera mai hi haurà la necessitat de que s'hagi d'haver d'accedir des del model a informació relacionada amb la vista.

El patró MVC especifica la divisió del codi de l'aplicació en tres capes, model, vista i controlador segons la naturalesa del codi. Però hi han altres patrons que especifiquen com subdividir cadascuna de les capes per oferir altres nivells d'abstracció sota certs aspectes. Symfony subdivideix cada una de les capes en diversos elements per donar una major flexibilitat.

Divisió de la capa del model

La capa de model pot ser dividida en una capa d'accés a dades i una altre d'abstracció de base de dades. D'aquesta manera s'obtenen crides que no depenen de la base de dades, és a dir, les consultes i modificacions que s'hagin de realitzar des de la capa, no estaran lligades específicament a una base dades concreta, i serà en la capa d'abstracció de base de dades on segons la configuració que tinguem variarà el codi que s'hagi d'executar. Pot semblar que el que s'aconsegueix és un major nivell de complexitat i de generació de codi, però s'ha de veure la capa d'abstracció com una capa que contindrà crides específiques a operacions a executar dins la base de dades, la resta de codi no es veurà afectat, i en el moment que es vulgui canviar de tecnologia utilitzada per l'emmagatzematge només es veurà afectada pel canvi part de codi molt específica, minimitzant molt el codi que haurà de canviar.

Divisió de la capa de la vista

La capa de la vista també es pot dividir en més components. És molt comú dins d'una pàgina web trobar-nos elements típics que formen l'estructura d'aquesta com poden ser les capçaleres, l'estil de la vista, els elements de navegació o el peu de pàgina. Una part dels elements de la vista poden ser fixos i només produir-se canvis en alguna de les parts de l'aplicació, per aquest motiu es produeix aquesta separació en el disseny(layout) i la plantilla(template).

El disseny és la part comuna que es manté a tota l'aplicació, tot i que com la gra majoria dels elements de dins de Symfony, és un comportament que es pot alterar, i fins arribar a tenir diversos dissenys depenent del context. La plantilla per una altre banda és la part que mostra les dades disponibles a través del controlador i que

s'afegeixen dins del disseny per acabar de generar la vista completa. Un tercer component serà necessari que és el mecanisme encarregat de unificar els dos components anteriors i generar la vista pertinent, a aquest element que ho gestionarà se li diu vista.

Divisió de la capa del controlador

En la part del controlador també es veu dividida en diferents elements. La major part de les funcionalitats de l'aplicació es veuen relacionades amb el controlador, ja que a diferencia de la vista que està associada a aspectes molt concrets, la tasca del controlador és controlar i gestionar la comunicació entre les altres capes i dur a terme les accions necessàries pel funcionament de l'aplicació. Moltes d'aquestes tasques són comunes a tots els controladors de l'aplicació i és per això que es divideix aquesta part en el controlador frontal de l'aplicació que s'encarrega de tasques com rebre les peticions i manegar-les, aspectes de la seguretat o carregar la configuració de l'aplicació.

Implementació MVC de Symfony

Aplicant els patrons comentats al framework de Symfony la estructura que ens queda per cada pàgina de la aplicació estarà organitzada i dividida d'aquesta manera:

- Capa del Model
    - Abstracció base de dades
    - Accés a dades
- Capa de la Vista
    - Vista
    - Template
    - Layout
- Capa del controlador
    - Controlador frontal
    - Acció

En total es poden comptar set components per una sola pàgina. Això però, no vol dir que per cada una de les pàgines que es creï hagi de tenir un d'aquests components independents, molts elements seran compartits amb altres pàgines. El controlador frontal i la vista són elements comuns a totes les accions de les aplicacions, es pot

tenir una configuració amb diversos controladors preparats per ser utilitzats, però només es necessita un, i és un component que es generat automàticament per Symfony. Les classes del model de dades també són generades automàticament basats en el model de dades que s'hagi definit al projecte. Aquesta és una tasca de la que s'encarrega l'ORM que s'utilitza, aquest conté classes esquelet i funcions per la generació del codi automàticament. I no només això, segons les funcionalitats que tingui el ORM si troba camps amb tipus de dades especials com dates, o claus foranes, els processarà i afegirà mètodes per consultar i manipular el contingut. En el cas d'haver de canviar el motor de base de dades tampoc seria motiu per haver de modificar codi, ja que els propis ORM tenen components que s'encarreguen d'ajustar-se al funcionament dels motors que estiguin suportats i així poder re-generar el codi sense haver de tocar-lo.

Classes del nucli de Symfony

La implementació MVC de Symfony utilitza un conjunt de classes amb els que m'he hagut de familiaritzar:

- sfController. És la classe del controlador. S'encarrega de descodificar les peticions i de redirigir a les accions oportunes.
- sfRequest. És la classe que s'utilitza per guardar tots els elements relacionats amb les peticions (paràmetres, cookies, headers,...)
- sfResponse. És la classe que conté les capçaleres de la resposta i el contingut d'aquesta. Aquest és l'objecte que més tard serà codificat i enviat com a resposta HTML a l'usuari.
- sfContext. El context de l'aplicació, guarda referències a tots els objectes del nucli de Symfony i a la configuració, és accessible des de qualsevol lloc de l'aplicació.

5.6.6 Organització del codi

Symfony organitza el codi en una estructura de projecte i col·loca els fitxers dins d'una estructura d'arbre. En un projecte de Symfony es tenen un conjunt de serveis i operacions sota un mateix domini que comparteixen el mateix model de dades. És per això que dins de l'estructura del projecte les operacions estaran agrupades per aplicacions. Les aplicacions normalment s'executaran independentment l'una de

l'altre. Un exemple pot ser la pròpia aplicació i tenir en una altra l'entorn propi per la gestió de les dades de l'aplicació principal (conegut com a backend).

Cadascuna de les aplicacions és un conjunt de mòduls, aquests mòduls solen representar una pàgina o un conjunt d'elles en que les funcions que s'executen estan mol relacionades. Dins de cada mòdul hi ha una divisió en accions, que són les funcionalitats que es podran cridar i executar i que es troben disponibles dins de un mòdul. En una situació general una acció es tradueix en una pàgina, tot i que una mateixa pàgina pot estar traduïda en diverses accions.

Estructura de fitxers en arbre

En la majoria de projectes ens trobarem els mateixos tipus de continguts:
- Base de dades
- Fitxers estàtics: HTML, Javascript, fulles d'estil, imatges, applets
- Fitxers pujats
- Classes i llibreries PHP
- Llibreries externes
- Logs
- Fitxers de configuració

Symfony ha tingut en compte tots aquests tipus de continguts que es poden trobar dins d'un projecte per crear la seva estructura de fitxers, però també té en compte els patrons en els que es basa Symfony i la relació entre aplicacions-mòduls-accions. Aquesta estructura és també creada automàticament quan s'inicialitza un projecte/aplicació/mòdul utilitzant les eines de Symfony a traves de la línia de comandes. L'estructura es pot reorganitzar i personalitzar a les necessitats de cada projecte a través dels fitxers de configuració.

Els directoris que es troben a l'arrel d'un projecte de Symfony per defecte són:
- apps. Conté un directori per cada aplicació que hi hagi dins del projecte.
- cache. Conté la configuració de l'aplicació en cache, i si s'activa la configuració també una versió de les accions i els templates del projecte. Aquest mecanisme s'utilitza per accelerar la resposta a les peticions web, i cada aplicació té el seu propi subdirectori.
- config. Conté la configuració general del projecte.

- data. Conté els fitxers de dades del projecte, fitxers SQL per crear bases de dades o carregar un conjunt de dades en elles, o també les bases de dades en SQLite.
- doc. Conté la documentació del projecte.
- lib. Conté les classes del projecte i llibreries externes, aquí es guarda el codi que es comú a totes les aplicacions. Dins del subdirectori model/ es troben els models dels objectes del projecte.
- Log. Conté els logs generats per les aplicacions del projecte. Està destinada per poder emmagatzemar tota classe de logs que estiguin relacionats amb les aplicacions, encara que no siguin els originals de Symfony. Symfony crea un log per aplicació i entorn d'execució.
- plugins. Conté els plugins instal·lats dins del projecte.
- test. Conté els jocs de proves escrits en PHP i compatibles el framework de testeig. Symfony afegeix automàticament alguns esquelets i proves bàsiques durant el procés de configuració del projecte.
- web. La carpeta arrel pel servidor web. Són els únics fitxers que seran accessibles des d'Internet.

Estructura d'arbre de l'aplicació

Per cada aplicació es troba un subdirectori amb el nom de l'aplicació concreta. L'estructura de directoris de totes les aplicacions és la mateixa:

- config. Conté el conjunt de fitxers de configuració en YAML amb la majoria dels aspectes de configuració de l'aplicació, apart dels paràmetres per defecte del framework, que es poden sobreescriure si es vol dins d'aquests fitxers.
- i18n. Conté els fitxers utilitzats per la internacionalització de la aplicació, es pot prescindir del directori si es prefereix utilitzar una base de dades per internacionalitzar.
- lib. Conté les classes i llibreries que són específiques de l'aplicació.
- modules. Conté els mòduls que proporcionen les funcionalitats de l'aplicació.
- templates. Conté els templates globals de l'aplicació. Aquí és on es troben la part del layout que comparteixen els mòduls d'una aplicació.

Les classes que es trobin dins del directori de llibreries de cada aplicació s'ha de tenir en compte que només seran accessibles per la aplicació en la que estan ubicades, és la diferència que trobem entre aquestes i la llibreria de tot el projecte. A l'hora de decidir com dividir aplicacions dins del projecte també s'haurà de tenir

en compte que no es podrà realitzar l'adreçament internament des de una aplicació a una altre, l'enllaç haurà d'estar en format absolut. N'hi han solucions que tracten aquest problema, totes tenen els seus aspectes negatius, alt component de programació per crear la funcionalitat (un nou Helper), haver de carregar tot el context de l'altre aplicació per poder obtenir la ruta, deshabilitar la cache,...

Estructura d'arbre dels mòduls

Per cada mòdul d'una aplicació es troba un subdirectori amb el nom del mòdul concret dins del directori modules de l'aplicació en la que està. L'estructura de directoris de tots els mòduls és la mateixa:

- actions/. Symfony per defecte genera només una classe anomenada actions.class.php en la qual es poden afegir totes les accions del mòdul.
- config/ . Els fitxers de configuració específics pel mòdul.
- templates/ . Conté les plantilles del mòdul corresponent (o no, com la majoria de comportaments a symfony es poden alterar, es veurà en el apartat del model de la vista) a l'acció amb el mateix nom.

Estructura d'arbre de Web

En aquesta estructura n'hi han poques restriccions. Aquí estan ubicats els fitxers ubicats que seran accessibles directament a través del servidor web. En el cas de que no es volgués mantenir aquesta estructura o que el servidor no la suportés es podria modificar. En el cas del projecte al no tenir cap mena de restricció i representar una estructura prou ordenada i còmode, s'ha mantingut l'estructura original, ja que dins de cada subdirectori es poden organitzar els fitxers com es vulgui sense cap haver de realitzar cap configuració addicional.

- css/. Conté els fitxers de les fulles d'estil.
- images/. Conté els fitxers d'imatges.
- js/. Conté els scripts de Javascript.
- uploads/. És el directori que es pot utilitzar per emmagatzemar tots els fitxers que siguin pujats pels usuaris, sense cap discriminació sobre el tipus de fitxer que es pugi.

5.6.7 Altres elements comuns

Aquests són elements o tècniques que s'utilitzen constantment o molt sovint per tal d'agilitzar el desenvolupament o el funcionament de les aplicacions dins del projecte i és convenient descriure per entendre certs components de Symfony.

Parameter holder (classe sfParameterHolder)

Moltes de les classes de Symfony contenen aquest element, que proporciona l'habilitat per estendre els objectes de les classes, aquesta funcionalitat també es pot afegir a les classes que es creen dins del projecte. El parameter holder que contenen s'encarreguen de manegar les dades, independentment del tipus que es vulguin afegir als objectes, i això sense que a la definició de les pròpies classes estiguin especificats els paràmetres. En general, quan es vol ampliar una classe el que es fa és redefinir-la o fer ús d'herència segons el cas, però si el que es vol és només poder emmagatzemar més dades sense haver d'afectar al comportament de la classe contenidora aquest manegador és la millor solució. Proporcionen les funcions pertinents per poder guardar/llegir les dades dins dels objectes còmodament, i en alguns casos la sintaxi arriba a ser molt simple i descriptiva, altres opcions que ens podem trobar és la utilització de valors per defecte a l'hora de no trobar l'atribut que s'intenta obtenir, o la utilització d'espais de noms:

Constants

Dins de Symfony no es troba cap constant definida per mètodes comuns de PHP ja que un cop s'han definit no es poden modificar, i tot i estar parlant de constants, podria ser informació típica de configuració definida per l'aplicació, però des de que estem parlant d'aplicacions web, que han de ser mantenibles, canviables i no molt rígides, aquesta elecció podria anar en contra dels requeriments. En el seu lloc utilitza un objecte de la classe sfConfig on es guardaran aquestes “constants”. Està proveïda amb els mètodes necessaris per poder accedir i llegir/modificar les dades en ell des de qualsevol lloc.

Autocàrrega de classes

Per utilitzar una classe concreta dins de l'aplicació es necessita incloure la seva definició abans de poder-ho fer. En un terreny en el que es poden acumular una gran quantitat de classes i la ubicació no és sempre la mateixa (recordem que les

llibreries poden estar localitzades per projecte, aplicació, mòdul o plugins) pot ser complicat i pesat la tasca de mantenir dins de l'aplicació aquestes inclusions. Per estalviar temps i feina Symfony proveeix d'una funció anomenada spl_autoload_register() que s'encarrega de cercar dins de tots els directoris de llibreries del projecte per fer les inclusions quan siguin necessàries.

Un error molt comú que es pot trobar al desenvolupar és acabar de crear noves classes i intentar provar el seu funcionament i obtenir un error perquè la classe no ha sigut trobada. Aquest error es dóna perquè el sistema d'auto-càrrega de Symfony escaneja els directoris durant la primera petició i emmagatzema aquest llistat de classes i fitxers en forma de vector associatiu dins d'un fitxer en memòria cau perquè les següents enquestes no hagin de reescanejar els directoris. Per aquest motiu quan la cache estigui activada i s'afegeixi o es mogui alguna classe s'ha de netejar.

5.6.8 Procés de creació d'una pàgina

Com s'ha explicat, Symfony agrupa les pàgines en mòduls. Llavors abans de poder crear una pàgina s'haurà d'haver creat un mòdul que inicialment tindrà una estructura de codi per omplir. A través de la línia de comandes es pot automatitzar aquest procés de creació de mòduls amb:
“symfony generate:module nom_aplicació nom_mòdul”.

Aquesta acció resultarà en la creació dels directoris actions/, templates/, dins de test/functional/nom_aplicació un fitxer per les proves del mòdul, i el fitxer d'accions del mòdul amb una plantilla per una acció anomenada index. Per cada mòdul, per defecte es crea una acció index, composada pel mètode executeIndex dins de les accions i la plantilla indexSuccess.php, aquesta és una de les convencions que pren Symfony per nombrar els elements, les accions amb el format executeXxxxx i les plantilles amb xxxxSuccess.php o xxxxxError.php depenent de com acaba l'execució de l'acció.

Aquesta plantilla per defecte ve buida, i l'acció executa una redirecció a la pàgina de felicitació per defecte del mòdul. També trobarem certes línies de comentaris, per symfony és també important la bona documentació del codi i per aquest motiu prepara cada classe per ser compatible amb l'eina de documentació phpDocumentor.

Per afegir una pàgina serà necessari tenir una acció on es trobarà la seva lògica, i la plantilla per la part de la presentació. Encara que la pàgina no hagi de tenir lògica l'acció sempre haurà d'existir, encara que sigui buida. El nom del mètode de l'acció, que ve després de “execute” ha de tenir la primera lletra en majúscules, s'ha de tenir compte amb les majúscules, ja que Symfony és discriminant en aquest aspecte. El format doncs de la URL serà:

http://host/aplicació.php/modul/accio

En cas d'estar accedint a un mòdul/acció que no es trobés es mostraria un error 404. Un cop que l'acció s'executa, espera una plantilla per renderitzar localitzada al directori *templates/* del mòdul, amb el mateix nom de l'acció i seguit de l'estat de l'execució de l'acció. Dins del template només s'espera codi per renderitzar, per això des de Symfony es recomana la utilització de la sintaxi alternativa de PHP, per tal de mantenir el codi el més intel·ligible possible i només poder utilitzar dins del codi estructures de control que són les que ofereixen aquesta sintaxis alternativa i evitar la col·locació de codi complex.

La tasca de les accions es executar la part de codi de càlcul, obtenció d'informació, i preparació de les dades necessàries per mostrar a les plantilles. Per poder accedir a la informació que les accions preparen només s'ha de assignar a variables dins l'espai de noms global d'aquesta manera : *$this->nomVariable*, així un cop dins de la plantilla es podran accedir les variables utilitzant-les pel seu nom només com qualsevol altre variable.

Tot i que els noms de les accions s'utilitzen per formar la URL de les pàgines, sota aquestes URLs hi ha un complet sistema d'adreçament que és el que defineix com és la estructura d'aquestes URLs. Això vol dir que donada una URL que utilitzem en una part del nostre codi, si es canvia la configuració de l'adreçament és molt probable que aquest enllaç deixi de funcionar. Per prevenir aquests errors es poden utilitzar unes funcions proporcionades pels Helpers associats a funcionalitats de URLs. La diferencia radicarà en que utilitzant les funcions rebran els paràmetres que li indicaran quina acció es vol executar i es crearà la URL de l'acció a partir de la configuració actual del sistema d'adreçament. Aquests Helpers no són més que funcions en PHP que estan dissenyades per ser utilitzades dins els templates per ajudar a crear codi HTML més ràpid i molt més flexible, ja que al ser codi PHP executat cada cop en el cas de que el sistema canvi el format de la URL

correspondrà al de la nova configuració. Aquests Helpers que retornen codi HTML també tenen la opció de rebre els atributs típics que poden esperar els elements HTML dins d'arrays associatius o strings com a paràmetres de les funcions.

A l'hora d'obtenir la informació ja sigui per POST o GET, es podrà accedir i obtenir mitjançant la funció del parameter holder de l'objecte petició sfRequest. A aquest objecte es pot accedir ja que és el paràmetre que reben les accions. En el cas de que es vulguin obtenir els paràmetres des de la plantilla també és possible fer-ho a través del objecte $sf_params amb el qual es pot accedir a tots els paràmetres de la petició actual. No només s'ofereix accedir al paràmetre, també hi han les funcions per consultar si la petició conté aquest paràmetre o no, i en el cas dels Getters, com s'ha comentat anteriorment suporten paràmetres per establir valors per defecte en el cas de no trobar el paràmetre en qüestió.

5.6.9 El sistema de configuració

Per tal de ser el més senzill d'utilitzar possible, Symfony utilitzar una configuració per defecte que hauria de respondre a la majoria de les necessitats més comunes de les aplicacions webs sense haver de modificar res. Aquesta configuració pot ser definida a diversos nivell: projecte, aplicació, mòdul. També pot ser definida segons l'entorn en el que s'estigui treballant: desenvolupament, test, producció i qualsevol altre que es vulgui definir. Per tal de cobrir les necessitats de qualsevol projecte ofereix un sistema en el que es poden modificar tots aquests aspectes i d'altres, com la manera en la que l'aplicació i el framework es comporten i es relacionen. En la majoria dels casos aquests canvis realitzats no suposaran afegir o canviar codi que s'executi en l'aplicació.

El que busca Symfony amb el seu sistema de configuració és:

- Potència: Que qualsevol dels aspectes que puguin ser manegats amb fitxers de configuració ho siguin a través de fitxers de configuració.
- Senzillesa: Molts aspectes de la configuració no apareixeran en les aplicacions ja que en poques ocasions es necessitarà canviar-los.
- Facilitat: Els fitxers de configuració són fàcils de llegir, de modificar i crear per qualsevol usuari.
- Llibertat: El llenguatge de configuració per defecte es YAML, però es pot fer amb fitxers INI o XML.

– Rapidesa: L'aplicació no serà l'encarregada de processar aquests fitxers, ja que el propi sistema de configuració serà qui analitzarà els fitxers i a partir d'ells generarà el codi PHP necessari.

YAML (YAML Ain't Markup Language)

Es tracta d'un llenguatge per serialitzar dades(codificar-les per tal de poder emmagatzemar-les) dissenyat per ser amigable i ser fàcil d'utilitzar amb els llenguatges de programació moderns per tasques comunes. YAML està composat per caràcters Unicode que poden ser impresos, alguns formen part de la sintaxis que es proveeix per poder estructurar les dades, i la resta conforme les dades mateixes. És un llenguatge molt net ja que la quantitat de caràcters que s'han d'introduir per estructurar la informació és molt baixa i s'aconsegueix que les dades que es representen estiguin estructurades d'una forma natural i entenedora. La senzillesa en la representació d'aquest llenguatge es conforma utilitzant tres tipus bàsics de dades: mapatges (hashes/diccionaris), seqüències(llistes/arrays) i escalars (strings/números). Però la definició de dades no està limitada només a aquests tres tipus, ja que s'ofereix una forma senzilla de definir tipus de dades més complexes a partir d'aquests.

La majoria de llenguatges suporten l'ús de YAML per la serialització de dades, però YAML sobresurt en els llenguatges que estan construïts fonamentalment al voltant d'aquests tres tipus, i aquí s'inclou com no Javascript i PHP. Tot i que el potencial que té per emmagatzemar dades és molt gran, YAML va ser creat per treballar amb àmbits com la configuració d'arxius, arxius de logs, missatges entre processos, compartir dades entre llenguatges, persistència d'objectes, i depuració d'estructures de dades complexes. El que es busca es representar les dades d'una forma simple i així poder simplificar la programació.

La primera especificació de YAML va apareix a l'any 2001, i la seva sintaxis prové del format de l'Email (RFC 0882). En certs aspectes i funcionalitats pot ser semblant a d'altres llenguatges per estructurar dades. En comparació amb JSON (*Javascript Object Notation*), un altre llenguatge per estructurar dades de manera àgil per tal d'intercanviar-les fàcilment, YAML és molt més llegible, ja que JSON sacrifica aquest aspecte per tal de poder generar i analitzar i extreure la informació amb un cost i una complexitat més baixa. En comparació amb l'XML tot i que aquest es pot fer servir en els casos en que s'utilitza YAML, aquest va ser dissenyat per complir altres

objectius, i per tant l'XML és un llenguatge molt més estructurat i imposa unes limitacions que no es troben a YAML, aquesta és la raó per la que Symfony utilitza YAML, ja que es considera un llenguatge en el que es pot reflectir les dades de configuració i modificar sense haver de tenir un alt nivell nivell de coneixement del propi llenguatge.

Convencions i regles

L'estructura de les dades en YAML es forma amb indentació, les indentacions han de ser amb espais, mai amb tabulacions, ja que sinó els analitzadors fallaran. En el cas de que s'utilitzin strings com a valors que continguin espais o caràcters especials els valors s'hauran d'introduir entre cometes simples, també serà així en el cas de que un valor es vulgui explicitar com a string, com pot passar amb true, off, i d'altres valors que es poden pendre per booleans, els quals tenen un ventall ampli a l'hora de ser representats, on, true, +, yes. Per definir strings llargs en múltiples línies es poden utilitzar els marcadors “<” i “|”, però també s'haurà d'afegir la indentació.

Els arrays venen definits per l'ús de “[]”, i els seus elements separats per comes. Hi ha una sintaxi alternativa en la que s'utilitzen guions i un element en cada línia, sense oblidar mai la indentació. Per mapejar elements en un vector associatiu s'utilitza {} amb els parells de clau i nom separats per dos punts. Els comentaris comencen amb “#”, i es poden afegir els espais que siguin necessaris per fer els fitxers més llegibles. En alguns fitxers de Symfony es possible trobar atributs de configuració comentats, aquests al estar comentats no es tenen en compte i resideixen allà per avisar l'autor de paràmetres de configuració del propi framework que es poden sobreescriure descomentant-los i modificant el seu valor. Una altre de les convencions de Symfony és utilitzar categories per agrupar llargues llistes de paràmetres, es defineixen utilitzant “.” seguit del nom i sota elles s'indenten els elements, així s'aconsegueix una millor llegibilitat, i el comportament no es veu afectat, ja que es segueix accedint a tots els valors de la mateixa forma com si no estiguessin les categories.

Fitxers de configuració

La configuració està repartida en fitxers i agrupada pels aspectes als que es refereixen. Els fitxers de configuració poden contenir dos tipus d'elements, definicions de paràmetres, o bé configuracions. Molts d'aquests paràmetres poden

ser sobreescrits en diferents nivells de les aplicacions, i uns altres són específics de certs nivells.

La configuració per defecte del projecte es troba als fitxers dins del directori nom_projecte/config/:

- ProjecteConfiguration.class.php : Aquest és el primer fitxer que s'inclou per qualsevol petició o comanda dins el projecte, per defecte conté el camí fins les llibreries del framework que s'utilitza, per canviar la versió utilitzada del framework s'ha de modificar aquest paràmetre. Altres configuracions es poden realitzar dins d'aquest fitxer.
- Databases.yml: En aquest fitxer estan definides les connexions amb les bases de dades que s'utilitzen dins del projecte amb els paràmetres necessaris per accedir a elles.
- Properties.ini: Conté paràmetres utilitzats per accions de la línia de comandes, en el nostre cas es poden trobar les configuracions que fem servir per sincronitzar el projecte entre els servidors web i els servidors locals del desenvolupador.
- rsync_exclude.txt: En aquest s'especifiquen els fitxers que són exclosos en el procés de sincronització entre els servidors.
- Schema.yml i propel.ini: Dins de schema.yml es troba la representació del model de dades del projecte. Propel.ini es crea automàticament per Propel, i conté les dades necessàries perquè Symfony i Propel puguin treballar amb les seves llibreries i amb el model de dades del projecte.

Es pot veure com hi han alguns fitxers de configuració en YAML i d'altres no, el motiu és que en aquells casos són fitxers que s'utilitzaran per components externs o per línia de comandes, abans que es pugui carreguar l'analitzador YAML del framework i processar els fitxers per ser utilitzats.

Configuració de l'aplicació

La major part de la configuració està localitzada dins de la configuració que fa referència a l'aplicació. La trobarem definida en diferents parts, dins del controlador frontal en el directori web/, dins del subdirectori config/ de cada aplicació, en el subdirectori i18n/ de cada aplicació i dins dels fitxers del framework.

Configuració del controlador frontal

La primera configuració que es carrega de l'aplicació es troba dins del controlador frontal, ja que és el primer script que s'executa quan arriba una petició al servidor. Cada aplicació té el seu controlador frontal el qual inclou la configuració del projecte, i carrega la configuració de l'aplicació amb el nom i l'entorn en el que s'està executant.

La classe de la configuració de l'aplicació que es carrega dins del controlador frontal té disponibles alguns mètodes que es poden utilitzar per comprovar la configuració durant la seva execució:

- $configuration->getRootDir(): Retorna el directori arrel del projecte.
- $configuration->getApplication(): Retorna el nom de l'aplicació que s'està executant.
- $configuration->getEnvironment(): Retorna el nom de l'entorn en el que s'està executant l'aplicació, a l'hora, l'entorn en que s'estigui determinarà quina configuració s'està utilitzant.
- $configuration->isDebug(): Retorna si s'està en mode de depuració, el mode de depuració també afecta a la configuració de l'aplicació i per tant a la seva execució en molts aspectes.
- $configuration->getActive(): Retorna una instància de la configuració actual.
- $configuration->getSymfonyLibDir(): Retorna el directori on estan les llibreries de la versió de Symfony que s'està utilitzant en el projecte.
- $configuration->getModelDir(): Retorna el directori de les llibreries del model del projecte.

En el cas que es vulguin modificar el més recomanable serà utilitzar una altre controlador frontal, i amb l'ajuda de les funcions existents a la mateixa classe vista per accedir a la configuració, també es permet de la mateixa forma utilitzar les funcions anàlogues per modificar els seus valors, d'aquesta manera un cop obtinguda la configuració dins d'un controlador frontal es pot utilitzar una configuració diferent per la aplicació sense afectar a la resta d'aplicacions o fins i tot només afectar a un entorn de desenvolupant en el que es vulguin fer proves.

Configuració principal de l'aplicació

La major part de la configuració de cada aplicació es troba dins del subdirectori config/ de cada aplicació:

- app.yml: Dins d'aquest fitxer hi ha la configuració específica de l'aplicació, es poden afegir variables globals o constants de l'aplicació que no s'hagin de emmagatzemar dins de la base da dades. El fitxer es crea per defecte buit.
- FrontendConfiguration.class.php: Aquesta classe s'encarrega de iniciar l'aplicació ja que carrega i inicialitza la configuració bàsica perquè es pugui iniciar, com la estructura de directoris de l'aplicació. La classe hereta directament de sfApplicationConfiguration.
- Factories.yml: En el seu interior es poden definir classes pròpies que s'encarreguin de les vistes, peticions, sessions,... en comptes de les que proporciona Symfony.
- Filters.yml: En aquest fitxer es defineixen quins filtres s'han d'executar durant cada petició que arriba al servidor.
- Routing.yml: Aquí es troben definides les regles d'adreçament que s'encarreguen de relacionar les regles amb les URLs de les peticions i fer les transformacions pertinents. Per defecte existeixen unes regles predefinides.
- Settings.yml: En aquest fitxer es troba la majoria d'aspectes a configurar dins d'una aplicació. Conté opcions que fan referència a la cache, la internacionalització, accions, mòduls, loging, de tot en general.
- View.yml: En aquest fitxer es configura l'estructura de la vista que s'utilitzarà dins de l'aplicació en general, a no ser que es sobreescrigui el comportament en algun mòdul.

Configuració de la internacionalització

Per els aspectes d'internacionalització de les aplicacions n'hi han dos fitxers que n'afectaran al seu funcionament, aquesta funcionalitat però s'activa dins del fitxer de configuració settings.yml de l'aplicació.

- factories.yml: Aquí es pot definir una classe pròpia per ocupar-se de la internacionalització, i d'altres aspectes generals com el llenguatge utilitzat i la cache per les traduccions,...
- Subdirectori i18n/: És on es troben els diferents diccionaris utilitzats per les aplicacions per traduir a cada una de les llengües que estiguin suportades per l'aplicació.

Configuració addicional de l'aplicació:

Com ja s'ha comentat, Symfony conté unes configuracions dins d'uns fitxers que es troben al directori de configuració de les seves llibreries del nucli que no apareixen

dins dels directoris de configuració de les aplicacions del projecte. Es tracta de configuracions que per defecte s'apliquen a tots els projectes de symfony, però també es poden sobreescriure.

Per fer-ho només s'ha de crear un fitxer amb el mateix nom que el seu homòleg dins del subdirectori de configuració de l'aplicació i agregar les configuracions que es vulguin modificar. La configuració que es defineixi dins de l'aplicació sempre té més preferència que les definides per Symfony. Els fitxers que es poden trobar al directori config/ de Symfony són:

- autoload.yml: En aquest fitxer es troba la configuració de la funcionalitat de auto-càrrega.
- core_compile.yml: Conté un llistat de les classes que s'han d'incloure per processar una petició.
- bootsrap_complie.yml: Conté un llistat de les classes que s'han d'incloure en la inicialització de l'aplicació.
- config_handlers.yml: En aquest fitxer es defineixen les classes que s'encarreguen de processar els fitxers de configuració.

Configuració del mòdul

Per defecte un mòdul no té configuració, però se li pot afegir creant un subdirectori config/ en el que es poden definir diversos aspectes que només afectaran a les accions i plantilles del mòdul. Aquests són els fitxers que es poden afegir i per defecte seran processats pel sistema de configuració:

- generator.yml: Per mòduls generats automàticament a partir de taules de la base de dades, aquí es podran definir els aspectes relacionats amb com es visualitzaran els camps i les funcionalitats que podrà realitzar l'usuari amb els elements i els llistats.
- Module.yml: Realitza la mateixa tasca que app.yml per l'aplicació, on es podran definir paràmetres i aspectes relacionats amb les accions del mòdul.
- Security.yml: Definicions de regles i restriccions d'accés a les accions del mòdul.
- View.yml: Configuració que sobreescriu les configuracions fetes en el view.yml.

El sistema de configuració de Symfony és un dels grans punts forts del framework, a l'hora d'utilitzar-lo et dona la possibilitat de modificar i adaptar la gran majoria de les característiques i el seu funcionament. Quan s'hagi de realitzar alguna

modificació ens trobem amb un sistema molt modular amb molt fàcil accés ja que la majoria de fitxers estan ubicats dins dels directoris de les aplicacions, mòduls i projecte, en directoris de configuració i agrupen els paràmetres i configuracions segons la tasque o l'aspecte de l'aplicació al que estiguin relacionats.

Els fitxers que s'han de modificar estableixen que per realitzar un canvi només s'hagi de canviar un valor i no haver de modificar la part del codi que s'executa, aïllant així els canvis realitzats del codi que de les aplicacions. I en el cas de que no es necessiti realitzar cap canvi no s'haurà de fer cap canvi degut a la configuració per defecte que s'estableix tant per part del framework, com per les aplicacions i mòduls creats dins d'un projecte.

Entorns

Symfony ofereix la utilització d'entorns com a una altre alternativa per tenir diferents configuracions per una mateixa aplicació. Dins d'un entorn el codi que s'executarà serà el mateix que en un altre entorn per la mateixa aplicació, el que canviarà serà la configuració que es carregarà. Per defecte es proveeixen de tres entorns: prod(producció), dev(desenvolupament), test(proves) i d'altres que es defineixin personalment.

Aquesta funcionalitat és present per facilitar les tasques que s'han de realitzar dins de cada entorn, ja que no s'esperarà utilitzar les mateixes configuracions en diferents entorns. En el cas d'un entorn de desenvolupament, el que interessarà és que els canvis es vegin reflectits immediatament o que es guardi tota la informació relacionada amb el funcionament i execució de l'aplicació per tal de facilitar la tasca als desenvolupadors. Però en canvi, en un entorn de producció amb una aplicació acabada el que interessarà serà obtenir el millor rendiment d'aquesta, o utilitzar un conjunt de dades diferent per l'aplicació que no siguin els utilitzats durant el desenvolupament.

Per tal de fer servir un altre entorn dins de l'aplicació, s'haurà de definir un altre controlador frontal, com ja hem vist abans, al executar el controlador frontal per carregar la configuració s'especifica l'aplicació i l'entorn a utilitzar, aquella és la forma en la que es crida un altre entorn, tenint un controlador que defineixi el paràmetre diferent.

Configuració en cascada

Dins de Symfony per tal d'aconseguir una plataforma completament configurable, però sense anar en detriment de la dificultat que pot comportar haver de configurar tots els aspectes d'un projecte, es fa servir un sistema de configuració en cascada, que permet la configuració a diferents nivells de molts aspectes. Aquestes configuracions es poden fer de dues maneres, segons l'entorn, o segons el nivell jeràrquic de l'aplicació. Per això el sistema s'encarrega d'assignar preferències entre aquests nivells. Així doncs obtenim un sistema de prioritats, de major a menor en aquest ordre:

1. Configuració del mòdul
2. Configuració de l'aplicació
3. Configuració del Projecte
4. Configuració per un entorn específic
5. Configuracions per tots els entorns
6. Configuració per defecte de symfony

Configuració de la caché

El sistema de configuració ofereix moltes funcionalitats, i s'encarrega de molts aspectes de l'aplicació, els quals s'han d'analitzar, crear les instàncies, i carregar-les per tal de poder executar el codi el comportament del qual depèn d'aquestes característiques descrites. Les aplicacions web estaran rebent contínuament peticions al servidor web, i si per cada petició s'hagués de realitzar aquest procés suposaria una ralentització innecessària. Si tenim en compte que un cop iniciada l'aplicació, la configuració en un entorn estable no hauria de variar, per aquests casos Symfony proporciona un mecanisme per tenir en cache la configuració i no haver de repetir constantment aquest procés. El funcionament d'aquest sistema es basa en que les classes encarregades de carregar les configuracions pertinents transformen els fitxers de configuració en un conjunt de fragments de codi PHP perquè es puguin executar ràpidament.

Aquesta és una de les diferències per defecte entre els entorns de producció i els entorns de desenvolupament. En els entorns de producció el framework fa ús en la configuració que s'ha generat i hi ha a la cache, però en els entorns de desenvolupament es compare les dades de modificació dels fitxers de configuració amb els de la cache, i re-processa els que troba que s'ha produït algun canvi des de la última petició. Per això en els casos d'haver-se produït canvis en l'entorn de

producció és necessari haver de netejar la caché, ja sigui amb la comanda proveïda de Symfony, o en el cas de no ser accessible per línia de comandes eliminant el contingut del directori cache/ del projecte directament.

Accedint a la configuració desde el codi

Generalment els fitxers de configuració seran transformats en codi PHP i la majoria seran utilitzats per les classes que defineixen el funcionament del framework sense una intervenció del propi codi de les aplicacions. Però en alguns casos es voldrà accedir a les configuracions establertes per redefinir el comportament o senzillament obtenir dades de l'aplicació emmagatzemades dins dels fitxers de configuració. Per accedir a aquests elements es facilita una classe anomenada sfConfig.

Aquesta classe ens ofereix els mètodes necessaris per poder accedir, i per tant consultar i/o modificar els paràmetres que tinguem des de qualsevol part del nostre codi. Per accedir a un paràmetre dels fitxers s'utilitza una nomenclatura específica com a paràmetre de la funció de sfConfig. El nom s'obté a partir de la concatenació en aquest ordre del nom del fitxer on es troba seguit els noms de les claus que contenen el paràmetre, i per últim el nom del paràmetre, totes les parts es concatenen amb un "_" i en minúscules.

Dins dels fitxers app.yml per defecte es troben unes claus que no s'utilitzen a l'hora de generar el nom per accedir als paràmetres des de sfConfig, aquests paràmetres fan referència als entorns, en el cas per defecte es troba "all:", i a les dades que hi hagin sota aquesta clau s'accediran des de tots els entorns. Així doncs, es poden definir paràmetres específics per cada un dels entorns i sfConfig s'encarrega d'aquest aspecte a l'hora de decidir quin valor agafar, en cas contrari trencaria amb la filosofia de que el codi sigui el mateix per entorns i només es vegi reflectit en les configuracions i paràmetres.

Un dels problemes que pot ocasionar inicialment treballar amb configuracions i paràmetres de fitxers en YAML és l'accés jeràrquic a paràmetres. Al intentar obtenir paràmetres en els quals la seva indentació és més profunda que tres nivells d'elements el que retorna es nul, per aquesta limitació convé estructurar les dades adequadament tenint en compte la informació que es voldrà accedir directament i la que no, i les agrupacions que es facin per tal fer els accessos el més directe possible. En el cas que es vulgui poder accedir directament a un array de

paràmetres és convenient utilitzar les categories, però tenint en compte que un cop s'afegeixen aquestes categories els accessos es faran recollint les dades que hi hagin sota el seu abast i retornant-les en forma d'array. Una de les altres possibilitats que ens ofereixen els fitxers de configuració i YAML a l'hora d'assignar valors, és utilitzar valors d'altres paràmetres els quals fent servir constants es farà un tractament especial, i en comptes de guardar un valor repetit quan s'accedeix a ell es retorna el valor de l'original, per utilitzar les constants s'utilitza una nomenclatura en forma de "%NOM_PARÀMETRE%".

Quan s'hagi d'accedir Symfony identifica la constant i retorna sfConfig::get("nom_parametre") sempre que el fitxer al que s'hagi d'accedir ja hagi sigut processat abans. Una altra forma de assignar valors és utilitzant codi PHP, aquest codi quan s'hagin de processar els fitxers serà executat i es guardarà el valor que retorni, per tant aquest codi només s'executarà en aquell precís moment, i fins que no es torni a processar de nou els fitxers de configuració no canviaran a no ser que es modifiqui durant l'execució de l'aplicació.

Una forma alternativa d'accedir a la informació continguda dins dels fitxers és utilitzant la classe sfYaml que processa els fitxers que se li demanen, aquesta classe proporciona un mètode per obtenir i retornar el contingut en forma de array associatiu de fitxers sencers, és útil quan tenim fitxers dedicats a dades que al utilitzar-les es processen tots els seus camps i no aïlladament, o fent accessos aleatoris.

5.6.10 Dins del la capa del controlador

Un cop dins de la capa del controlador de symfony que fa de nexe d'unió entre la capa de presentació i la del model, es poden diferenciar els següents elements en els que s'ha fet la divisió dels components:

- El controlador frontal, és el component per on s'inicia l'execució de l'aplicació, rep la petició i carrega la configuració inicial i crida a l'execució.
- Accions de l'aplicació. Reben la informació de les peticions, la tracten i s'encarreguen de preparar la informació necessària per mostrar-la a la vista.
- La petició, resposta, sessió, són representats per objectes dins de Symfony amb els que s'ofereix les funcionalitats relacionades amb el tractament de la informació que contenen o hauran de contenir per facilitar les tasques a realitzar dins la capa.

El controlador frontal

Totes les peticions són rebudes per algun controlador frontal de l'aplicació, sense aquests no es poden iniciar les aplicacions, per això una manera de bloquejar l'accés a les aplicacions és deixar el controlador inaccessible o comentant el codi que conté perquè no realitzi la seva funció. Quan el controlador rep una petició s'inicia el mecanisme d'adreçament per relacionar la URL rebuda amb un mòdul i una acció de l'aplicació. Les accions que realitza el controlador frontal per tal de realitzar la petició rebuda són:

1. Càrrega de la classe configuració del projecte i les llibreries del nucli.
2. Crear la configuració de l'aplicació i el context de l'aplicació.
3. Carrega i inicialització de les classes del nucli del nucli.
4. Carrega la configuració.
5. Descodificació de la URL rebuda per determinar mòdul/acció i els seus paràmetres.
6. En cas d'error redirigir a 404.
7. Activació i execució de filtres.
8. Execució d'acció i renderització de la vista.
9. Execució de filtres segona passada.
10. Mostrar el resultat.

El controlador frontal per defecte

El controlador frontal que Symfony crea per defecte al crear una nova aplicació dins del projecte és el fitxer PHP index.php de la carpeta web/ del projecte. Els controladors venen a ser un fitxer en PHP com l'anterior que contenen un fragment de codi en el que es carrega la configuració inicial de l'aplicació, i acte seguit li passa l'execució de les següents parts al controlador del nucli de Symfony sfController.

Accions

Les accions probablement seran el nucli de les aplicacions que desenvolupem, ja que contindran la majoria del codi que s'haurà d'executar per definir el comportament de l'aplicació. L'execució d'una acció ve determinada per la URL de la petició que arriba, en la qual el nom de l'acció ve inclòs. Aquest s'identifica entre les accions ja que serà el mateix que hi ha dins de les classes de accions dels

mòduls, en forma de “executeXxxxx” dins d'un conjunt d'accions nombrat com a “xxxxxActions” que identificaran les accions d'un mòdul de l'aplicació, el nom del qual serà el mateix que el que vingui especificat dins de la URL de la petició. Quan s'analitza el contingut d'un dels fitxers d'accions trobarem que la classe hereta de sfActions. Aquesta és la classe del nucli que defineix el comportament genèric per totes les accions de Symfony.

Aquests fitxers contenen un conjunt de funcions amb la nomenclatura abans comentada, però per tal de poder mantenir una organització més estructurada en el cas que fos necessari, s'ofereix una sintaxis alternativa, en la que en comptes de tenir totes les accions en una mateixa classe, es guarda una acció per fitxer. Totes aquests segueixen heretant de sfActions com abans, i el nom de la classe passa a ser “xxxxxAction” i la única funció que hi haurà serà l'acció i el seu nom només serà “execute”.

Accés a la informació

Des les accions es pot accedir directament a tota la informació relacionada amb la capa del controlador directament, i això és possible gràcies a que les accions hereten de sfComponent, que és una de les classes nucli de Symfony en la que es troben tots els components dins la capa del controlador.

El context és una classe singleton que ja hem vist com es creava al rebre la petició i iniciar l'aplicació pel controlador frontal. Aquesta classe conté tota la informació relacionada amb l'execució de l'aplicació en procés, i per tant si es pot accedir a ell es pot accedir a qualsevol element. Dins de la capa del controlador es proveeix de diferents funcions per accedir a aquest i d'altres objectes de classes de Symfony:

Acabament de les accions

Com s'ha pogut veure, la renderització de la vista, en concret de la plantilla que s'utilitzarà es pot veure afectat per l'execució de l'acció. Dins del seu cos es poden utilitzar unes constants pròpies de la classe sfView per activar aquest comportament explícitament. N'hi han dos valors possibles sfView::SUCCESS i sfView::ERROR, que estan relacionats amb la nomenclatura de les plantilles que es mostraran en acabar les accions. En el cas que la crida s'hagi executat correctament, es mostrarà la plantilla xxxxxSuccess.php, aquestes plantilles s'utilitzaran per defecte en el cas de que no s'especifiqui quin ha estat el resultat

per la visualització de la plantilla. També es procedirà a mostrar aquestes plantilles si les accions que es criden són buides, ja que no comportarà cap error.

En el cas de que es vulgui comunicar que l'execució ha comportat en l'error d'alguns dels aspectes de la crida, s'utilitzarà la constant sfView::ERROR, la qual farà que es cridi a la plantilla amb nom xxxxxxError.php. En el cas de que es vulgui mostrar una plantilla amb un format del nom personalitzat es podrà fer retornant com a valor directament el nom que complementarà al nom de l'acció per identificar la plantilla: return 'Foobar'; -> nomacciófoobar.php

Una altre opció present és pel cas en que es realitzin accions que no necessiten de la renderització de cap element de la vista, com pot ser una crida AJAX al servidor o l'execució de tasques a la base de dades. D'aquesta manera s'evitarà la capa de la vista i no s'executarà cap dels seus passos. Per aquests casos es faciliten funcions en el cas de que es vulguin enviar missatges a través de les respostes HTTP, o en el cas que només s'hagin de retornar les capçaleres com en el cas de X-JSON s'utilitzarà la constant de sfView::HEADER_ONLY.

I per últim, també es pot determinar durant l'execució de les accions quins seran els elements que s'utilitzaran dins la capa de la vista, podent obligar a utilitzar les plantilles i layouts diferents als que estiguin definits en la configuració de la vista, algunes de les funcions que permeten alterar aquests comportaments són setTemplate() i setLayout(), per establir un template i layout respectivament diferent.

Canviant a altres accions

En alguns casos l'acció que s'inicia per la crida a una acció no té perquè acabar en aquella mateixa, sinó que pot estar composada per diverses crides en la que cada una es realitza una tasca o de diverses, que complementant-se composen una funcionalitat de l'aplicació. Aquest esquema també es pot utilitzar per definir àlies per les accions, d'aquesta forma es poden realitzar crides a accions amb el nom que es vulgui veure reflectit a la URL, però un cop s'arribi a l'acció, aquesta pot redirigir a una altre acció sense que l'usuari s'adoni no alterant la URL original.

Symfony ofereix dues solucions per realitzar aquestes redireccions:

- forward(): Redirigeix a una altre acció. Aquesta funció no envia cap missatge de redirecció cap al navegador de l'usuari ja que és una redirecció interna que realitza Symfony.
- redirect(): Redirigeix a una altre URL. A diferència de forward(), en aquesta es fa una redirecció que envia el missatge al navegador de l'usuari i per tant es veurà reflectida. S'utilitza sempre que s'han rebut les dades d'un formulari per POST, així s'evita que si es refresca la pàgina un cop enviat no es tornin a enviar les dades al servidor, i en cas de navegar cap enrere es retorna al formulari en comptes de preguntar si es volen tornar a enviar les dades.
- forward404(): Redirigeix a una pàgina de l'acció de “pàgina no trobada” de Symfony. Aquesta acció es troba per defecte dins del directori controller/default/ en el que es troben la acció i el template del mòdul als que es redirigeix amb la funció. Com en altres casos, es pot sobreescriure aquest comportament afegint un nou mòdul dins de la nostra aplicació amb la seva acció i template corresponent amb els mateixos noms, o bé, es pot modificar dins dels paràmetres de settings.yml assignant una acció o mòdul específics per utilitzar en aquests casos.

Per complementar a aquestes funcions, també s'ha tingut en compte que en la majoria de casos les redireccions es produeixen quan es compleixen o no certes condicions. Per plantejar aquestes situacions s'ofereixen uns mètodes semblants als anteriors que introdueixen aquestes condicions, aquests poden ser forwardIf(), redirectIf(), forwardUnless(), en els que només se'ls hi ha de passar uns paràmetres addicionals que representen les condicions a complir per executar o no els redireccionaments.

El funcionament de la redirecció dins de Symfony funciona de tal manera que un cop s'executa una d'aquestes funcions es llença una excepció sfStopException, que serà recollida per Symfony i ignorada. D'aquesta manera un cop sigui llegida no es seguirà executant més codi, per aquest motiu s'ha de tenir cura de no deixar codi després de les funcions ja que no serà executat a no ser que la redirecció estigui condicionada.

En el cas de realitzar un forward404 i les seves variants condicionals, es llençarà una sfError404Excepction per controlar la resposta 404 que es retorna. Coneixent aquest comportament podem fer servir dins de qualsevol part del nostre codi aquestes excepcions per enviar missatges d'aquest tipus.

Repetint codi per múltiples accions d'un mòdul

Una de les possibilitats que ens ofereixen l'ús de les convencions dels noms donats a les diverses funcions dins la capa del controlador, és el poder afegir-ne d'altres que no seran reconeguts directament per l'aplicació a l'hora d'executar-se normalment i que per tant no afectaran o interferiran en el funcionament correcte dels mecanismes d'execució de la capa. D'aquesta manera es poden afegir mètodes dins de les accions només fent que els seus noms no comencin per execute, aquesta tècnica facilita no tenir accions de grans proporcions, obtenint un codi molt més llegible, mantenible i reutilitzable dins dels fitxers de les accions.

Una altre de les tècniques que s'utilitza dins de les accions és la refactorització de codi que s'executa. Alguns fragments de codi es poden extreure i inserir dins del mateix fitxer d'accions d'un mòdul com a funcions anomenades preExecute() i postExecute(). Com es pot intuir la primera executarà el codi abans de l'execució d'alguna de les accions del mòdul, i la segona executarà el codi després d'haver-les executat.

Utilitzant aquests mètodes es poden estalviar parts com comprovacions de certs paràmetres o condicions que s'han de complir amb la petició al servidor per tal de rebutjar-la o permetre la continuació i execució, o bé la comprovació de la integritat del missatge després d'executar el codi de l'acció abans de mostrar el resultat a l'usuari en la vista.

Utilització de l'objecte de la petició

Com ja s'ha comentat, a la acció li arriba com a paràmetre un objecte de la classe sfWebRequest de Symfony, aquest objecte conté tota la informació relacionada amb la petició que el servidor rep de l'usuari. Una de les aplicacions que hem vist que té és la consulta dels paràmetres que arriben amb ella fent ús del parameter holder que conté. Però ni molt menys està limitada a recollir paràmetres, es poden consultar tots els aspectes de la petició actual:

- Informació de la petició: si es POST o GET, SSL, cookies, capçaleres.
- Paràmetres de la petició ja vist i comentat.
- Informació relacionada amb la URI: el host, script demanat, path de la aplicació,...
- Informació del navegador de l'usuari: llenguatges admesos, conjunt de caràcters,...

Amb l'existència d'aquesta classe deixa de ser necessari l'ús de les variables globals de PHP com $_SERVER o $_ENV, o la resta de variables que segons el servidor que s'utilitzi poden tenir un format diferent i trobar-se amb les diferencies al accedir als seus continguts. Amb Symfony no existeix aquest problema, ja que la classe s'encarrega de gestionar aquestes diferències i ofereix una estructura idèntica per tots els casos, a més d'una forma més elegant i més fàcil de llegir.

Gestió de la sessió

Symfony també proporciona un objecte per gestionar la sessió de l'usuari, la classe sfUser, per accedir dins de les accions s'utilitza el mètode getUser(). Una de les funcionalitats que ofereix és com d'altres objectes de les classes del nucli, el parameter holder, però en aquest cas la seva vida útil no s'acaba amb cada petició, ja que l'objecte i el seu contingut es manté mentre existeixi la sessió de l'usuari. És una altra forma de passar informació de l'usuari cap a la vista, ja que des de les plantilles també serà possible accedir a la sessió de l'usuari de la petició rebuda.

S'ha de vigilar amb l'ús d'aquestes classes, ja que no està exempt de problemes, tot i tenir un parameter holder que pot emmagatzemar tota mena de dades, objectes fins i tot, entre les peticions que es realitzen al servidor pels mateixos usuaris, els objectes de les sessions es serialitzen per guardar-se, i al arribar una nova petició i deserialitzar-los s'obtindrà un error ja que aquest procés es realitza abans que l'autocàrrega de les classes. Aquests incidents però es poden solucionar realitzant aquesta serialització explícitament amb la funció de PHP serialize() abans de guardar l'objecte dins la classe perquè Symfony no ho hagi de portar a terme. Un hagi arribat una altra petició i es vulgui accedir als objectes, manualment també s'haurà de utilitzar unserialize() per realitzar el procés invers.

Altres incidències que poden ocórrer utilitzant els objectes de les sessions per emmagatzemar dades en ells és la neteja de la informació que desem dins l'objecte. Rl parameter holder que conté permet amb els seus mètodes eliminar paràmetres seleccionats o fins i tot netejar tots els que s'hagin introduït. Però aquestes funcions no eximeixen d'haver de controlar i executar-les quan es cregui convenient, per aquest motiu és possible que es quedin elements guardats dins la sessió fins que no expiri.

Per evitar-ho es proporciona l'ús d'uns atributs anomenats “flash”, aquests atributs es poden guardar i utilitzar en una petició, i després de la següent seran eliminats

de la sessió. Aquestes dades es comú utilitzar-les a l'hora de marcar errors o avisos després de realitzar certes tasques que s'haurà de tenir en compte que s'han produït, però que un cop tancat el cicle no tornaran ser necessaris. En el cas de que no s'arribessin a utilitzar el seu comportament no es veuria afectat i desapareixerien de la mateixa forma.

La configuració de la gestió de les sessions també es pot realitzar, des de modificar el nom de les cookies que s'utilitzen en l'usuari que per defecte es diuen 'symfony', com modificar l'inici de les sessions que també és automàtic per defecte. Aquests canvis es poden realitzar modificant el fitxer de configuració factories.yml, configurant aspectes com els anteriors, la duració de la persistència de les sessions, si es guarden en fitxers o a la base de dades del projecte, o canviar les classes que s'encarreguen de la gestió per unes pròpies.

Seguretat en les aplicacions

Symfony també ofereix un conjunt de funcionalitats per poder assegurar les aplicacions i poder dur a terme un control tant de l'accés que es permet als usuaris a les aplicacions, com a les parts per les que es composa. Aquestes funcionalitats es composen de dos processos:

- Control d'autenticació dels usuaris per accedir a aplicacions segures.
- Utilització de credencials assignats als usuaris de les aplicacions que determinaran quins privilegis tenen dins de l'aplicació i les accions que poden dur a terme amb ells.

Com d'altres aspectes a Symfony la configuració de les aplicacions es pot fer o bé dins del fitxer security.yml de la carpeta config/ de l'aplicació per establir un comportament genèric en l'aplicació, o bé es poden definir dins d'un fitxer security.yml ubicat dins dels subdirectoris de configuració dels mòduls que sobreescriuran a la configuració de seguretat definida per l'aplicació. Per defecte aquests fitxers venen configurats perquè les accions estiguin obertes, el mateix passaria si no es trobés el fitxer de seguretat.

Les possibilitats a l'hora de definir credencials són múltiples, dins dels fitxers de configuració, no només es poden definir un sol tipus de credencials per accedir a les aplicacions, la sintaxis de YAML permet utilitzar ORs i ANDs per tal de poder fer combinacions de credencials per accedir a les accions i aplicacions.

Utilitzant les regles de seguretat que Symfony ofereix es poden definir diversos esquemes o situacions per l'accés a les aplicacions, que es poden utilitzar a part de per obtenir un control l'accés, també per definir com poden interactuar els usuaris segons el grup de regles en el que encaixin. Amb aquests esquemes es pot oferir una escalabilitat de permisos encara més gran que només utilitzant credencials o autenticació. Situacions possibles:

- L'usuari esta autentificat i té credencials assignats
- L'usuari està identificat però no té credencials
- L'usuari no està autentificat

Per poder realitzar la gestió de les regles que es defineixen amb els fitxers de configuració es proveeix d'un conjunt de funcions que es poden executar sobre l'objecte sfUser per tal de canviar l'estat de l'usuari en termes de seguretat. Les comprovacions que hi han afegides dins dels fitxers de configuració s'executen automàticament pels filtres del sistema de seguretat, però a l'hora de realitzar un login o voler canviar els privilegis d'un usuari s'haurà de realitzar dins del codi de les nostres aplicacions, ja sigui des d'un backend dissenyat per la gestió o fent servir mòduls i accions específics. Els mètodes disponibles que per si sols es descriuen són:

- isAuthenticated() / setAuthenticated():
- hasCredential() /setCredential()/removeCredential()/clearCredentials()

Filtres

El procés que s'executa per realitzar les comprovacions de seguretat que s'ha comentat era un filtre que s'ha de passar per tal que es permeti l'execució de l'aplicació correctament quan es rep una petició. Aquest filtre realitza unes comprovacions, i depenent dels resultats la petició seguirà el seu curs natural, o bé es veurà alterada i redirigida a una acció diferent que el filtre ordeni.

Els filtres a Symfony, com tots els elements que composen el funcionament del framework, al ser orientat a objectes no són més que classes, un conjunt de classes que s'ha definit que hereten de sfFilter, classe del nucli i que s'executen quan s'ha de processar cada petició d'una aplicació determinada. El conjunt de filtres s'executa com una cadena, que a l'hora és el nom que s'utilitza internament per l'execució de tot el conjunt.

Quan una petició arriba s'executa el primer filtre, sfRenderingFilter que s'encarrega de fer la renderització de la vista, durant l'execució d'aquest filtre i sense que acabi, fa una crida al següent filtre de la cadena, el qual no estarà definit dins del codi del filtre que el crida, la cadena de filtres es continuarà executant fins que arribi a l'últim, sfExecutionFilter, que s'encarrega d'executar les accions i vistes associades a la crida. Un cop finalitzada l'execució es retorna al filtre que l'havia cridat, i així fins que es torna a arribar a sfRenderingFilter el qual acaba la cadena de filtres.

Per cada una de les classes dels filtres es troben uns elements molt importants i imprescindibles per l'execució i funcionament dels filtres. El primer és com estan definides, com ja hem dit les classes hereten de la classe filtre de Symfony, i contenen un mètode anomenat execute() el qual rep com a paràmetre la cadena de filtres que obligatòriament han de rebre tots els filtres en forma de variable $filterChain, paràmetre de la classe sfFilterChain.

Els objectes d'aquest classe contenen un array amb els filtres que s'han executar i un index per indicar el següent a executar, junt amb els mètodes necessaris per inicialitzar-los i dur a terme correctament la execució de la cadena. Durant l'execució del filtre, per continuar amb el següent es fa la crida execute() en aquesta variable i es passa al següent filtre de la cadena.

És aquest punt el que produeix una diferenciació en dos parts del codi que hi ha escrit dins de cada filtre. El codi que hi hagi abans d'aquesta crida serà executat abans de l'acció i la vista, i el codi que ve després s'executarà només quan s'hagi processat correctament l'acció i la vista. Al ser una cadena de filtres, un filtre només s'executarà si els que el precedeixen han sigut superats. Alternativament dins d'un filtre, es poden realitzar redireccions a altres accions quan es vulgui trencar la cadena de filtres, com pot passar en el cas del filtre de seguretat quan no es compleixen els requisits necessaris per renderitzar la petició rebuda.

La cadena de filtres que s'han d'executar no es defineixen dins del codi dels filtres, ja que afectaria greument a la canviabilitat i reutilització dels filtres. Els filtres que s'afegiran a la cadena de filtres per ser executats i el seu ordre, queden determinats en els fitxers filters.yml dels subdirectoris de configuració de les aplicacions. Per defecte Symfony executa els següents filtres:

- sfRenderingFilter: Ja comentat, s'encarrega de renderitzar la vista.
- sfBasicSecurityFilter: El filtre amb les funcionalitats de seguretat de Symfony.

- sfCacheFilter: Filtre que controla si la caché està activada i si la URI que es demana està cachejada per tal de no haver d'executar sfExecutionFilter, o realitzar el caching de la URI després de l'execució.
- sfCommonFilter: Filtre que només s'executa després de la renderització de la vista i afegeix els Javascript i fulles d'estil en el contingut de la resposta retornada a l'usuari.
- sfExecutionFilter: Com ja s'ha comentat, és l'últim filtre registrar i s'encarrega de l'execució de la acció i la vista.

Com s'ha pogut observar, els filtres estan definits per classes dins del fitxer filters.yml. Els seus paràmetres i opcions estan definides dins del fitxer filters.yml del nucli de Symfony, per aquest motiu es mostra "~" en l'interior del fitxer de l'aplicació. Per tant a l'hora d'afegir un nou filtre s'hauran de tenir en compte tots aquests aspectes, creant primer la classe que definirà el comportament i execució del filtre, que hauran de complir amb els requisits comentats anteriorment sobre el funcionament d'un filtre. També s'haurà de definir per tan dins del fitxer filters.yml la declaració no només del nom del filtre, també de la classe que el composa, i els paràmetres que es vulguin afegir.

Configuració del mòdul

De la mateixa manera que es tenen paràmetres de configuració dins de app.yml que fan referència a l'aplicació, també sabem que es poden tenir per mòduls. Dins d'aquest fitxer es poden sobreescriure paràmetres de configuració que hi són a l'aplicació, però també es poden configurar alguns aspectes propis a la execució del mòdul, com és la desactivació de totes les accions del mòdul, de poder-les executar només de manera interna, o substituir la classe que defineix la vista pel mòdul.

5.6.11 Dins la capa de la vista

La vista és responsable de mostrar el resultat que es produeix al realitzar una acció demanda. En Symfony es divideix la part de la vista en diferents parts per facilitar el seu disseny, aportant una estructura lògica en la que es poden diferenciar i distribuir les tasques que s'han de realitzar per visualitzar el contingut de les respostes. Els aspectes que es tenen més en compte són la facilitat a l'hora de dissenyar i realitzar una vista, ha de ser una tasca que no exigeixi uns alt grau de coneixements del framewok ni de PHP, ja que la finalitat es renderitzar, no executar

codi. Per tal de potenciar la reusabilitat, es té en compte que dins d'una aplicació o pàgina hi han elements que no canvien o que podrien no fer-ho, com capçaleres, menús o banners, i ha de ser possible no haver d'estar reescrivint per cada vista que s'hagi de generar de nou tots aquests elements. La configuració ha de ser senzilla i potent, per tal que afecti el mínim possible als elements de la vista tal que no s'hagin de refer al realitzar canvis.

Helpers

Els Helpers són funcions PHP que al executar-les retornaran codi HTML per ser mostrat en les vistes. La majoria de Helpers són funcions que s'encarreguen de mostrar un codi a partir d'uns paràmetres o opcions, però siguin els paràmetres o opcions que siguin el que es mostra té la mateixa finalitat, uns tractaran enllaços a rutes de l'aplicació, uns altres imatges, o Javascripts, o fulles d'estil, etc... Aquestes funcions faciliten la codificació de les plantilles permetent-nos estalviar temps a l'hora de dissenyar-les, i escrivint fragments de la millor manera possible, oferint la possibilitat de configurar els elements amb totes les opcions que admetrien de la forma original, obtenint com a result el mateix amb un bon rendiment i la millor accessibilitat, ja que al ser funcions PHP resultarà en un codi dinàmic entre les peticions i canvis que es puguin realitzar dins del projecte al generar cada cop el resultat.

Una de les diferencies amb les classes de Symfony que tenen els Helpers és que no s'autocarreguen. Això passa perquè es declaren com un conjunt de funcions i no una classe amb els seus mètodes com en els altres casos. Aquest conjunt de funcions es troben ubicats en fitxers PHP amb nom XxxxHelper.php. Per que es puguin utilitzar en la plantilla s'hauran d'incloure amb una crida a “use_helper(Xxxx)”, i en el cas que es necessiti carregar més d'un la funció accepta afegir tants paràmetres com Helpers es vulguin afegir. Pels motius anteriors l'ús d'aquests elements per generar codi HTML a les plantilles és molt comú i per defecte hi han alguns grups que són disponibles sense haver-los de declarar explícitament:

- Helper: El helper que s'utilitza per incloure altres helpers, conté la funció use_helper().
- Tag: Conté funcions que s'utilitzen per generar etiquetes HTML com tag(), content_tag(), _tag_options(), _parse_attributes(),...
- Url: S'utilitza per generar les URLs i els seus enllaços a aquestes, link_to(), mail_to(), url_for(),...

- Asset: Proporciona funcions per afegir elements a la capçalera HTML i crear enllaços a recursos de la carpeta web com imatges, CSSs, Javascripts.
- Partial: Proporciona funcions per afegir components en slots de les vistes, hi han funcions com: include_slot(),get_slot(), has_lot() que es veuran més endavant.
- Cache: Permet gestionar fragments de codi a la cache. Proporciona dues funcions, cache($name, $lifetime) i cache_save() per iniciar, aturar i guardar fragments amb el nom seleccionat.
- Form: Per introduir els elements típics d'un formulari: input_tag(), label_for(), textarea_tag(), select_tag(), options_for_select(),...

Els Helpers que es carreguen per defecte estan definits dins del fitxer de configuració settings.yml i es poden modificar, excepte per Helper, Tag, Url i Asset, ja que són utilitzats per la vista per generar els components, per aquest motiu no es poden ni modificar en els fitxers de configuració del framework. Com aquestes funcions són afegides per la capa de la vista, des de les accions no es podrà la funció use_helper, i en el cas que es vulgui utilitzar una de les funcions s'haurà de cridar a través de la classe de Symfony sfLoader, que proporciona dos mètodes, un per cercar i localitzar els directoris dels helpers d'un mòdul, i una altre per carregar Helpers.

Si els Helpers que hi han disponibles a Symfony no compleixen les nostres expectatives sempre es poden crear-ne de nous o personalitzar-los. L'únic que s'ha de tenir en compte a l'hora de crear-ne de nous són dos aspectes: el nom ha de ser de la forma xxxxHelper.php, i aquest fitxer ha d'estar dins d'un subdirectori helper/ de algun directori lib/ del projecte/aplicació/mòdul, que són els directoris on es cerquen Helpers per carregar-los. En el cas de que es vulguin modificar alguns de Symfony simplement introduint-los dins dels directoris de Helpers del nostre projecte es carregaran i sobreescriuran als originals. Totes les funcions hauran de ser re-definides o introduir còpies, ja que els fitxers originals no es carregaran i només es podrà accedir a les funcions que hi hagin dins del Helper redefinit.

Disseny de la pàgina

Dins dels continguts de les plantilles podem veure que no hi és present l'estructura lògica d'un document HTML. Falta descriure doncs una part del document que és el que defineix l'estructura on es troba el contingut que s'ha de visualitzar. Aquesta part del codi és el que a Symfony es denomina com 'layout', aquests fitxers

contenen parts de codi HTML que és comú a totes les pàgines de l'aplicació (o casi totes, veurem com aquest comportament es pot alterar fàcilment) i com en altres casos per evitar la repetició en cada plantilla s'extreu i es col·loca dins d'unes plantilles globals per l'aplicació. Aquesta estructura es basa en el patró de disseny estructural anomenat decorador, en el qual es busca agregar funcionalitats a un objecte mitjançant la associació de classes. En el cas que ens ocupa tenim el resultat d'una vista, en el qual tenim dos components com són la plantilla i el layout que hereten de sfComponent del nucli del framework que s'utilitza per definir components de les vistes. Per crear la renderització que es veurà com a pàgina de l'aplicació s'hereta el contingut del layout i s'associa la plantilla com es pot veure en el següent esquema:

Dins dels layouts no només s'inclou el contingut que correspon a la part del cos del document, sinó que també s'han d'afegir els elements necessaris a la capçalera. Per realitzar aquestes inclusions es faciliten de nou unes funcions de AssetHelper que com ja s'ha vist abans permeten afegir declaracions a les capçaleres dels documents HTML finals.

Accés a dades de symfony

Des de les plantilles es facilita l'accés a certa informació de l'execució de l'aplicació. Concretament es proporciona accés directe a alguns dels objectes clau del funcionament de les aplicació que són:

- $sf_context: Instancia del context.
- $sf_request: Petició rebuda pel servidor.
- $sf_params: Paràmetres de la petició.
- $sf_user: Sessió actual de l'usuari.

Els mètodes que hem vist que hi havien disponibles des de la capa de controlador es poden utilitzar també sobre els respectius objectes de la mateixa manera per obtenir les dades necessàries.

Separació de codi

De manera anàloga a la capa del controlador, deixant de banda el layout de les pàgines, es pot donar el cas que tinguem codi que es repeteixi durant diverses pàgines de les aplicacions, però no en totes, de manera que no es pot afegir directament dins d'una plantilla global. Per aquests casos s'ofereix noves formes per

organitzar el fragments de codi per tal d'evitar la repetició i poder realitzar canvis sense que la resta dels components es vegin afectats. Les solucions disponibles per refactoritzar el codi depenen de la naturalesa d'aquest:

- Parcials: Si el comportament és similar al d'una plantilla, en la que es vol mostrar contingut al que s'accedeix mitjançant variables i sense haver de generar ni executar cap codi extra.
- Components: Si el que es vol és poder accedir al model de dades i realitzar accions i/o procesar dades que precisi d'execució de codi en la capa del controlador i del model.
- Slots: Si el que es vol es delimitar una zona en la que s'haurà de mostrar fragments específics de l'aplicació.

Tots aquests fragments de codi que es poden incloure amb els diferents elements nombrats s'afegeixen dins de les plantilles i dels layouts fent ús d'un dels Helper abans descrit PartialHelper, a més és un dels Helpers que hi ha disponible des de qualsevol template per defecte.

Partials

Per fer ús d'un parcial s'utilitza la funció use_partial() del Helper, aquesta s'encarrega de buscar el parcial dins de tots els directoris de templates, ja sigui dins del mòdul, en un altre o en els de l'aplicació. Per poder utilitzar-lo ha de tenir la nomenclatura "_xxxx.php".

És tracta d'un fitxer PHP que al igual que les altres plantilles pot contenir codi HTML i codi PHP per executar, el seu contingut pot ser idèntic al d'una plantilla, la única diferència és que no serà específic d'una acció per defecte, es trobarà dins de plantilles i layouts on s'hagin inclòs. Al ser elements que s'inclouen dins de la capa de la vista, i no venen de l'execució d'una acció, per tal de poder accedir a paràmetres s'han de especificar quins podrà utilitzant passant-los com a paràmetres en les funcions del Helper.

En el cas de que es renderitzi directament des de una acció un component o un parcial amb renderPartial() o renderComponent() tampoc es podrà accedir directament a les variables, i s'haurà d'especificar de la mateixa forma les variables com a paràmetres dins d'un array.

Components

Els components són els únics elements d'entre els altres que es descriuen, que permet l'execució de codi de la mateixa forma que les accions. Hi han certes similituds entre components i accions per facilitar l'execució de la lògica desitjada, però també hi han diferencies suficientment substancials com perquè es puguin utilitzar dins d'una plantilla o altre element de la vista.

Els components s'inclouen amb la funció del Helper corresponent include_component(), en la que s'especifica dins de quin mòdul està i el nom del mètode del component a executar. Els components es troben ubicats en els mateixos directoris que les accions, dins dels mòduls, i fins i tot comparteixen la mateixa forma de designar els noms de les accions i dels fitxers. Addicionalment de la mateixa manera que es poden passar variables a un parcial, també es pot fer en un component.

Un cop trobat el fitxer del component desitjat, el qual hereta de la classe de symfony sfComponent i no sfActions, es comença a executar. Un cop acaba es passa a representar la part equivalent a la plantilla del component que s'ha de trobar dins d'un parcial i tenir el mateix nom que el mètode del component.

A diferència dels parcials com els anteriors, aquest si que té accés a les variables que es poden passar des del component de la mateixa manera que ho fa una plantilla amb la seva respectiva acció. La gran diferència entre les accions i els components radica en el funcionament subjacent dels mecanismes que els executen. Per un nou component afegit dins la vista no s'executarà la cadena de filtres que s'executa durant una acció, això permet una execució més ràpida. A més al ser un component i haver de formar part d'altres elements de la vista no es permet la seva execució individual, només pot aparèixer inclòs dins un altre, podent-se incloure dins d'un altre component també.

Slots

Els slots tenen una finalitat lleugerament diferent, també s'utilitzen per poder afegir continguts dins de les vistes. Els slots defineixen aquestes zones i no el contingut, el programador serà qui seleccionarà quan i amb que es fan les inclusions en cada situació que es vulgui plantejar. Els slots proporcionen una forma per gestionar unes

zones comunes dins les vistes, en les que poden anar diferents continguts, i fer que canviïn segons l'execució de l'aplicació.

Amb els slots pots tenir zones definides dins del layout amb uns continguts per defecte, com pot ser un menú especific de l'aplicació, i que a l'hora d'executar en un cert context de l'aplicació aquest menú no es mostri o sigui substituït per un altre. D'una altre manera s'hauria de definir en totes les vistes generades per l'aplicació quin element es mostra, generant codi redundant.

El funcionament d'aquest element es gestiona en dos fases amb les funcions del Helper, per una banda es defineix un slot utilitzant els delimitadors slot() i end_slot(). Un cop s'hagi definit un slot, des de la part de la vista que es vulgui controlar aquest slot es farà ús de les funcions del mateixHhelper: has_slot() i include_slot(), la primera es pot utilitzar per saber si en algun moment de la renderització de la petició de l'aplicació s'ha incorporat aquest slot o no, i l'altre funció s'utilitza per incloure un slot ja definit en alguna part del codi amb el mateix nom que s'ha rebut com a paràmetre.

Component slots

Aquesta és un component resultant de la combinació dels dos elements descrits anteriorment. Com en el cas dels slots es defineixen unes zones que segons el context s'ompliran amb continguts diferents. Fins aquí sembla un slot, però difereix en la forma en la que s'omplen aquestes continguts on entra en acció el component. La funcionalitat que s'obté d'aquesta fusió és la possibilitat d'executar diferents components segons el context de l'aplicació, el resultat dels quals sempre serà mostrat en la zona definida pel slot. La defnició d'un component_slot consta de 3 parts:

- La part de la vista que pot ser el layout, on s'utilitza include_component_slot('xxxx') per definir la zona on estarà localitzat el slot.
- Els fitxers de configuració view.yml, en els quals dins de l'apartat component es definira per cada component utilitzat el mòdul i el nom del component a executar.
- El component que s'executa, la seva definició és la mateixa que els altres components descrits anteriorment, amb el seu component.class.php i el seu partial corresponent.

Configuració de la vista

Dins de la capa de la vista a través dels elements anteriors es pot manipular tot el contingut que es presenta a través de codi HTML i PHP que s'executa per ser renderitzat en la vista. Però hi ha uns altres aspectes a tractar en aquesta capa com pot ser com poden ser inclusions d'elements predefinits de Javascript i fulles d'estil, o informació addicional que no es mostra directament a l'usuari però pot ser igualment utilitzada a la xarxa. Tots aquests aspectes es poden controlar a través dels fitxers de configuració de la capa de la vista. En cas de no ser possible, s'haurien d'utilitzar dins la capa del controlador l'objecte de la classe sfResponse per tal de modificar directament aquests paràmetres abans de ser retornat a l'usuari.

Tots els paràmetres a configurar dins la capa de la vista es troben dins dels fitxers view.yml que es poden definir a diferents nivells i segueixen la jerarquia de preferència entre els fitxers de configuració abans descrita. En cas de modificar un dels paràmetres dins de la capa del controlador a través de sfResponse, aquestes opcions tindrien preferència sobre les que hi haguessin definides als fitxers view.yml. Dins dels fitxers view.yml, els paràmetres es poden definir per dos casos, el cas general que afectarà per defecte a totes les vistes, i també es poden definir la configuració per cada vista utilitzant el nom complet de la plantilla com a clau per identificar-la.

El objecte sfResponse

Alguns aspectes i elements de la vista en quant a la resposta que es retornarà a l'usuari són inclosos dins la configuració de la vista, però poden estar sotmesos a certes condicions que es poden produir durant l'execució de les peticions. Els mateixos paràmetres que es poden configurar dins dels fitxers es poden veure sobreescrits dins del codi que es generi. D'aquesta manera es proporciona una flexibilitat encara major, permetent canviar els aspectes relacionats amb la vista que s'ha de mostrar, abans que s'arribi a la capa de la vista. La finalitat d'utilitzar els mètodes associats serà voler controlar certes situacions durant l'execució que no poden ser definides en els fitxers de configuració, amb sfResponse s'ofereix la possibilitat d'introduir el control d'aquestes condicions i modificar el comportament de la vista abans d'arriba a la seva execució i quan sigui es cregui convenient.

Paràmetres de configuració de la vista

Els paràmetres que es poden configurar dins dels fitxers i modificar a través de la classe sfRequest són diversos també, i a l'hora de modificar-los i sobreescriure'ls en diferents nivells s'han de tenir certs aspectes en compte.

- Els paràmetres que hagin sigut definits i tinguin un únic valor, al assignar un de nou amb més preferència serà eliminat i sobre-escrit, però pels paràmetres que tinguin múltiples valors definits se li aniran afegint els diferents valors que es vagin acumulant a través de les diferents configuracions trobades.
- Per evitar el comportament acumulatiu dels paràmetres de configuració que suporten diversos paràmetres, es pot utilitzar una nomenclatura que permetrà eliminar-ne alguns el caràcter “-” davant del nom de l'element perquè no aparegui, o eliminar tots utilitzant “*-”.
- Des de la capa del controlador, s'ofereix el mateix comportament per modificar els paràmetres oferint dues funcions diferents, una per substituir el valor, i una altre per afegir-los sense sobreescriure.
- Alguns dels elements afegits necessiten de les seves respectives funcions del Helper Asset per ser introduïts dins de les capçaleres del document final.
- Es facilita la inclusió dels fitxers com CSS i Javascripts, que no necessitaran de cap funció extra, automàticament són afegits a les capçaleres pel filtre sfCommomFilter ja comentat.

Output escaping (mecanisme d'escapament)

Un altre dels mecanismes de seguretat que ofereix Symfony en el desenvolupament d'aplicacions web aquest cop dins la capa de la vista, és el mecanisme d'escapament. Aquest mecanisme s'encarrega de processar les dades que s'introdueixen dins la capa de la vista i que al ser mostrades poden comportar en l'execució de codi maliciós o no esperat per part de l'aplicació. Els casos més comuns són els atacs cross-site scripting(XSS) i les injeccions SQL.

A través dels primers es poden executar scripts dins del client i afectar tant a l'usuari en cas de no ser el causant de l'atac i per tant accedir a dades que estiguin emmagatzemades dins les seves sessions d'usuari, o bé pot arribar a afectar a les pàgines i aplicacions, ja que si no estan correctament protegides por resultar en una injecció de codi en els documents HTML. Per aquest motiu és de vital importància

proporcionar protecció tant a l'aplicació, com als usuaris que es poden veure afectats. Per una altre banda estan les injeccions SQL que també són molt utilitzades. Aquests atacs intenten mitjançant sentencies SQL atacar la base de dades que hi hagi darrere de l'aplicació, ja sigui per robar dades o per modificar-les o eliminar-les.

Per configurar aquesta protecció de l'aplicació s'han de modificar els paràmetres dins del fitxer settings.yml de l'aplicació que apareixen com:

- escaping_strategy: Permet activar o desactivar el mecanisme.
- escaping_method: Determina com es tractaran les dades, els Helpers que seran utilitzats per tant i executaran el codi que processarà les dades a tractar:
    - ESC_RAW: Sense escapar les dades.
    - ESC_SPECIALCHARS: Utilitza la funció htmlspecialchars() que converteix els caracters que per HTML tenen un significat especial com poden ser “<”, o “&”, a la seva corresponent entitat HTML.
    - ESC_ENTITES: Utilitza htmlentities(), que a diferència de htmlspecialchars() converteix qualsevol caràcter que tingui una entitat HTML per ser representat, i utilitza la opció ENT_QUOTES que convertirà tant les cometes dobles com les simples.
    - ESC_JS: Escapa els valors que seran utilitzats com a cadenes de Javascript.
    - ESC_JS_NO_ENTITIES: Escapa els valors que seran utilitzats com a cadena de Javascript, però no li afegeix les entitats HTML corresponents

Un dels errors que pot produir l'ús d'aquestes tècniques és en el cas d'utilitzar arrays amb algunes funcions PHP com a paràmetres, les funcions de Symfony no tenen problemes a treballar amb arrays decorats, ja que elles mateixes proporcionen els mecanismes per tractar les dades, però n'hi han d'altres funcions que no funcionaran correctament i s'ha de tenir cura de no utilitzar directament sobre aquestes variables funcions les variables ja que no tenen en compte aquest comportament. La resta de dades també es veuran afectades per aquestes mesures de protecció, no solament variables simples i arrays, els objectes també seran

processats i les instàncies dels objectes de Symfony que també es poden fer servir dins la capa de la vista com sf_user, sf_request,... oferint una protecció a totes les dades que es poden utilitzar en l'aplicació.

5.6.12 Dins la capa del model de dades

El sistema que utilitza Symfony per controlar i accedir al seu model de dades es recolza en la capa d'abstracció que proveeix el sistema de mapeig objecte/relacional Propel. Actualment es pot utilitzar un altre ORM pel que hi ha força suport conegut com Doctrine, no obstant, Symfony des de la seva aparició ha treballat junt amb Propel, i és l'ORM que tenia un millor suport i integració en la versió 1.1, amb una gran quantitat de plugins desenvolupats per complementar-lo. En el projecte s'utilitza una versió antiquada de Propel, ja que cada versió de Symfony té suport per les versions amb les que apareix, per tal de poder utilitzar unes altres s'hauria de modificar les llibreries del nucli del framewok.

L'ús d'un ORM per abstreure la capa de dades de la base de la base de dades s'ha comentat que pot tenir molts aspectes positius com són la separació i encapsulació de codi encarregat únicament d'obtenir dades i per tant poder reutilitzar-lo dins la capa del controlador a través de totes les aplicacions i mòduls que sigui necessari.

Aquest ús evita la duplicació de codi i dóna una independència del motor de base dades que s'utilitza aportant a més funcionalitats i eines. Dins del desenvolupament de les aplicacions l'aspecte més important que proporciona l'ús d'aquestes tecnologies és l'aprofitament de la programació orientada a objectes junt amb la capacitat d'autocàrrega de Symfony ens permet la utilització de les classes que representen el model de dades en qualsevol moment. Al estar representades per objectes podem estendre el seu comportament afegint nous mètodes dins de les classes, o estenent les seves definicions de classe aprofitant l'herència de classes per complir amb les necessitats de l'aplicació i així no estar limitats a les funcions bàsiques per obtenir i modificar els valors del atributs membres de las taules relacionades a la base de dades.

Un dels elements que no s'utilitzen directament, sinó a través de Propel és Creole. És una capa d'abstracció per PHP que s'utilitza per accedir a la base de dades. S'encarrega d'abstreure les crides a funcions específiques de motors de bases de

dades per poder crear aplicacions portables, fent ús d'interfícies basades en la orientació a objectes. La API que s'utilitza està basada en la de JAVA JDBC. Un altre aspecte tecnològic de Propel és l'ús de PHP Data Objects, (PDO) que es tracta d'una extensió de PHP que generalment ve per defecte amb la seva instal·lació.

Creole defineix una interfície per poder accedir i treballar amb bases de dades independentment del motor utilitzat, de manera que junt amb PDO s'utilitzaran un conjunt de drivers específics pels motors de bases de dades que s'encarreguen d'implementar les funcionalitats específiques per les bases de dades. Per tant, combinant PDO i els seus drivers es pot crear codi destinat a comunicar-se amb la base de dades del model sense que en aquesta comunicació hi hagi aspectes específics del motor utilitzat més tard.

Les habilitats de PDO però, no s'estenen més enllà d'aquesta especificació, el seu funcionament no compta amb un tractament de de les funcionalitats de cada motor per optimitzar-les, per tant si un motor no suporta cert aspecte d'una consulta, encara que es facin servir les funcions implementades pel driver de PDO, aquesta no es realitzarà correctament, s'ha de tenir en compte doncs les capacitats dels motors que es vulguin utilitzar.

Per tal de definir les interfícies d'accés i que els drivers específics les implementin s'utilitza la herència de classes que ofereix PHP5, per això l'ús de PDO no està suportat per versions més antigues. Algunes de les funcions que es proporcionen són molt comunes com les consultes a les bases de dades, transaccions i tractament dels resultats que ofereixen la majoria de bases de dades actuals.

Esquema de la base de dades de symfony

Per tal de traslladar l'esquema definit a la base de dades al model de dades que s'utilitzarà dins de l'aplicació l'ORM necessita fer la traducció pertinent de cada taula a l'objecte corresponent i de cada una de les columnes als atributs de les classes. Aquest esquema es pot definir en diversos llenguatges, nosaltres hem utilitzat un fitxers escrit en YAML per definir cada un dels esquemes de les bases de dades que s'utilitzen.

Dins dels esquemes trobarem les representacions de les taules que hi han dins les bases de dades, esquemes que es defineixen per les connexions utilitzades i no per base dades concreta. Anteriorment ja s'ha parlat dels entorns en els quals s'ofereix

un conjunt de configuracions adaptades per la situació i tasca que es vulgui desenvolupar: desenvolupar, depurar, testejar,... L'ús de connexions es deu a que és molt probable que dins de cada entorn es vulguin utilitzar conjunts de dades diferents, i definint diferents connexions es poden utilitzar altres servidors, o bases de dades amb diferents conjunts de dades amb els mateixos esquemes. Aquest és el principal motiu per fer aquesta distinció, així es permet tenir només un esquema de base de dades que es podrà utilitzar amb diferents connexions. Amb la sintaxis de YAML es defineixen les taules i dins de cada una es defineixen les seves columnes podent afegir paràmetres per definir les relacions entre columnes i/o les restriccions que hi han definides a la base de dades o que es volen imposar pel model de dades de l'aplicació.

Classes del model

A partir de l'esquema comentat l'ORM pot generar el model d'objectes. Propel proporciona un conjunt d'eines a través de la línia de comandes de Symfony per automatitzar i realitzar aquestes tasques, per generar el model de dades que prèviament ha estat definit en el seu esquema corresponent a la carpeta config/ del projecte es generarà executant només:

“symfony propel:build-model”

Un cop generat el model de dades observarem que dins de lib/model/, el directori on s'emmagatzemen les classes dels models de dades utilitzades al projecte, es crea una estructura jeràrquica de classes que afecta a cada una de les classes. Per cada una de les taules que s'hagin definit en l'esquema obtindrem 4 classes:

– BaseXxxx.php i BaseXxxxPeer.php dins de lib/model/om
– Xxxx.php i XxxxPeer.php dins de lib/model

La ubicació com comentarem més endavant pot variar dins del directori lib/model degut a la utilització de paquets de models.

Les classes Base dins del subdirectori om/ són re-generades cada cop que es detecta un canvi a l'esquema al realitzar la crida a la comanda de generació del model. Aquestes classes contenen codi generat directament pel ORM i conté els mètodes relacionats amb l'accés i la modificació als objectes de la classe. Aquestes classes no seran utilitzades directament, i per aquest motiu el primer cop que es

genera el model o s'afegeix alguna taula nova a l'esquema es creen les altres dues classes objecte que hereten directament de les Base.

Dins d'aquestes classes objecte és on s'introduiran els mètodes propis que s'hagin d'afegir a la capa del model, o en el cas que els mètodes que hi hagin a les classes Base no satisfacin les nostres necessitats en tots els aspectes, es poden sobreescriure dins les classes objecte redefinint de nou la funció per afegir el codi que es cregui convenient i fent la crida a la funció original. Aquest mecanisme d'extensió també és possible gràcies a la herència que existeix entre les classes Base i les objecte. Aquestes classes seran també les que s'utilitzaran dins de l'aplicació per representar les dades del model, a través d'elles es crearan, modificaran i es cridaran els mètodes tant de les pròpies classes com de les implementades per l'ORM.

És important seguir aquest esquema, ja que en el moment que s'hagi de modificar l'esquema es reescriuran les classes Base i es perdran les funcions que no s'hagin afegit dins de les classes per personalitzar. També s'haurà de tenir en compte que al no modificar-se les classes objecte, si es realitza algun canvi que afecti alguna columna o restricció d'una classe ja existint i es tingui en compte en els mètodes afegits, s'haurà d'adaptar el codi manualment a les noves característiques.

De les quatre classes generades es poden diferenciar també en dos tipus, segons si són classes objecte o són Peer. Les classes objecte representen l'objecte en si mateix, i cada una de les instàncies de classe representen una fila (o possible candidata a fila) a la base de dades. Són les que proveiran mètodes per poder accedir a informació relacionada amb l'objecte com poden ser consultes d'altres objectes referenciats per claus foranes, o per accedir als seus atributs i/o canviar-los a través de mètodes getters i setters. Aquests mètodes ja estaran implementats per defecte dins de les classes Base que són les que proveeixen accés als atributs i relacions de les taules.

Els mètodes que s'hagin d'executar sobre una instància concreta de la classe i utilitzar, modificar o fer comprovacions sobre aquesta s'hauran de cridar a través d'una instancia de la classe. Dins d'aquestes classes també estan els mètodes per control del cicle de vida de l'objecte i la seva persistència a la base de dades, cridant al mètode save() podrem guardar o actualitzar el contingut de l'objecte dins de la taula on està representat, i amb delete() es pot eliminar un objecte emmagatzemat a la taula. Aquesta és la forma de modificar el contingut d'una

taula utilitzant instàncies d'objectes individuals, les classes Peer utilitzen uns altres mètodes per realitzar aquestes operacions sense haver de crear o obtenir les instàncies dels objectes.

Per les classes Peer a diferència de les classes objecte no estan relacionades ni representen una instància o fila concreta de la classe o taula. Dins les classes Base-Peer l'ORM implementa unes funcions estàtiques que permeten realitzar consultes i altres operacions a les taules que representen. Aquestes funcions retornen informació sobre el resultat de les operacions, o en cas de consultes conjunt d'objectes de les taules.

Per realitzar operacions a la base de dades a través de les classes Peer existeixen dues vies. En la primera molt més simple, es basa en la necessitat d'obtenir un determinat element, aquests mètodes s'executen utilitzant com a paràmetres les claus primàries de les taules, ja siguin simples o múltiples. En el cas de que sigui necessari fer una consulta més complexa i restrictiva s'utilitzaran els mètodes de les classes Peer junt amb uns objectes de la classe Criteria o Criterion.

Les classes Criteria i Criterion de Propel són la solució que proporciona l'ORM per tal de simplifcar l'ús de consultes SQL. Utilitzant-les com a objectes es poden construir consultes d'una complexitat considerable sense utilitzar sintaxis SQL. El funcionament de la classe Criteria es basa en mètodes per afegir clàusules i restriccions a les relacions entre atributs que es volen afegir a les consultes. Les restriccions que s'introdueixin dins de l'objecte poden ser de tot tipus de comparacions, i quan no sigui possible definir-les mitjançant operadors propis de la classe es poden fer servir d'altres per definir-ne manualment(fins i tot en SQL quan no sigui possible d'altre manera).

En aquests objectes Criteria també es pot fer ús de les relacions que hi hagin dins les taules afegint les clàusules per realitzar joins de tots tipus (left, right, inner) dins de les consultes que es volen executar.

Uns altres aspectes que permet personalitzar Criteria a les consultes amb mètodes de classe és l'addició d'ordenació en el resultat, o la selecció de les columnes que vulguem obtenir en cas que no vulguem rebre un conjunt d'objectes. En aquest últim cas s'haurà de processar la informació manualment, ja que Propel quan s'utilitza l'objecte Criteria per cridar consultes com User::doSelect($myCriteria) s'encarrega de realitzar la consulta i un cop rebut el resultat en forma d'un conjunt

de dades va creant els elements de la classe a retornar i omplint-los, totes aquestes operacions es realitzen amb funcions implementades i personalitzades per crear i omplir els objectes adequadament segons la definició de la classe i els atributs esperats.

Perquè es produeixi aquest mecanisme Propel ha de rebre conjunts de dades que facin referència a objectes complets, en cas contrari si s'intenta realitzar consultes quan s'intenti omplir els objectes amb els resultats sense portar tots els atributs o atributs d'altres taules que poden haver-se obtingut amb una join, es produirà un error. Per aquest motiu al seleccionar camps específics es proporcionen altres mètodes per realitzar consultes que només s'encarreguen de fer les crides per executar les consultes i retornar els resultats, el tractament de la informació rebuda serà tasca del programador.

Altres aspectes que cobreix Criteria és a l'hora de voler afegir a la consulta diferents relacions lògiques entre diferents columnes. En aquests casos s'ha d'utilitzar forçosament una altre sintaxis,la dels objectes de la classe Criterion, que en realitat són les clàusules que s'afegeixen transparentment dins dels objectes Criteria quan s'utilitzen els mètodes de la classe. Aquesta sintaxi és torna una mica més complicada ja que comporta la creació d'objectes Criterion explícitament, l'adició de les clàusules a cada objecte i després la inserció d'aquests objectes dins de un Criteria.

En el cas que les consultes no es puguin realitzar fent ús de Criteria i les consultes relacionades amb aquests, es proporciona la possibilitat de realitzar consultes amb mètodes de les classes Peer amb consultes escrites en SQL. Però aquest fet no implica que s'hagi de realitzar tot i així tota la feina manualment, ja que per no haver de processar els resultats i crear els objectes es poden utilitzar també a través de les classes Peer les corresponents funcions com populateObjects() que s'encarregarà de retornar un conjunt d'instàncies de la classe objecte omplides a través del conjunt de dades que rep.

Abans quan descrivia la utilització i les funcions que tenia la classe objecte he comentat que una de les formes de realitzar operacions sobre els elements de la taula seria utilitzant les instàncies de la classe i executar-les individualment, però en el cas que es vulguin realitzar aquestes operacions per un conjunt d'elements, hi ha la possibilitat d'executar-les en una crida per tal de no haver de fer un recorregut sobre un conjunt d'objectes de la classe. La sintaxis i utilització dels mètodes resulta

molt similar al de les consultes, utilitzant també objectes de la classe Criteria, i com en el cas de les operacions quan es realitzen en SQL la informació que es retorna quan s'executa correctament fa referència al número d'elements afectats per la operació realitzada.

Altres funcionalitats que aporta Propel i que estan implementades dins les classes dels objectes és la possibilitar de validar les dades per un objecte tal i com ho realitza per executar les operacions demanades a través de funcions validate() per comprovar que les instàncies són vàlides respecte a la definició de la classe i la taula de la base de dades.. A l'hora de realitzar còpies dels objectes també tenim mètodes a la nostra disposició dins de les classes, en general Propel ofereix al programador el conjunt de funcions utilitzades pel propi ORM per realitzar les tasques dins de cada una de les classes i adaptades als seus camps i restriccions.

Conexions a la base de dades

Per definir les connexions a la base de dades que s'utilitzaran es poden utilitzar les comandes de Symfony que faciliten la seva inclusió a través de les tasques que hi ha definides de Propel. Aquestes comandes accepten diversos paràmetres per poder configurar les connexions en totes les opcions que es suporten dins del fitxer de base de dades databases.yml. El fitxer a configurar es troba dins el directori de configuració del projecte, i les opcions que hi han de configuració són:

- Entorn: Les connexions es poden definir per una connexió determinada.
- Aplicació: Les connexions també es poden definir per que sigui utilitzada per una aplicació terminada. En aquest cas la connexió es trobarà dins del directori de configuració de l'aplicació.
- Nom conexió: El nom que se li donarà a la connexió definida, per defecte se l'anomena “propel”.
- Classe: El tipus de motor de base de dades que s'utilitzarà, aquest ha d'estar suportat per Creole. En aquesta versió hi ha suport per MySQL(un dels que utilitzem al projecte), SQLite(també utilitzat), PostgreSQL, Oracle i Microsoft SQL Server. En el cas d'utilitzar una base de dades SQLite generalment hauria d'estar ubicada dins del directori data/ del projecte, en cas contrari s'haurà d'especificar el directori.

Sintaxi del schema

La definició dels esquemes de les bases dades en fitxers YAML és una tasca senzilla gràcies a la notació YAML i la quantitat d'atributs disponibles per determinar les característiques d'una taula. Dins dels esquemes de cada taula es poden definir aspectes relacionats amb les pròpies taules com el nom de la classe en el que es vol traduir(si no en té, el cas per defecte, se li assigna el nom original en format camelCase),el mètode utilitzat per assignar identificadors nous, atributs per saber is la taula conté atributs localitzats, i la taula relacionada, o el package. El package determina en quin paquet de classes estarà ubicada la classe, és important per tenir millor organitzat el directori lib/model/, per cada paquet es crearà un subdirectori dins d'aquest on es trobaran els que s'hagin definit amb el package.

Per definir els atributs es pot posar la clau que identifica al clau només, amb lo qual Symfony esbrinarà el tipus adient per assignar al camp. En aquests casos s'han de tenir en compte certes convencions que es poden utilitzar per fer l'esquema encara més senzill, en el cas que Symfony es trobi amb un camp anomenat com a “id” es prendrà com a enter clau primària i auto increment, i en el cas de trobar una clau de la forma “xxxx_id” es prendrà com una clau forana fent que es referencii a la taula que té com a nom el prefix de la clau. Unes altres convencions que es poden aprofitar estan relacionades amb les claus created_at i updated_at, són camps molt comuns i que s'han d'omplir automàticament, per això Symfony assigna com a tipus data i s'encarrega d'omplir aquests camps transparentment.

Un altre mecanisme que té Symfony per facilitar la definició dels esquemes està relacionat amb les taules d'internacionalització, si es troba una taula denominada com a “xxxx_i18n” la identifica com a contingut internacionalitzat de la taula “xxxxx” en la que afegeix la informació necessària per apuntar cap al contingut internacionalitzat i a més, si no estan definits afegeix dos camps, una clau forana cap a la taula del contingut original, i un altre camp per guardar la cultura en la que s'ha traduït dins de la taula internacionalitzada.

Tots aquests aspectes es poden manualment introduint els tipus i característiques. Es poden definir molts més aspectes per una columna que el tipus, com poden ser restriccions de grandària, si és un camp obligatori, si no accepta valors nuls, clau primària, clau forana, que fer en cas d'eliminar-se una entrada,... Hi han sintaxis alternatives per definir elements com claus foranes, i indexes, que pot ser realment

útil per definir regles i restriccions que no només afectaran una columna, sinó a un conjunt dins d'una fila.

Es poden utilitzar per definir els esquemes de les bases de dades fitxers escrits amb YAML, però Propel treballa amb esquemes definits en XML. En el moment que s'executi alguna tasca de Symfony relacionada amb Propel es crearan uns fitxers dins del directori de configuració del projecte que començaran per “generated-xxxxx.xml” que són els que Propel utilitzarà després que hagin sigut processats els esquemes .yml i creats els esquemes equivalents en XML. Aquests fitxers són un alternativa a les definicions descrites fins ara, ja que es permet utilitzar XML per definir-los directament, tot i que són fitxers més complexos, difícils de llegir i entendre poden haver limitacions en la definició en YAML, per aquest motiu en alguns casos pot arribar a ser obligatori la seva utilització.

Per agilitzar les tasques de creació del model de dades Symfony ofereix un conjunt de tasques que Propel pot executar per tal d'automatitzar els processos intermediaris. A part de la generació del model, és a dir, el conjunt de classes extret a partir dels esquemes de les bases de dades, es permet la creació de bases de dades amb els esquemes que tinguem definits. Aquest procés està composat per dos passos:

“symfony propel:build-sql”

Que generarà un conjunt de fitxers lib.model.sql dins de la carpeta data/sql del projecte que contindrà per cada package del model de dades les sentencies SQL a executar per crear les taules. Aquests fitxers es poden utilitzar manualment en el servidor de la base de dades, o bé amb la tasca de symfony:

“symfony propel:insert-sql”

Addicionalment dins del directori data/sql podem tenir dumps de les bases de dades perquè cada cop que es crei una base de dades es pugui carregar amb un conjunt de dades i treballar amb elles. Aquestes tasques per obtenir copies de les dades i carregar-les són:

“symfony propel:data-dump”
“symfony propel:data-load”

Per defecte els fitxers que es generen i s'utilitzaran són del format YAML, aquests simplifiquen la seva sintaxis respecte als dumps sql que podem trobar generalment, i estableixen les relacions entre files d'una forma més natural i fàcil d'entendre utilitzant les claus que identifiquen cada una.

Aquestes tasques utilitzen Propel, i per tant la seva execució estarà especificada pels motors de base de dades que utilitzem. Per aquesta raó s'hauran de modificar uns paràmetres dins del fitxer propel.ini del directori de configuració del projecte perquè apunti a la nostre base de dades on es vol executar la tasca. Les configuracions no suporten múltiples bases de dades, i per tant s'haurà d'executar i modificar o tenir tants propel.ini diferents com bases de dades on es vulguin executar les comandes.

5.6.13. Adreçament

El sistema d'adreçament tracta sobre la forma en que es reben les URLs de les peticions, com es tracten i es reescriuen per obtenir URLs més llegibles, millor organitzades, i obtenir una millora en seguretat per l'aplicació. Els principals motius pels que Symfony utilitza el paradigma del controlador frontal i el sistema d'adreçament és perquè les URLs poden reflectir i aportar informació important pels usuaris, aquestes apareixen dins de cercadors com a resultats, en els enllaços que es facin i apareixen dins les pàgines que es visualitzen. És per això que és important que aportin informació clara i concisa del que contenen o on apunten.

L'ús d'un sistema també ha de proporcionar mecanismes per tal que si es produeix un canvi en l'estructura de les URLs no s'hagi de realitzar un conjunt de modificacions en tota l'aplicació on s'utilitzin ja que seria contraproduent pel desenvolupador i dificultaria el desenvolupament i el manteniment de les aplicaicons.

Per evitar aquestes incidències es busca un sistema on les URLs puguin ser generades automàticament i que sigui transparent pel programador i a l'aplicació. La seguretat també és un aspecte molt important a tenir en compte dins les URLs, ja que pot aparèixer informació que ajudi a identificar l'arquitectura i funcionament intern de l'aplicació facilitant la entrada i execució de codi no desitjat i altres atacs. El sistema ha de ser capaç de mostrar en les URLs una altre cara que no estigui

directament relacionat amb l'execució interna de l'aplicació com mostrar els recursos i paràmetres utilitzats.

El que es pretén és utilitzar la URL com si formés part de la interfície, de manera que l'usuari pugui identificar fàcilment les parts de l'aplicació i accedir a elles. Aquest té l'avantatge que un cop rebuda una petició la URL que es mostra a l'usuari pot no ser la mateixa que la URI interna i no afectar al funcionament de l'aplicació i les peticions rebudes. D'aquesta forma es poden presentar URLs a l'usuari en les que identificarà i relacionarà els continguts que se li mostraran i tindran un significat que podrà ser utilitzat per autodefinir-se o ser utilitzat per cercadors com ja s'ha comentat, que permetrà que les diferents parts de l'aplicació siguin més accessibles. Aquestes URLs també seran més fàcils de recordar i serà més àgil accedir i navegar a través de l'aplicació, ja que no solament es podrà fer a través dels enllaços trobats, sinó que l'usuari veient l'estructura dels enllaços pot escriure'ls ràpidament, sense haver de modificar o refer tota la URL, o intentar recordar com són els paràmetres o recursos que s'utilitzen en ella.

El control sobre l'aplicació és molt major, ja que al tenir un sistema subjacent que analitza i transforma les URLs es controlaran i redirigiran les URLs que no encaixin en la estructura definida i per tant s'evitarà l'accés o l'intent d'esbrinar els continguts de l'aplicació.

Per obtenir URLs diferents de les que es mostren a l'usuari, es separen les URLs externes de les URIs internes amb el sistema d'adreçament. La sintaxis que hi haurà en les URIs internes serà similar a les típiques URLs amb els paràmetres seguits de “&” i “=” pels valors. Per fer les transformacions d'una banda a l'altre el sistema utilitza les regles que s'hauran definit dins del fitxer routing.yml de cada aplicació on es troben definides.

La definició d'aquestes regles és senzilla, utilitzant la sintaxis de YAML s'especifiquen les regles cada una amb un identificador com a nom, i dins de cada una hi intervenen dos paràmetres. El primer és el format de la URL que es mostrarà a l'usuari final, i l'altre els paràmetres que contindrà la URI interna de symfony. Per realitzar les transformacions primer es reben les peticions al controlador frontal, un cop carregada la configuració i iniciada, el sistema d'adreçament realitza una comparació amb la URL rebuda.

Quan es troba una regla en la que la URL compleix el seu patró, es passen els paràmetres que estan especificats a la regla, continuant l'execució cap al mòdul i acció que han de processar la petició i que tindran accés a la informació que s'ha definit a la regla amb la que s'ha accedeix. A l'hora de fer el pas invers, generar una URL a partir de una URI el sistema funciona de manera inversa. Al definir les regles d'adreçament s'han de veure com un conjunt en el que quan abans aparegui dins del fitxer, més preferència tindrà. El sistema funciona de tal manera que comença a comparar una per una i en la primera que es compleix es deté, deixant de comparar.

Com es pot veure la sintaxis que es pot utilitzar per definir les regles pot ser tan restrictiva com es vulgui, podent deixar elements que no hagin d'afectar a l'aire, o especificant tots i cada un d'ells. També es poden arribar a utilitzar restriccions més enllà de l'estructura, i contemplar característiques com el tipus dels paràmetres, aquestes restriccions es poden utilitzar per discriminar encara més i evitar problemes donats per l'ordre de preferència de les regles.

Un cop arribats al mòdul concret és quan entra en acció uns altres elements que interaccionen amb el sistema d'adreçament. Aquests existeixen per facilitar la generació de URLs externes i perquè en cas que es produeixi algun canvi en les regles o l'estructura de l'aplicació l'impacte sobre les URLs generades sigui mínim.

Aquests elements són els Helper, es va parlar a la capa de la vista d'aquest conjunt de funcions, entre els quals es troba l'anomenat UrlHelper que es el que proporciona les funcions que ens donen aquesta interacció amb el sistema d'adreçament a l'hora de crear URLs. Els Helpers poden rebre tant URIs internes com externes, i les rutes poden ser relatives com absolutes. També hi ha l'opció d'utilitzar en comptes de les URLs el nom de la regla que es vol fer servir seguit dels paràmetres si s'escauen, d'aquesta forma s'evitarà realitzar la comparació amb les regles per mostrar la URL.

Un aspecte important a tenir en compte en quant a la llegibilitat i estètica de les URLs que seran mostrades a l'usuari són els controladors frontals. Quan l'aplicació es trobi en producció el més lògic serà que s'utilitzi per defecte l'entorn de producció amb una configuració adient per obtenir un bon rendiment i altres configuracions que no es tindrien en compte en altres entorns.

En aquests casos el més lògic seria que dins de la URL no aparegués el controlador frontal, ja que no aporta cap informació a l'usuari i hauria de ser transparent que

l'aplicació s'executi amb una o altre configuració. Per aquest motiu tant a les URIs internes com a les URLs externes en el cas de producció no hauria d'aparèixer. Symfony per defecte quan l'aplicació es troba en un entorn de producció el nom de l'script no apareix a les URLs, però es pot configurar dins de settings.yml per canviar aquest comportament.

Perquè les rutes funcionin tal i com s'ha explicat en un entorn de producció caldrà modificar la configuració d'accés d'Apache en el fitxer .htaccess dins del directori web del projecte perquè redirigeixi al controlador que interessa. Aquesta configuració, ben feta també evitarà que es pugui accedir a la resta d'entorns a través de la URL.

En aquest fitxer es poden configurar aquests comportaments afegint regles que s'interpretaran pel mòdul mod_rewrite d'Apache que s'encarrega d'analitzar i transformar URLs dinàmiques en estàtiques basant-se en la configuració que s'hagi fet.

5.6.14 I18N – L10N

Per tractar amb la internacionalització de les aplicacions s'han de tenir en compte els diferents continguts que intervenen en ella i que hem vist durant els anteriors capítols. Aquests poden ser les dades que l'usuari hagi guardat i que es carreguin , que travessen des de la capa del model fins arribar a la vista i a la inversa mostrades sota demanda.

Per una altre banda també hi ha la part de la interfície de l'aplicació, és a dir, la part estàtica, de la que formen part també dels elements que es mostren en les accions de l'aplicació, i els elements que s'hagin de construir segons les regles d'estàndards i formats que hi hagin en cada cultura com poden ser dates, numeracions o unitats de mesura. Symfony ofereix un conjunt de funcionalitats i tècniques per afrontar aquest procediment amb la major transparència possible pel funcionament de l'aplicació.

La primera característica de Symfony en aquest àmbit és el tenir un accés directe a la informació que dictaminarà com s'haurà de mostrar les dades, la cultura de cada un dels usuaris. La cultura conté la informació necessària per determinar l'idioma i

la localització de l'usuari que està accedint a l'aplicació. Symfony facilita i manté un accés a aquesta informació guardant-se dins la sessió de l'usuari.

L'accés a la cultura de l'usuari doncs es realitza a través de la sessió de l'usuari que ja hem vist com és accessible des de tant la capa del controlador, com la capa de la vista. Aquesta funcionalitat només comporta l'ús de dues funcions, per accedir i modificar el valors. Amb la cultura de la sessió es pot controlar una cultura definida a l'usuari, però per definir-la s'ha d'utilitzar altres mètodes.

Com la informació de les preferències i cultures suportades per l'usuari són un aspecte que controlen els navegadors i envien com a informació amb les peticions al servidor, la informació s'introduirà i es podrà accedir dins l'aplicació a través de la classe sfWebRequest. Els mètodes que es poden utilitzar proporcionen accés als idiomes de l'usuari, i realitzar comparacions amb els idiomes que es tinguin a l'aplicació per determinar el més adient per l'usuari.

Un cop es té definida la cultura que s'utilitzarà dins de l'aplicació es pot realitzar la internacionalització i localització adequada dels continguts de l'aplicació. Dins del context de dades que s'emmagatzemen a la base de dades ja s'ha vist com es poden definir taules utilitzades per emmagatzemar versions traduïdes de les dades d'una taula definides a l'esquema. Aquestes dades que inicialment estarien en una sola taula es divideixen en dues taules, en una hi hauran les dades que no s'han de traduir, i en la que es crea nova conté la informació que s'ha de guardar traduïda amb la informació necessària per identificar a quina fila de la taula original pertanyen i la seva cultura.

Aquest esquema es trasllada al model de dades, i quan s'obtinguin instàncies de les classes que contenen dades internacionalitzades el comportament continuarà sent el mateix que si la classe no traduís les dades, el motiu es que si s'hagués de fer d'una altre forma, el codi i la execució seria explícitament depenent de la cultura que hi hagués establerta. Per evitar aquest fet els mètodes de les classes objecte treballen per defecte amb la cultura de la sessió sense haver d'especificar-ho, però en el cas que es vulgui executar d'una altre manera suporten un segon paràmetre per utilitzar una altre cultura que no sigui la definida a la sessió.

Aquest comportament també es troba a les classes Peer dels objectes, proporcionant mètodes per realitzar consultes amb paràmetres relacionats amb les cultures possibles. Aquest mecanisme com ja s'ha comentat aconsegueix que la

part de l'esquema internacionalitzada sigui transparent a l'hora de reflectir-se al model de dades.

Per una altre banda, per realitzar les traduccions de contingut estàtic, és a dir, contingut que es pot trobar a la part de la capa de la vista com textos a les plantilles i als layouts, els continguts dels menús dins de parcials, i dels components, es poden dur a terme utilitzant uns diccionaris que contindran les traduccions de cada llengua dels continguts de les aplicacions.

Symfony pot realitzar la majoria de la feina que comporta la generació, estructuració i ús dels diccionaris, per activar-ho primer de tot s'ha de configurar el paràmetre dins de settings.yml de cada aplicació que ho necessiti. Un cop activat en cada aplicació es podran utilitzar els diccionaris a través del Helper 'I18N' amb la funció de traducció '__()'.

Quan la internacionalització està activada la funció pren el paràmetre i cerca dins del diccionari de la llengua de la cultura de l'usuari per mostrar la traducció. En el cas de no trobar la traducció o no existir un diccionari corresponent es mostrarà el paràmetre rebut a la funció. Aquestes funcions també ofereixen suport per l'ús de paràmetres dins del contingut a internacionalitzar, i en el cas de tenir un element com pot ser una frase, dins la qual es vol mostrar el correu de l'usuari actual que segons l'usuari s'haurà de carregar i serà diferent. En comptes de dividir la frase en dues i entre elles afegir el correu, s'afegeix el camp dins la traducció i serà passat a través de la funció d'internacionalització com a paràmetre per ser introduït on està marcat en la traducció.

Els fitxers utilitzats per realitzar les traduccions estan escrits en XLIFF, un llenguatge basat en XML del que ja s'ha parlat. Els diccionaris XLIFF es troben dins del subdirectori de cada aplicació 'i18n/', dins d'aquest directoris trobarem un altre per cada cultura que tingui un diccionari. El nom de les cultures que s'utilitzarà ha d'estar en el format ISO, i els noms d'aquests fitxers per defecte seran de la forma “messages.XX.xml”. No obstant, no tots els continguts han d'estar en aquests fitxers, o ser obligatori utilitzar aquests, es poden definir d'altres diccionaris amb noms diferents però per tal que Symfony els pugui utilitzar s'haurà d'especificar en la funció del Helper el nom del diccionari a utilitzar.

Symfony un cop més ens facilita la tasca de crear i omplir els diccionaris amb els continguts a internacionalitzar de l'aplicació, no realitzarà les traduccions que

s'hauran de afegir manualment pels encarregats de traduir les interfícies. La tasca que ofereix Symfony a través de la línia de comandes és:

symfony i18n:extract [--display-new] [--display-old] [--auto-save] [--auto-delete] aplicació cultura

Aquesta tasca només es pot utilitzar amb els diccionaris per defecte messages.XX.xml, i permet actualitzar el seu contingut afegint nous elements i eliminant els que no ja no es trobin a l'aplicació.

Per configurar la cultura que l'aplicació mostrarà per defecte en el cas que l'usuari no tingui cap assignada a la seva sessió, o les opcions que tingui no estiguin suportades, es pot definir dins del fitxer settings.yml de cada aplicació.

Quan es realitzen modificacions en la configuració relacionada amb la cultura dins l'aplicació els canvis no afectaran als usuaris actius, ja que aquests ja contindran dins la seva sessió la informació que determina el comportament de la l'aplicació respecte a la cultura, i no es veurà afectada per aquests canvis fins que la sessió expiri. En el cas d'haver de realitzar canvis en l'aplicació s'ha de mantenir una consistència en la versió de producció, si les configuracions que es modifiquessin canviessin en el mateix moment que s'efectuessin els usuaris podrien trobar-se amb canvis inesperats que farien semblar a l'aplicació inestable.

5.6.15 Gestió de les aplicacions

A part de les funcionalitats implicades en el disseny i creació de les aplicacions com per la codificació, Symfony proporciona un conjunt d'eines complementàries que aporten informació d'execució de les aplicacions per poder revisar i corregir el funcionament d'aquestes. Les eines que es poden utilitzar per aquests fins són els logs, i eines de depuració.

Per la gestió de logs s'ofereix dues possibilitats, per una banda es pot configurar mitjançant els fitxers de configuració el comportament del sistema de logging, i per una altre, dins de l'aplicació es poden utilitzar mètodes per accedir a les classes que s'encarreguen de loguejar informació o personalitzar i estendre les funcionalitats d'aquestes classes.

Per configurar els logs i la informació que es guardarà durant l'execució de les aplicacions es poden definir i modificar els paràmetres dins dels fitxers de cada aplicació settings.yml. En aquests es pot sobreescriure els paràmetres que defineixen el nivell d'informació que es mostrarà en els logs de PHP per l'execució de cada aplicació sense que afecti a la resta d'scripts que s'executin amb PHP.

Un altre aspecte a configurar són els logs que ofereix Symfony per cada aplicació dins del directori 'logs/' del projecte amb el nom de l'aplicació seguit del seu entorn. Les configuracions que es poden realitzar afecten al nivell d'informació que es mostrarà en ells i també al format amb el que es mostraran els missatges dels logs realitzats per Symfony automàticament.

Els missatges que es poden arribar a enregistrar en els logs de Symfony poden tenir un nivell de precisió molt alt respecte a l'execució de les aplicacions, podent mostrar per cada petició capçaleres d'aquestes, paràmetres, els filtres que s'executen, les accions, mòduls, els elements a renderitzar a la capa de la vista, les consultes que SQL que es realitzen a base de dades,... En un entorn de desenvolupament s'utilitza el nivell més alt, enregistrant tots els missatges en els logs, mentre que en un entorn de producció el logging està desactivat per defecte. Els nivells es poden configurar dins dels fitxers factories.yml de cada aplicació, podent definir diferents nivells segons l'entorn.

Si es vol loguejar informació personalitzada dins dels fitxers de logs que genera Symfony es pot realitzar des de qualsevol de les capes en les que es necessiti amb mètodes diferents. Per agregar un missatge des d'una de les accions, la classe sfComponent de la que s'hereta ja proporciona un mètode logMessage($message, $level) per poder fer-ho.

Per poder introduir missatges als logs des de un component de la vista s'haurà d'utilitzar el helper 'Debug', que proporciona els mètodes corresponents. Si el missatges es vol afegir des de un altre lloc, com una classe dins la capa del model de dades, s'haurà d'obtenir el context que ja sabem que és accessible des de qualsevol part de l'aplicació, i a partir del context obtenir la instància de la classe Logger i cridar al mètode corresponent per introduir un missatge.

Una altre alternativa podria ser redefinir el comportament dels logs creant classes que heretin de sfLog i que sobreescriguin el mètode doLog($message,$priority) que és el que s'utilitza per realitzar-los. També es pot crear un altre mecanisme per

realitzar els logs a apart del de symfony, per això es proporciona en la llibreria de Symfony la classe sfLoggerInterface, com el seu nom indica és una interfície per poder crear classes de logs compatibles amb les de Symfony.

Per evitar haver d'utilitzar scripts externs a Symfony o que la col·lecció de logs creixi fins a menjar-te per un oblit, es proporcionen comandes per fer neteges del directori de logs i per automatitzar la rotació i el backup dels fitxers de logs de les aplicacions.

Depuració

Symfony també té recursos per utilitzar durant l'etapa de desenvolupament per facilitar la detecció d'errors que puguin ocórrer i així agilitzar el desenvolupament de les aplicacions. Per realitzar aquestes operacions s'ofereix un context diferent durant l'execució de les aplicacions. Quan està activat a través de la configuració del controlador frontal en el que es vol depurar l'aplicació en l'entorn es veuen alterats els comportaments següents:

- Per la configuració no s'utilitza la caché, en cada petició es comproven els fitxers de dades i configuració per no haver de netejar la cache i que els canvis es vegin reflectits immediatament.
- En cas d'error en comptes de realitzar la redirecció al mòdul/acció/template que s'encarrega de mostrar un missatge de que no s'ha pogut realitzar l'acció, es mostra una pila de seguiment de com s'ha arribat a produir l'error.
- Són disponibles eines de depuració com la barra de Symfony.
- El mode de depuració de Propel s'activa, i quan es produeixi un error en una crida al utilitzar les classes de Propel es mostra informació detallada i un seguiment també de l'error produït.

Les excepcions produïdes són de gran ajuda ja que permeten fer un seguiment de part del codi executat a través de les crides que s'han anat executant i visualitzant part del contingut de cada un dels blocs executats. A part de l'excepció que es produeix es mostra un missatge donant informació del tipus d'error que s'ha produït. Les excepcions també són accessibles pel programador i es poden utilitzar per provocar situacions i controlar-les.

Web debug toolbar

Es tracta d'una barra que apareix en la part superior dreta de l'aplicació quan ens trobem en el context de depuració i que conté informació sobre l'execució de l'aplicació pel desenvolupador. Dins de la barra trobarem els diferents elements:

- Logo de Symfony!
- Versió de l'aplicació
- vars & config: En una primera línia informa sobre la configuració de l'aplicació, aspectes com l'estat de la cache, xdebug, accelerador PHP,... Per una altre banda es mostra un conjunt de dades relacionats amb la configuració i paràmetres de l'aplicació:
  - Petició: Es mostren el conjunt de paràmetres i atributs de la petició rebuda.
  - Resposta: Es mostren els paràmetres i atributs que s'envien junt amb la resposta al navegador de l'usuari.
  - Usuari: Es mostra les dades de la sessió de l'usuari.
  - Configuració: Es mostren tots els paràmetres de la configuració obtinguda al analitzar els fitxers de configuració.
  - Globals: Es mostren les variables globals de PHP i els seus continguts.
  - PHP i Symfony: Mostrà versions de software extensions i ubicacions.
- Logs & messages: Mostra els missatges dels logs que s'han generat durant la petició que s'ha rebut i executat. Dins del llistat de missatges estan ordenats per execució, cada missatge conté el tipus, que no és més que la classe de la que prové el missatge, i el missatge. En cada missatge es pot desplegar un seguiment de les crides que s'han realitzat fins a realitzar el missatge de log. Dins d'aquest llistat de logs també apareixeran els que s'hagin introduït manualment.
- Base de dades: Es mostren les consultes SQL completes amb els valors dels paràmetres afegits que s'han realitzat a la base de dades i el temps que han trigat en executar-se cada una.
- Memòria consumida al executar la petició.
- Temps que ha trigat en servir-se la petició.

Deployment

Symfony també ofereix eines per sincronitzar diferents versions de projectes entre diferents servidors. El procés normalment consta de tres passos, freeze, transferència i unfreeze del projecte. El motiu de la realització d'un freeze és la forma en la que es treballa durant el desenvolupament i manteniment d'un projecte.

En un entorn que no sigui de producció es treballa amb les llibreries de Symfony fora del directori del projecte per no tenir repetides en cada projecte o les diferents versions, i que puguin ser utilitzades i actualitzades sense que els projectes s'hagin de veure afectats. Per aquest motiu s'han d'incloure les llibreries de Symfony i els elements com imatges, css que utilitzi symfony per defecte, temporalment dins del projecte per convertir-lo en una aplicació independent i tancada sobre si mateixa.

Un cop es té el projecte congelat i preparat es pot transferir al servidor destinatari. Per realitzar la transferència es proporciona la possibilitat d'utilitzar rsync per fer una transferència intel·ligent del projecte i així estalviar haver de reenviar cada cop tot el projecte. A més de la millora en rendiment en la transferència, també es proporciona una capa addicional de seguretat, ja que Symfony ofereix accés al servidor destinatari a través de connexions amb SSH per realitzar la sincronització.

Les configuracions utilitzades a través de la línia de comandes eviten haver de introduir totes les dades per les connexions que es vulguin utilitzar permetent la seva configuració dins del fitxer properties.ini al directori de configuració del projecte.

Tasks

Symfony ofereix un conjunt de tasques a través de la línia de comandes, que en realitat són un conjunt d'scripts ubicats al directori task/ de les llibreries de Symfony que s'executen a través de la línia de comandes. Dins d'aquestes es troba la classe sfBaseTask de la que hereten la resta de tasques. Aquesta classe s'encarrega de proporcionar les funcionalitats bàsiques de les tasques que s'executen amb les comandes.

Altres beneficis d'utilitzar tasques es que tenen accés a totes les llibreries de Symfony i si es fa que s'hereti de sfPropelTask en comptes de la sfBaseTask, es podrà accedir a les funcionalitats de la capa del model. Les tasques oferides per Symfony es poden sobreescriure definint tasques amb el mateix nom, i el mateix poden fer els plugins.

5.6.16 Formularis de Symfony

Els formularis a Symfony són tractats d'una forma especial, tenen una estructura jeràrquica de classes que defineix tots els elements i aspectes dels formularis, i consten dels seus propis mecanismes per realitzar les tasques de creació, visualització, validació , emmagatzematge de dades i confirmació.

Els formularis estan representats a Symfony per la classe sfForm de les llibreries del nucli. Els formularis que es crean seran objectes d'una classe creada específicament per descriure'l i que heretarà de sfForm. Quan es mostra un formulari podem veure diversos camps que el composen amb els que l'usuari interaccionarà i és on podrà introduir les dades que després hauran de ser enviades i processades pel servidor.

Aquests camps del formulari estan representats com a widgets, es proporciona un gran conjunt de widgets per afegir als formularis, però en cas de no ser suficients es poden afegir de nous amb plugins com sfFormExtraPlugin utilitzat en el projecte. Els widgets hereten de la classe sfWidget de les llibreries de Symfony que proporciona les funcionalitats bàsiques d'aquests com són la creació, configuració, visualització, o afegir opcions i atributs.

Els widgets utilitzats dins dels formularis hereten de la classe sfWidgetForm, aquesta els proveeix de funcionalitats pel seu ús en formularis com poden ser obtenir l'identificador de l'element en el document HTML o definir el format de com es generarà l'atribut, posar-lo com a camp invisible,... A l'hora de visualitzar-los s'ha de tenir en compte que no guarden cap mena d'estat de la seva visualització, i per tant, si es decideix mostrar els camps del formulari un per un a les vistes, es pot decidir no mostrar-ne algun, com repetir-ne.

Els widgets poden ser de molts tipus, i tots poden acceptar diferents opcions i atributs per configurar el seu comportament i propietats que contindran al ser visualitzats. Les opcions són múltiples ja que no s'espera el mateix d'un camp de text en el que s'ha d'omplir amb un string, que una llista de selecció, en la qual es voldrà afegir un conjunt d'opcions que es visualitzin al ser desplegada.

Les opcions que es poden configurar solen ser les que es podrien afegir als camps introduint-los en HTML pur, en un camp de text es pot afegir un contingut per defecte, o determinar la classe i identificador dins del document per exemple. A

més els mètodes que s'apliquen als widgets tenen cura de controlar els continguts que puguin ser introduïts automàticament sense ser necessari que el mecanisme d'escapament estigui activat.

A l'hora de definir els widgets dins d'un formulari aquests es fan utilitzant la classe sfWidgetFormSchema, aquesta classe representa un conjunt de widgets i és la forma en que el formulari conté i agrupa tots els seus widgets. A través d'ell es facilita l'accés com si d'un array de widgets es tractés. També proveeix d'un conjunt de mètodes per aplicar al conjunt de widgets com pot ser l'assignació de valors per defecte amb setDefaults().

Dins del formulari, els widgets es defineixen en del mètode configure(), aquest serà el constructor dels formularis, i utilitzant en el seu interior el mètode setWidgets(), que accepta un o més widgets en forma d'array. Les classes resultants dels formularis es guarden en el directori lib/form, tot i que no es obligatori, ja que si es prova a desar-los en qualsevol altre directori de les llibreries del projecte els acceptarà, ja que el sistema d'autocàrrega que és l'encarregat de buscar i incloure'ls no fa distinció entre les llibreries.

Un cop definides les classes dels formularis es poden utilitzar dins les accions i plantilles de l'aplicació. Dins de l'acció s'haurà de crear el formulari creant una instància nova de la classe del formulari, al crear la instància es cridarà automàticament a la funció configure que inicialitzarà el formulari deixant-lo llest per mostrar-lo a la vista.

Dins de la vista una de les formes, la més ràpida i senzilla per mostrar el formulari es fer un 'echo' de la variable que el conté, aquesta acció desencadena un toString sobre el formulari, el qual serà executat sobre cada un dels widgets que tenen definides les funcions per tal de mostrar el codi HTML pertinent. Mostrant el formulari d'aquesta forma s'haurà de contenir-lo dins d'una taula ja que per tal de mostrar els elements del formulari d'una forma ordenada i estructurada ho fa dins d'elements <tr></tr> que mostra directament amb cada element del formulari a visualitzar. Aquest comportament es pot configurar dins dels formularis amb la classe sfWidgetFormSchemaFormatter que és la que s'utilitza per definir l'estil del format amb el que es mostraran els camps del formulari.

Un altre aspecte que es cobreix automàticament amb aquesta tècnica són els títols o definicions que apareixeran amb els camps com a <label></label>. A symfony

per defecte quan s'inrtodueix un widget es guarda també el seu corresponent 'label', però també es poden especificar dins del formulari. De la mateixa manera que es defineixen els widgets amb setWidgets() dins del formulari existeix un mètode setLabels() al que se li passarà un array amb el nom del widget i el seu label corresponent que li volem assignar. De manera opcional fent ús del widgetSchema que conté tots els widgets es pot accedir a un en concret i fer servir el mètode setLabel per només modificar un en concret.

Fins aquí s'ha tractat la creació i visualització dels formularis i els seus elements, però com s'ha comentat abans els mecanismes de symfony no es detenen en aquest punt, també ofereixen el necessari per validar-los. Un cop a l'usuari se li mostra l'usuari i l'omple amb les seves dades, es reenvia al servidor per tal que siguin validades les dades, i determinar si són correctes, o no s'han introduït correctament.

Per realitzar aquestes comprovacions es fa ús dels validadors que es definiran i s'aplicaran per cada un dels camps que tinguem definits dins dels formularis. La forma en que es defineixen aquestes regles de validació segueix un esquema molt similar al vist fins ara amb els widgets.

Cada una de les regles és una instància d'un objecte de la classe del validador específic, que a l'hora hereta de sfValidatorBase de Symfony. La relació entre widgets i validadors es defineix utilitzant el mètode setValidators() a la configuració del formulari de la mateixa forma que amb setWidgets(). Aquest mètode a més crea un sfValidatorSchema on introduirà tots els validadors que es passin com a paràmetres i afegirà regles de validació per defecte per controlar que tots els camps del formulari siguin validats i que no s'introdueixi cap camp extra no especificat a la configuració del formulari protegint així la integritat del conjunt de dades rebut.

Hi ha un conjunt de validadors que s'encarreguen de molts aspectes individuals dels camps, grandària de textos, format d'un correu introduït,... però es pot tenir la necessitat utilitzar validadors més complexos. En aquests casos podem utilitzar validadors lògics que ens permetran definir regles en les que es tinguin en compte més d'un aspecte per un mateix camp.

Si el que es necessita és una validació no tant específica per un camp concret, es poden fer servir els validadors globals. Aquestes regles es defineixen amb pre i post validadors, dins dels quals es poden definir validadors com els lògics o realitzar

altres comparacions amb sfValidatorSchemaCompare que permet realitzar comparacions entre widgets diferents en una mateixa regla.

Un cop tenim completament definit els formularis es podran utilitzar els mecanismes de validació dels formularis. Per realitzar la validació de les dades, una acció molt comuna és primer comprovar com han arribat les dades, un cop comprovat, el primer que s'ha de realitzar és la obtenció de les dades del formulari. Les dades estaran dins dels paràmetres rebuts amb la petició, accedint a través de l'objecte $request, i lligant-les a la nova instància del formulari (ja que les instàncies de les classes no són persisteixen entre peticions). Acte següent s'activa el mecanisme de validació, i es comprova si el formulari compleix amb els validadors o no.

En el cas que el formulari sigui vàlid es pot dur a terme el processament de les dades que sigui necessari. En el cas que no es passi la validació del formulari, es pot tornar a mostrar de nou el formulari directament. Aquest fet és possible degut a que al executar bind() si el procés de validació falla s'encarrega de guardar els missatges d'error dins del formulari perquè puguin ser mostrats sense cap feina extra junt amb les dades introduïdes per l'usuari que s'hagin de mantenir (no com passwords, captchas,.. que es buidaran).

A l'hora de personalitzar els formularis també es pot modificar aspectes relacionats amb la validació com els missatges que es mostren als usuaris quan la validació falla, les classes dels validadors s'encarreguen d'oferir en els mètodes per crear-los i configurar-los la possibilitat d'afegir paràmetres per modificar els missatges per defecte i d'altres per quan no es compleixin certes característiques com la grandària d'un text.

Per mostrar el formulari a l'usuari s'ha utilitzat una de les formes en les que no es tenia en compte aspectes com l'ordre o el posicionament dels camps dins de la plantilla on es vol utilitzar ja que s'utilitzava una instrucció per mostrar-lo de cop, però hi ha una altre via per poder configurar la visualització del formulari.

Amb <?php echo $form ?> no ens hem de preocupar dels camps, els títols, i els errors, ja que els mecanismes del formulari s'encarreguen automàticament, però per l'altre via que es presentarà depenent del mètode que s'utilitzi haurem de tenir en compte tots aquests elements. Els mètodes a utilitzar per dividir la presentació del formulari:

- Per files: Utilitzant renderRow() sobre tots els camps dels formularis serà molt proper a fer un echo $form exceptuant el fet que no es tractaran els errors globals.
- Per columnes: Utilitzant render(), renderLabel(), renderError() per mostrar cadascún dels apartats dels camps dels formularis, en aquest cas també faltarà la gestió els errors globals.

Per tractar els errors es convenient utilitzar un mètode de comprovació d'errors abans de mostrar-los. Aquest comportament es pot estendre als mètodes renderGlobalErrors() i hasGlobalErrors() per gestionar l'aspecte que quedava pendent respecte a l'altre via per mostrar els formularis.

Resultarà molt probable que els formularis que s'hagin d'utilitzar estiguin relacionats amb la creació i edició d'informació relacionada amb el model de dades de les aplicacions, és més, és probable que s'utilitzin per crear registres de les classes del model i emmagatzemar-les a la base de dades. Recordem que aquestes classes estan controlades per Propel a la capa de dades, i en els formularis relacionats amb el model tornarà a aparèixer.

Propel proporciona una altre tasca per generar més classes, aquesta s'encarrega de generar les classes dels formularis que representen les classes respectives en el model.
“symfony propel:build-forms”

Després d'executar la tasca, Propel haurà generat dos classes de formulari per cada taula que hi hagi en el model, classes 'BaseXxxxxForm' a lib/form/base i 'XxxxForm' dins del directori lib/form del projecte (en el cas que no hi hagi package determinat, si n'hi ha s'afegirà un altre nivell més en els directoris /lib/form igual que a lib/model/ aquesta estructura sempre es preserva si està definida). Addicionalment es crea una classe BaseFormPropel que hereta de sfFormPropel de la qual heretaran les classes dels formularis creades, aquesta ens permetrà afegir configuracions a personalitzar a les que ja té Propel.

Com s'ha pogut imaginar, la classe Base de cada formulari és la que conté els widgets i els validadors basats en les característiques i els atributs de la taula a la que representen. Les classes objecte dels formularis hereten de les Base, i seran les que podran ser utilitzades per personalitzar els formularis a mostrar sense haver de modificar els generats per Propel.

A l'hora de personalitzar els formularis a través de les classes objecte tenim total llibertat per modificar el comportament en el mecanisme de validació i els elements que es reflectiran al formulari visualitzat. Mitjançant el widgetSchema i validatorSchema es té accés a tots aquests elements del formulari. Un cop accedit un es poden realitzar les operacions que cada classe té definides o heretades com si s'estigués creant i configurant des de el propi formulari on es defineix.

Es poden afegir opcions i nous atributs com amb setOption() i setAttribut(), o es poden afegir i/o canviar els seus valors. En el cas dels widgets es pot redefinir el seu comportament creant-lo de nou i assignant-lo en la seva posició del widgetSchema.

En el cas dels validadors també es poden modificar de la mateixa manera i d'afegir-ne d'altres més específics encara proporcionats per Propel fent ús de la opció d'introduir instàncies de la classe Criteria per poder realitzar consultes amb les que es podran contrastar les dades rebudes del formulari i ser validat. En el cas que hi hagi algun camp que no vulguem que es mostri com dates de logueig o d'últim accés que no depenen de l'usuari es poden treure dels formularis amb unset().

Al fer ús dels formularis generats per Propel personalitzats o sense, les tècniques que s'han descrit per mostrar i validar els formularis no han de variar, en comptes d'això, el que es pot fer és afegir una única instrucció més perquè les dades del formulari es guardin correctament, sense haver de realitzar cap altre acció addicional.

El mètode definit als formularis per realitzar-ho és save(), en ell és on resideix la lògica que determina si l'acció a realitzar es guardar una nova instància de la classe a la base de dades, o si el que s'està realitzant és un actualització de les dades d'un objecte que ja hi és la base de dades.

El mètode save() també es pot personalitzar dins de cada classe objecte del formulari, en aquest mètode s'hauria d'introduir el codi relacionat amb accions extres a realitzar cada cop que s'ha de guardar un objecte de la classe del formulari. S'ha de tenir en compte que el codi introduït dins de save() no forma part de la transacció, si el que es vol definir es codi que formi part de la transacció llavors s'haurà de afegir dins del mètode doSave().

Un altre aspecte a considerar en la configuració del mecanisme de persistència de la informació dels formularis és el mètode updateObject(), aquest mètode es crida

abans de realitzar doSave() i es pot utilitzar per realitzar modificacions en l'objecte que s'ha de fer persistent a la base de dades de l'aplicació.

Una altre funcionalitat dels formularis que arriba a ser molt útil quan tens dades repartides entre diferents taules que s'han de complementar per formar un mateix formulari és el mètode mergeForm(). Aquest mètode fusiona els elements d'un altre formulari dins del que s'executa, i al visualitzar i utilitzar els mecanismes es comportarà com un de sol.

La responsabilitat d'executar els mecanismes correctament al guardar el formulari recauran en el formulari dins el qual s'ha realitzat la fusió, i per tant haurà de fer les crides oportunes per guardar o actualitzar el formulari que s'haurà fusionat amb ell. Les tècniques utilitzades segueixen sent les mateixes, personalitzant les funcions save() i updateObject() dins del formulari es poden afegir les parts necessàries per definir el nou comportament.

Els formularis també tenen continguts que formaran part de la interfície que es pot internacionalitzar. Els continguts dels formularis a traduir són els títols de cada camp i els missatges d'error que poden aparèixer durant el procés de validació.

Com que aquests continguts són estàtics i formen part de la interfície, la tècnica utilitzada en aquests casos serà de nou els diccionaris. En comptes d'haver de carregar per cada una de les classes el Helper adient per poder fer ús de la funció '__()', els formularis presenten un mètode amb el que definir un diccionari a utilitzar en cas de voler internacionalitzar el seu contingut i no haver d'afegir cap més instrucció, exceptuant els widgets que es veuran afectats per la cultura que s'utilitzi com poden ser els selectors de zones horàries, llenguatges, o país, en aquests caldrà passar la cultura en la que s'haurà de formatjar el seu contingut.

5.6.17 Plugins

Els plugins són paquets que busquen estendre les funcionalitats de les aplicacions. A Symfony són un conjunt d'elements comuns als projectes com llibreries o mòduls tancats en un paquet per poder ser distribuïts i utilitzats en altres projectes. No només és una solució a l'hora d'integrar funcionalitats ja existents de tercers als nostres projectes, sinó que també es pot utilitzar per refactoritzar codi que creiem

convenient o útil per ser utilitzat en altres projectes, ja siguin en els nostres o d'altres desenvolupadors.

Dins d'un plugin ens trobarem una estructura de directoris molt semblant a la utilitzada dins d'un projecte, però sense tants elements. Dins dels directoris de configuració trobarem la configuració del plugin en un fitxer PHP, la configuració d'adreçament que pugui aportar el plugin i els esquemes de bases de dades si es que contenen el seu propi esquema.

Aquest esquema serà introduït automàticament a les nostres bases de dades, i quan el model es generi també es generaran les classes del model necessàries ubicant-les però en el seu directori lib/ i no en el del projecte.

Dins les llibreries es trobaran totes les classes que es puguin utilitzar amb el plugin, a diferencia de les aplicacions aquí es localitzaran en els seus directoris pertinents les classes dels filtres, tasques, formularis i model. En el cas de contenir diccionaris d'internacionalització i fitxers utilitzats en la capa de la vista, es trobaran en els directoris i18n i web/ respectivament del plugin.

I per últim també es poden introduir mòduls amb accions i plantilles corresponents, aquests aniran directament en el directori modules del plugin i seran accessibles des de l'aplicació. Aquests es permet sobreescriure'ls creant els mateixos dins les aplicacions i afegint els elements que es vulguin personalitzar i utilitzar en comptes dels original del plugin.

Per la distribució i instal·lació dels plugins només s'haurà de crear un fitxer en XML que contindrà la especificació dels continguts, dependències i d'altre informació sobre el plugin. Aquesta informació serà utilitzada per la tasca “plugin:install” de Symfony per generar els paquets i instal·lar-los correctament.

SfGuard

sfGuard és un dels plugins més utilitzats en la comunitat. Symfony per defecte ofereix una capa de seguretat limitada i senzilla, de manera que el propi creador de Symfony va afegir aquest plugin com a funcionalitats extres en comptes d'estar inclòs dins les llibreries del framework, ja que segons el tipus de projecte que es vulgui dur a terme la capa de seguretat i autenticació de Symfony pot ser suficient.

sfGuard s'encarrega per una banda d'estendre les funcionalitats relacionades amb l'autenticació dels usuaris i per una altre proporciona una millor gestió de l'autorització i els credencials del usuaris, afegint un model ampliat amb classes relacionades amb els usuaris, grups, i permisos.

Els mòduls que sfGuard proporciona són 4:

- sfGuardAuth: Conté les accions i templates per defecte que es poden executar per realitzar el login i logout dels usuaris.
- SfGuardGroup: Administració dels grups dels usuaris.
- sfGuardPermission: Administració dels permisos.
- sfGuardUser: Administració dels usuaris.

Per defecte les accions dels mòduls sfGuardGroup, sfGuardGroup, sfGuardUser venen buides. Aquestes classes no proporcionen funcionalitats per utilitzar dins les accions de les aplicacions enfocades a l'usuari, sinó que són mòduls pensats per poder gestionar des d'un backend les dades relacionades amb aquests tres elements de la seguretat de les aplicacions.

El motiu pel que estan buits és perquè al tractar-se de mòduls orientats a un backend les funcionalitats principals que s'esperen són les d'un esquema CRUD (Creation/Retrieval/Update/Delete), Propel precisament pels mòduls amb aquestes funcionalitats ofereix una tasca “propel:generate-crud” que s'encarrega de crear els mòduls dins de les aplicacions que se li especifiquin amb accions i templates relacionats amb el llistat d'elements i formularis per realitzar les operacions descrites automàticament.

En canvi, el mòdul sfGuardAuth si que conté un mòdul plenament funcional, que es pot utilitzar configurant les aplicacions correctament. Els aspectes a cobrir en la configuració són els mateixos que fins ara, afegir els filtres que es vulguin utilitzar, especificar les accions realitzades en l'autenticació i activació de mòduls a utilitzar dins l'aplicació.

Amb el sistema d'autocàrrega de classes de Symfony no caldrà realitzar cap configuració addicional per utilitzar les classes. Però és probable que haguem de modificar el comportament d'algun de les classes que ja tinguem creades com l'usuari i l'encarregada de les funcionalitats de seguretat.

El plugin sfGuard afegieix 7 taules a la nostra base de dades:

- sf_guard_user: Conté les dades dels usuaris que sfGuard controla com passwords, usernames, últim accés, métode de codificació de password, si està actiu, l'últim accés i si és super adminsitrador.
- sf_guard_permission: Identifica els permisos existents, amb identificador, nom i descripció.
- sf_guard_user_permission: Relacions entre els usuaris i els permisos.
- sf_guard_group: Identifica els grups d'usuaris existents.
- sf_guard_user_group: Cada entrada relaciona un usuari amb un grup.
- sf_guard_group_permission: Cada entrada relaciona un grup amb un permís.
- sf_guard_rememeber: Identifica les Ips guardades relacionades amb els seus corresponents usuaris.

Al realitzar les tasques de creació de base de dades, i càrrega de dades també s'afegiran les dades inicials necessàries per utilitzar el plugin, com els permisos i grups inicials disponibles ubicats al fitxer fixtures.yml del subdirectori data/ del plugin.

Si tenim informació en una classe de l'usuari, amb el nou esquema s'haurà de dividir en dos parts. sfGuard ha de contenir la informació que s'ha descrit, mentre que la resta d'informació que tinguem pot anar en la nostra pròpia classe d'usuari que haurà de tenir una referència a la seva instància de sfGuardUser.

Si es tenen funcions relacionades amb l'usuari i la gestió de la seguretat s'haurà de fer que aquesta classe hereti de sfGuardSecurityUser, que a l'hora hereta de sfBasicSecurityUser. D'aquesta manera no es perden les funcionalitats que es tenien anteriorment i es poden utilitzar les de sfGuard.

També es poden sobreescriure els elements del mòdul sfGuardAuth, ja sigui l'acció, el template o els dos. Aquesta capacitat ja es va explicar al definir el comportament i la personalització dels mòduls.

SfGuardSecurityUser és potser la classe més important en tot el procés d'autenticació, ja que és aquesta la que conté tots els mètodes per realitzar les accions d'autenticació, per accedir als objectes sfGuardUser i als seus perfils, comprovació de credencials i grups. Sobre ella recau l'execució i desencadenament dels mecanismes de seguretat normalment, a no ser que s'hagin definit els propis mètodes, que a l'hora cridaran a aquests.

gpServices

Dins del projecte les aplicacions s'han basat en una arquitectura orientada als serveis. Dins d'una aplicació es pot accedir a diferents tipus de serveis amb múltiples instàncies de cada un que són les que ofereixen els continguts, que a l'hora depenen del tipus de servei al que corresponen. Aquesta gestió es realitza d'una manera uniforme en tots els serveis, l'arquitectura necessària havia de tractar amb diferents tipus de serveis que es podrien modificar durant el temps de vida del projecte i les aplicacions sense haver de realitzar canvis en el mecanisme de funcionament dels serveis.

Aquesta és l'arquitectura base sota les aplicacions i mòduls desenvolupats dins del projecte per tal d'oferir un capa d'abstracció sense la qual no es podrien executar els serveis utilitzant la configuració i esquema actual, que repercutiria en la quantitat de codi que s'hauria d'implementar per poder executar cada un dels tipus de serveis diferents i haver de redefinir en cada cas molts dels aspectes de configuració tractats en els anteriors capítols.

Aquesta arquitectura es va veure plasmada en la realització d'un plugin per part d'un dels membres que va participar en el projecte, que en comptes d'afegir la configuració pertinent al projecte i a cada aplicació, sumant un conjunt de llibreries afegides a les del projecte per aportar les funcionalitats necessàries, veient les possibilitats i aplicacions que podria tenir en altres projectes va encapsular tots els elements relacionats amb l'arquitectura de serveis en un plugin.

L'esquema del plugin és el següent:

- Service: Conté informació relacionada amb una instància de servei:
- User_permission_service: Estableix una relació entre un permís donat a un usuari per a un servei.
- group_permission_service: Estableix una relació d'un permís donat a un grup per a un servei.

Dins les llibreries del model del plugin es troben algunes les classes relacionades amb les taules i d'altres que no, a les que s'han afegit mètodes relacionats amb l'aplicació:

- Service.php: Conté mètodes per obtenir i generar les URLs dels serveis, i la funcionalitat per donar permisos a l'usuari per el servei utilitzat.

- ServicePeer.php: Proporciona mètodes per cercar serveis pel seu tipus i permisos, i cerques dels serveis als que un usuari té accés, també es proporciona accés a la pila de serveis de l'aplicació en la petició actual i per accedir al servei actual i als seus paràmetres.
- ServiceType.php: Aquesta classe representa el tipus d'un servei, no està representada dins la base de dades ja que pel moment són un conjunt de dades bastant reduït al que s'accedeix constantment per realitzar només consultes amb el mecanisme de serveis.
- ServiceTypePeer.php: Mètodes per accedir a la pila dels tipus de serveis que es troben actius en la petició actual, per cercar tipus de serveis pel nom, i per obtenir els subtipus d'un servei si es que en té.
- UserPermissionServicePeer: Mètodes per realitzar consultes sobre els permisos d'usuaris als serveis, es poden realitzar cerques especificant els paràmetres o buscant per servei.

Dins les llibreries del plugin:

- sfGuardServiceSecurityUser: Classe que hereta de sfGuardSecurityUser i esten el comportament i les comprovacions dels permisos afegint una capa més per comprovar no només que té els credencials per accedir al mòdul i acció, sinó que també els té per accedir al servei al que s'intenta executar. També proporciona mètodes per obtenir els serveis als quals l'usuari que està connectat té accés.

Dins les llibreries de validadors del plugin:

- serviceParameterValidator: Validador utilitzat dins dels formularis per crear un paràmetre amb un format vàlid i un valor únic amb el que poder adreçar-lo.

Dins les llibreries de formularis del plugin:

- ServiceForm: El formulari d'un servei mostra dos elements propis, el títol del servei i el paràmetre que s'utilitza per generar la URL del servei, per la resta de camps el formulari obté el tipus de servei que s'està intentant crear. Aquest és accessible des del formulari, i un sabent la classe del servei que s'intenta afegir pot crear una instància del seu formulari i fusionar-la dins del formulari del servei. Aquesta és la solució que s'utilitza per no haver d'estar creant nous formularis o que el mecanisme de validació i persistència dels objectes no s'hagi de redefinir per cada tipus de servei nou.

- ServiceParameterForm: Hereta de ServiceForm i és una variació de ServiceForm que no mostra el paràmetre.

Ara es descriurà el cicle de vida dels serveis dins de les aplicacions perquè quedi més clar el seu funcionament i les seves utilitats. Primer de tot els serveis es crearan ja siguin carregant-se a través d'un arxiu amb un conjunt de dades preparades a la base de dades o utilitzant els formularis de serveis.

La creació d'un nou servei es realitzarà dins d'una acció que pot crear un formulari de dues formes. En la primera, saben el tipus de servei que es vol crear, s'utilitzarà un formulari ja creat i que hereti de ServiceParameterForm on es podran realitzar les modificacions que es creguin necessàries per mostrar el formulari personalitzat.

L'altre forma s'utilitza quan es volen crear nous serveis sense saber fins que arriba l'execució de l'acció el tipus de servei que serà. En aquest cas es crea un formulari ServiceParameterForm al que se li passen com a paràmetres un servei buit excepte pel seu camp tipus de servei.

Quan el formulari es crea i es configura es realitza el procés de fusió amb la classe a la que pertany el tipus de servei que s'ha pogut accedir a través de l'objecte que s'ha enllaçat al formulari i que contenia el paràmetre amb el valor assignat. Un cop el formulari s'envia i es valida es pot crear el servei, encarregant-se ServiceForm de guardar les dades com ja s'ha explicat en l'apartat dels formularis.

Un cop creats els serveis es poden accedir a ells. Per accedir es pot entrar la URL directament si es coneix, de no ser així es pot accedir a través de navegació per enllaços de l'aplicació. Els enllaços es generen accedint a les URLs que poden retornar les instàncies de serveis amb els corresponents mètodes com getIndexURL(), aquests mètodes accedeixen al tipus de servei que conté entre d'altre la URL per les accions que pot contenir el servei. Si no hi hagués una jerarquia de serveis no es necessitaria res més per generar la URL, però els serveis estan organitzats de manera que existeixen un serveis pares o arrel des dels que es pot accedir a altres que en pengen i que no poden existir sense ells. Tota aquesta informació es defineix als fitxers de configuració de l'aplicació.

Aquesta jerarquia i pila de serveis es reflecteix en la URL, que conté els paràmetres dels serveis dins dels que s'està. Aquests serveis són accessibles a través de mètodes que ja s'han comentat i s'utilitzen per generar les URLs també. Un cop

obtingut el mòdul i acció del servei al que s'ha de redirigir s'obtenen els paràmetres dels serveis pares navegant a través del camp dels serveis que els representa com a tals fins arribar a l'arrel i s'afegeixen a la URL per completar-la.

Un cop tenim la URL i s'intenta accedir a un servei s'executa el mecanisme de seguretat que realitza les comprovacions per permetre o denegar l'accés. Aquí es quan comença l'execució de sfGuardServiceSecurity, dins del mètode hasCredential() s'encarrega de comprovar que les piles de serveis i tipus de serveis actuals siguin vàlides, comprovant que els serveis identificats a la URL també ho són.

Un cop es tenen les piles es realitzen les comprovacions relacionades amb els permisos que té l'usuari assignats als serveis tenint en compte tots els aspectes dels credencials com són els introduïts pel plugin i per sfGuardAuth. S'ha de tenir en compte un aspecte relacionat amb els credencials, com que els serveis estan agrupats i ordenats seguint una jerarquia, per facilitar la gestió de permisos es permet activar un comportament anomenat a service type com a cascada, en el cas que un tipus de servei suporti aquest comportament si un usuari té permisos per accedir a aquest servei també es permetrà l'accés als serveis fills que contingui sense haver d'estar donant permisos per tots i cada un d'ells que es el que s'hauria de fer si es desactiva aquest comportament.

5.7 MySQL

L'aplicació web conté un conjunt de dades persistents que s'utilitzen pel seu funcionament. Aquestes dades que poden anar canviant han d'estar emmagatzemades en un entorn que permeti la seva identificació i accés. També s'ha d'oferir un entorn que permeti gestionar i realitzar un manteniment en l'estructura que s'utilitza per emmagatzemar-les. Totes aquestes tasques es poden dur a terme utilitzant sistemes de gestió de bases de dades, que proveeixen d'eines per la gestió i administració, interfícies de programació, servidors SQL i clients per accedir al servidor i facilitar les tasques a realitzar.

L'SGBD utilitzat al projecte és MySQL, aquest gestor ofereix moltes possibilitats i té aspectes molt positius a destacar:

- Velocitat. MySQL actualment pot competir amb altres SGBDs com Oracle, Microsfot SQL Server, i en proves de rendiment s'obtenen resultats que el posicionen entre els millors.
- Facilitat d'ús, la instal·lació i configuració és més senzilla que d'altres grans SGBD's que necessiten de suport tècnic especialitzat.
- SQL. Té suport per llenguatge SQL (amb algunes variacions implementacions com la majoria de SGBDs que el suporten)
- Funcionalitats. Suporta diferents tipus d'interfícies per accedir al servidor, es pot integrar en diversos llenguatges de programació actuals utilitzant la API disponible que hi ha per cada llenguatge.
- Connexió i seguretat. MySQL proporciona connectivitat a través de la xarxa amb els seus propis sistemes d'autenticació i soporta diferents tipus de connexions a través de la xarxa com SSH.
- Portabilitat. MySQL es troba disponible per la majoria de sistemes operatius i plataformes com Windows, famílies BSD i Linux.
- Espai. MySQL comparats amb altres gestors ocupa molt poc espai en disc.
- Cost. MySQL s'ofereix sota diferents llicencies, per usos no comercials està disponible sota llicencia GPL, i per l'ús en situacions comercials disposa de llicencies menys restrictives.
- Open Source i distribució. El codi està disponible i es pot adaptar a les necessitats de cada projecte.

Tot i les gran quantitat d'eines i possibilitats que pot tenir desenvolupar aplicacions utilitzant MySQL, el seu ús en el projecte està relegat en un segon pla. Symfony junt

amb Propel s'encarrega de proporcionar unes capes d'abstracció que eliminen la necessitat d'utilitzar per part del desenvolupador amb les APIs disponibles pels llenguatges que s'utilitzin per accedir a les dades i interactuar durant la fase de codificació amb MySQL directament. I per una altre banda està el suport per realitzar tasques que comprenen les fases de creació i manteniment(càrrega i copies de la base de dades) de les taules de les bases de dades, el seu accés i manipulació que es pot realitzar a través de tasques, backends, mòduls, accions i formularis generats per Propel que representen el conjunt de dades.

Hi ha un aspecte de SQL i de MySQL en concret per la seva pròpia sintaxis, que s'ha hagut d'utilitzar i ha sigut de gran importància durant l'etapa de desenvolupament. Com més endavant podrem veure, les necessitats a l'hora de decidir quina informació es guardava ha anat variant durant el procés de desenvolupament, i aquests canvis han afectat a l'estructura del model de dades i les taules que les representen dins les bases de dades. Al existir una versió de l'aplicació amb dades ja introduïdes i que s'utilitzaven dins l'aplicació, al fer un canvi en l'estructura on es guarden aquesta informació s'ha de processar i realitzar les manipulacions necessàries per poder-les seguir utilitzant, i el més important, sense haver de tornar a introduir manualment totes les dades.

La solució passava per realitzar aquestes operacions fora de l'aplicació, ja que implicaven diverses versions de dades i esquemes que no es trobarien dins d'una mateixa aplicació. Les tasques a realitzar estaven enfocades a executar procediments i transaccions. SQL ofereix la possibilitat d'ús de procediments emmagatzemats, es tracta d'una via per agrupar un conjunt de transaccions SQL a realitzar que s'executaran com si es realitzes una crida a una funció en altres llenguatges de programació. Dins de MySQL, a partir de la versió 5 es dóna suport pels procediments emmagatzemats propis de MySQL(amb una sintaxi que difereix en certs aspectes com les declaracions, i els cursors de la especificada en SQL).

A l'hora de definir un procediment s'ofereixen totes les operacions SQL que es poden realitzar dins del cos del procediment, és a dir, es tenen a la disposició les instruccions necessàries per consultar i modificar la base de dades que són exactament les operacions que es necessitaven i es volien executar. A més s'ofereix l'ús d'altres sentències per realitzar les operacions d'una forma més procedimental:

- Paràmetres. Els procediments poden rebre paràmetres que seran accessibles dins del seu cos, i es pot definir el seu comportament, si són d'entrada/ sortida, o solament d'entrada o sortida.

- Declaració de variables. Es permet la declaració de variables amb tipus definits, els tipus disponibles seran els mateixos que són suportats per MySQL en les taules. Dins del procediment es podran utilitzar per guardar resultats de consultes o es poden fer servir individualment amb clàusules com SET. També tenen suport per valors per defecte i es fa ús de nocions d'abast de les variables segons el bloc en que siguin declarades.
- Estructures de control. Per controlar i afectar al flux d'execució del procediment es poden utilitzar les típiques estructures de control:
  - Condicionals if/else/case
  - Bucles com while i repeat...until, loop
- Labels. Són declaracions que es fan a sentències ubicades en el cos del procediment que es poden utilitzar junt amb les estructures de control per redirigir el flux d'execució amb instruccions com LEAVE per abandonar un bucle declarat amb un nom per exemple.
- Control d'errors. Es poden utilitzar també els codis d'error per configurar manegadors per executar en cas de produir-se errors específics i tractar-los.
- Cursors. Es poden utilitzar cursors amb consultes per tal de poder recórrer cada fila del resultat i poder realitzar les operacions pertinents.

Els procediments es criden amb la sentencia CALL nom_procediment(), es poden declarar per ser emmagatzemats dins la base de dades i executar-se puntualment o periòdicament. Un altre avantatge respecte a realitzar operacions fent ús de les APIs en algun dels llenguatges de programació dels que disposen d'elles, és que fent servir un procediment s'evita el tràfic a la xarxa i la connexió al servidor de la base de dades, ja que els procediments s'executen directament dins i passen a formar part d'un altre element més dins la base de dades.

5.8 SQLite

Un altre tecnologia utilitzada durant el desenvolupament del projecte és el gestor de base de dades SQLite, aquest gestor resulta molt oportú en etapes de desenvolupament o com a base de dades d'aplicacions, ja que a diferència dels SGBD's comuns SQLite passar a formar part de la pròpia aplicació, integrant-se dins del codi d'aquesta.

SQLite està suportat per Symfony i Propel i també s'integra sense problemes en el framework. La seva configuració és inexistent durant la seva instal·lació, hi han eines per a la visualització, clients o frontends per accedir a les bases de dades, i també es proporciona un conjunt d'utilitats a través de la línia de comandes. Altres punts a favor de la utilització de SQLite és l'ample de banda, al no establir-se connexions amb un servidor extern cada vegada que s'ha de realitzar consultes i operacions a la base de dades.

Una de les configuracions possibles com es dóna durant el projecte és el tenir diferents bases de dades, una utilitzant un SGBD extern a l'aplicació i una altre amb SQLite amb informació vital per la seva execució. Amb aquesta separació de les dades en diferents vies es redirigeix part de l'ample de banda i latència que pugui haver en algunes parts de l'aplicació.

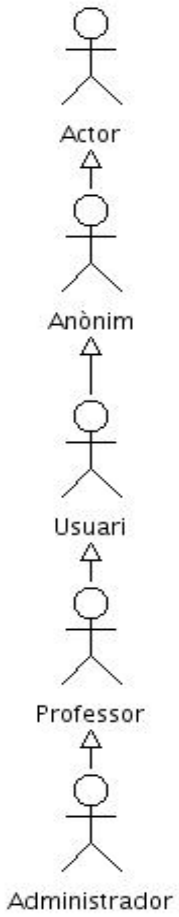
S'ha de veure SQLite com un motor de base de dades molt compacte i poc pesat, amb moltes menys extensions i funcionalitats que els grans SGBDs, i per tant en certs aspectes pot arribar a ser molt ràpid fent la seva feina. En entorns en que es realitzin operacions de consulta, insercions i modificacions es realitzaran molt ràpidament, ja que s'executaran directament a la RAM si estan carregades les dades a memòria, sinó al disc. Però, en el cas que les consultes es compliquin es notarà una baixada de rendiment, ja que no pot competir amb la sofisticació d'altres SGBDs que incorporen optimitzadors i planificadors per les consultes.

Pels desenvolupadors també ofereix una gran ajuda a l'hora de crear i distribuir bases de dades durant la fase de desenvolupament, ja que per tenir diferents bases de dades es tan senzill com tenir diferents fitxers que es poden transmetre sense cap mena de manteniment addicional.

6 Especificació

En aquest apartat es presentaran els diagrames de casos d'ús propis de cada tipus d'usuari de l'aplicació i també es farà una explicació d'aquests acord amb els requeriments funcionals que s'han descrit anteriorment. Els casos d'ús s'encarreguen de definir els escenaris en els que es realitzen les accions per part de l'usuari, definint els elements que intervindran i repassant els esdeveniments i les diferents vies d'execució que poden existir per cada un dels casos.

Abans de esquematitzar i definir els casos d'ús es descriurà la jerarquia que existeix dins l'aplicació dels usuaris. Es poden definir 4 tipus d'usuaris dins l'aplicació cadascun tenen privilegis diferents, i es podrien ordenador verticalment segons la seva especialització. Es poden veure com diferents nivells dins de l'aplicació, cada nivell es superior a l'inferior, i pot realitzar totes les accions que els nivells inferiors més unes altres que només poden dur a terme aquest i els nivells superiors. L'esquema d'actors quedaria així:



Aquesta estructura ha sigut escollida perquè les accions a realitzar dins de l'aplicació estan escalades, la base de tots es l'accés als continguts de l'aplicació, si un és un usuari ha de poder accedir a certs continguts, aquests poden estar limitats en el conjunt total del que es pot accedir, però sempre s'ofereix la possibilitat de d'utilitzar alguns serveis.

A l'hora, el motiu lògic d'aquesta estructura és perquè l'usuari mai vegi les seves possibilitats retallades. Quan l'usuari estigui a cert nivell ha de tenir accés a les funcionalitats que li siguin útils, i en el cas de que vulgui obtenir privilegis majors no s'ha de trobar amb que certes accions que podia realitzar li han sigut retallades, o ha de necessitar diversos tipus de comptes per realitzar tasques diferents. Amb aquest model l'usuari pot escollir quin és el tipus d'usuari que li és més adient observant fins a on vol poder arribar a l'aplicació. Pot decidir tenir capacitat per crear els seus propis continguts, però podent seguir accedint als continguts que abans podia tenir accés.

Ara descriurem els tipus d'usuari, el perquè de la necessitat de la seva especialització i quines són les possibilitats que té dins de l'aplicació i els serveis que conté.

Administrador

L'usuari que té com a rol administrador està definit i relacionat amb una escola. Les funcionalitats específiques del seu rol estan relacionades amb tasques de manteniment i gestió de l'escola a la que pertany.

Aquest rol va néixer per la introducció en l'aplicació de l'entorn d'escola. Aquest entorn proveïa als usuaris que es volguessis identificar dins d'una escola una forma de reunir un conjunt de serveis que fins ara existien, però que s'entenien com a entitats individuals ja que no es podien establir relacions entre elles i la gestió de diverses aules, i llavors la possibilitat de facilitar accés i gestió era inexistent. Per aquest motiu s'havia d'afegir un nou tipus d'usuari que seria l'encarregat de gestionar tot aquest entorn.

Per realitzar les tasques que se li permeten havia de tenir accés als altres serveis amb els que estigués relacionada l'escola, ja que sense ell no podria obtenir

informació sobre els serveis que s'estan executant i quins continguts es poden trobar en ells, per aquest motiu es va deixar que tingués totes les funcionalitats que hi ha entre els altres usuaris.

Les seves funcions específiques són les de gestionar els elements que estan relacionats amb l'escola a la que pertany. Dins d'aquests elements es troben les aules i els seus serveis (es permet l'accés a ells com si tingués rol de professor), les quals pot gestionar decidint crear-ne o administrar les ja existents, i l'accés que es dona als usuaris. També es el que té la capacitat per assignar permisos d'escola als usuaris, sense els quals no es pot accedir ni crear continguts dins d'ella.

Dins d'aquest rol hi ha un cas especial, es tracta de l'autor i propietari del servei de l'escola, aquest és un administrador de l'escola també que se li ofereix un tractament especial per tal que els seus privilegis no es vegin afectats per altres administradors de l'escola.

Professor

El rol de professor és immediatament inferior a l'anterior d'administrador i per tant no pot realitzar les funcions específiques d'aquest. Els professors són usuaris que estan relacionats amb una escola també directament, ja que l'escola a la que estiguin associats es a través de la qual es podrà accedir a les seves aules.

Al estar relacionats amb una escola també s'ofereix un part de la gestió que es troba dins d'ella, aquestes funcionalitats li permeten gestionar les seves aules i proporcionar accés a altres usuaris que es trobin a l'aplicació. Els usuaris que pertanyin a aquest rol han de tenir en compte que els administradors de l'escola que es troben en un nivell superior tenen la possibilitat d'accedir als seus continguts i administrar-los en el cas que ho creguin convenient. Seria una estructura similar a dels moderadors i administradors d'un fòrum.

Aquests usuaris junt amb els administradors, són els que tenen la possibilitat de crear i afegir continguts a les aules, sense els quals no podrien accedir els altres usuaris.

Usuari

El rol de l'usuari és un altre rol bàsic que ha existit en l'aplicació. Els usuaris que es trobin dins d'aquest grup no poden accedir a les funcionalitats de creació d'aules i de continguts a no ser que se'ls hi atorguin permisos pels nivells superiors.

Aquest perfil d'usuari no estan relacionats directament amb l'escola, els permisos que reben d'accés estan donats per accedir a les aules en concret. Aquests permisos els hi permeten accedir en mode de lectura a les aules i als seus serveis, però mai editar-los, ni accedir als entorns de gestió que disposen els altres usuaris.

Anònim

El rol d'usuari anònim és un rol especial i que es va derivar de la nova estructura de l'aplicació. A diferència dels altres no es un usuari pròpiament dit, en termes d'execució de l'aplicació si que se'l tracta com si ho fos, però internament, dins de la base de dades no es troba cap dada emmagatzemada.

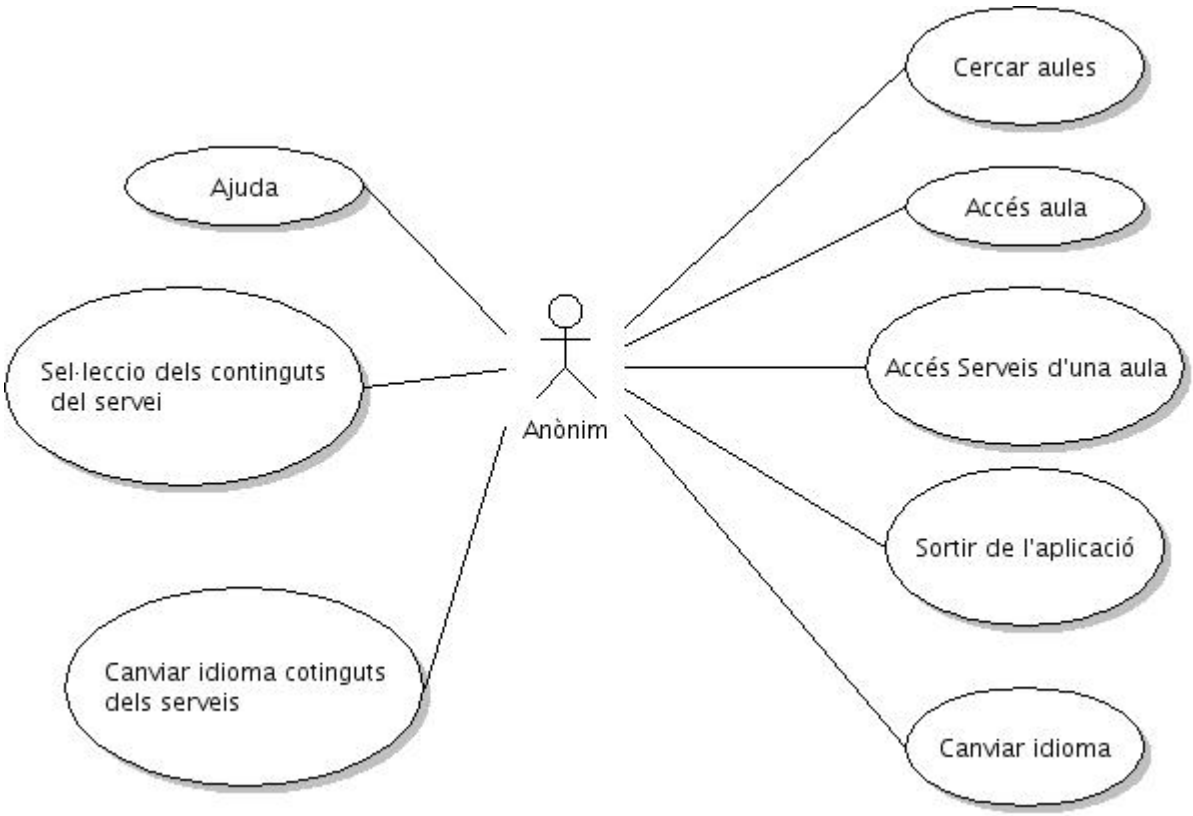
La vida d'un usuari anònim dura mentre algú accedeix a una aula concreta. La necessitat dels usuaris anònims prové de la solució donada per tal d'oferir als professors una forma de que terceres persones puguin accedir a les seves aules i continguts sense haver de registrar nous usuaris que només voldran accedir a una aula en concret. D'aquesta forma els responsables de facilitar aquest accés a usuaris anònim seran els professors, que seran els qui assignaran claus a les seves aules.

Els usuaris anònims són per tant específics per cada aula, els seus privilegis són molt semblants als d'un usuari semblant, però a diferència d'ells només podran accedir a una aula en concret i si coneixen el password corresponent.

Ara que ja s'han descrit els rols que interactuaran en l'aplicació podem definir per cada un d'ells els casos d'ús que poden realitzar, començant des del rol més baix, l'usuari anònim s'aniran descrivint els casos que li siguin accessibles, però sense repetir-ne en cada nivell els dels nivells inferiors.

6.1 Diagrames casos d'ús

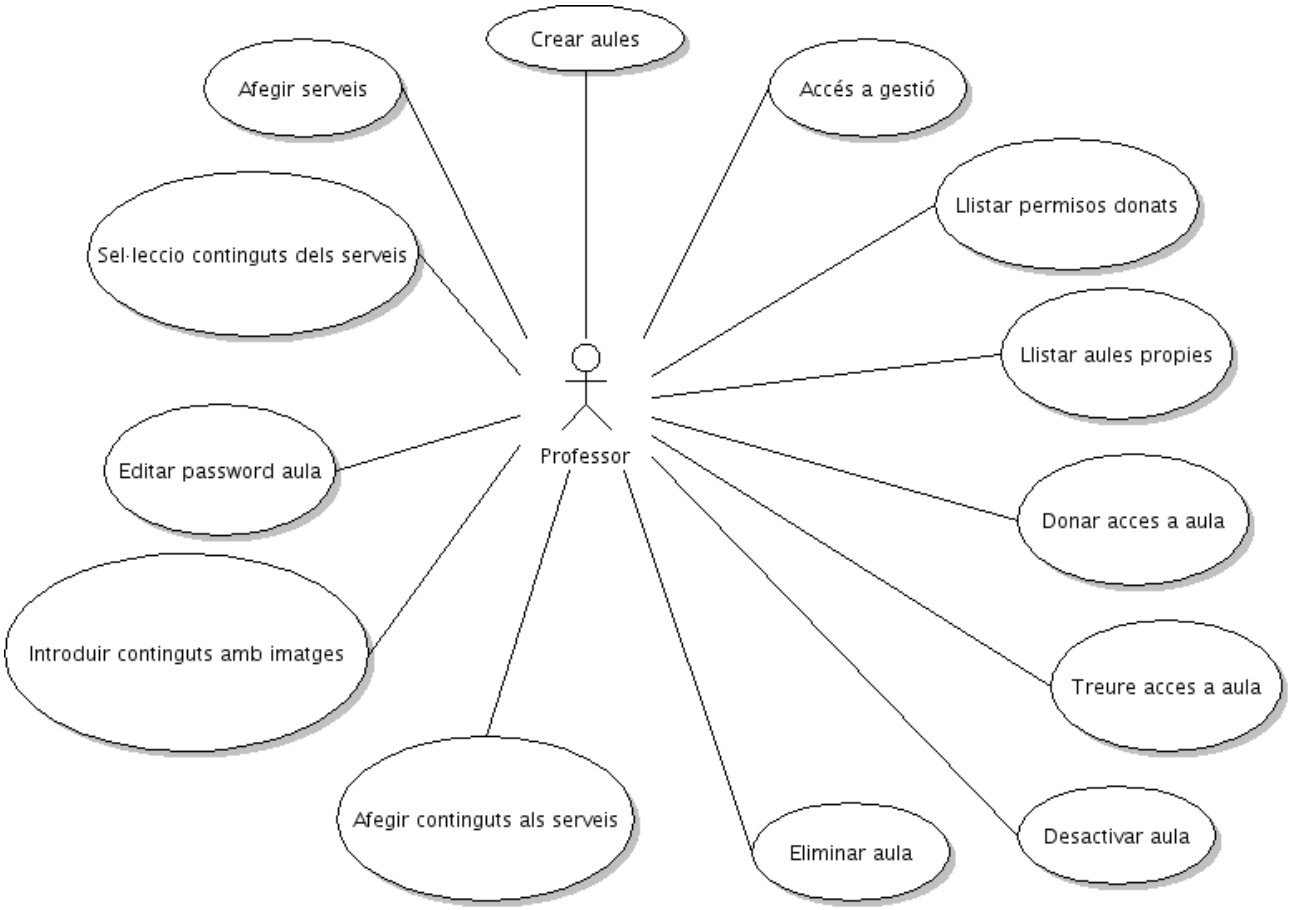
6.1.1 Casos d'ús del rol anònim



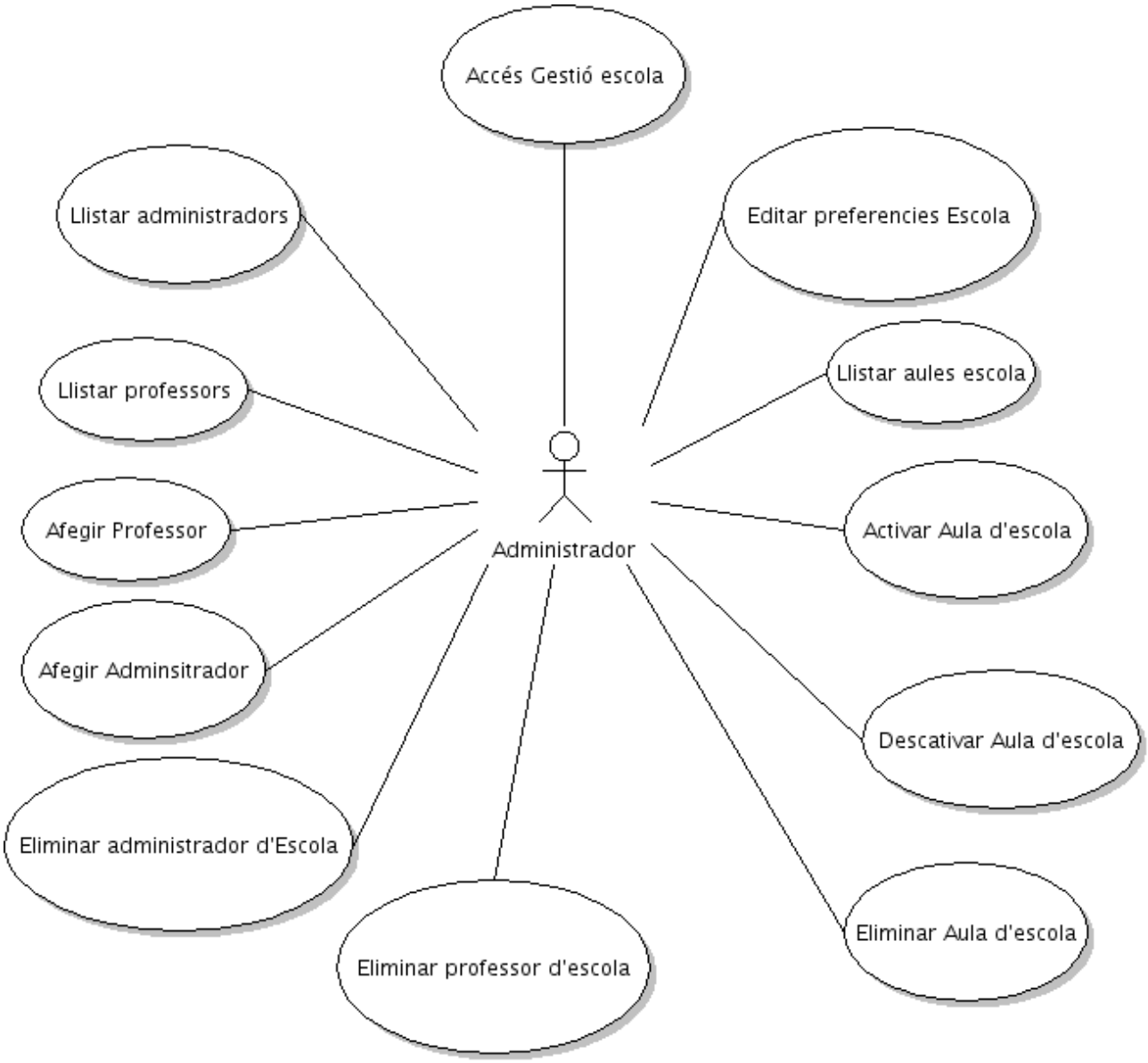
6.1.2 Casos d'ús del rol Usuari

El diagrama de casos d'ús d'usuari no es mostra ja que és el mateix que el de l'usuari anònim excepte per la forma en que accedeix a l'aplicació.

6.1.3 Casos d'ús del rol Professor



6.1.4 Casos d'ús del rol Administrador



6.1.4 Assignació de casos d'ús

Com s'ha comentat abans, els diagrames de casos d'ús no estan complerts, ja que no no s'han introduït totes les funcionalitats que poden realitzar els usuaris perquè els diagrames fossin més clars. En la següent taula es pot veure quines casos d'ús poden realitzar cada usuari i es pot comparar i veure amb claredat les diferències que hi han entre els rols que s'han descrit anteriorment.

Funcionalitat	Anònim	Normal	Professor	Administrador
Accés aplicació		X	X	X
Sortir aplicació	X	X	X	X
Cercar aules	X	X	X	X
Accés a aula	X	X	X	X
Accés serveis d'una aula	X	X	X	X
Accés a ajuda	X	X	X	X
Seleccionar continguts d'un servei	X	X	X	X
Canviar idioma de l'aplicació	X	X	X	X
Canviar idioma d'un servei	X	X	X	X
Crear aules			X	X
Afegir serveis			X	X
Seleccionar continguts a afegir			X	X
Introduir imatges			X	X
Editar clau de l'aula			X	X
Afegir continguts als serveis			X	X
Accés a gestió d'aules propies			X	X
Llistar aules propies			X	X
Llistar permisos donats			X	X
Donar permis a usuari			X	X
Treure permis a usuari			X	X
Desactivar aula			X	X
Eliminar aula			X	X
Accés a gestió d'escola			X	X
Llistar aules d'escola				X
Activar qualsevol aula de l'escola				X
Desactivar qualsevol aula de l'escola				X
Eliminar qualsevol aula de l'escola				X
Llistar administradors				X
Afegir administrador				X
Treure administrador				X
Llistar professors				X
Afegir professor				X
Treure professor				X
Editar dades escola				X

6.2 Descripció dels casos d'ús

En el següent apartat es procedeix a descriure els casos d'ús que s'han especificat oferint informació relacionada amb l'intercanvi d'informació que es produeix entre l'usuari i l'aplicació, descrivint quins els cursos possibles dels casos d'ús i les interaccions que es produeixin i com es duen a terme.

6.2.1 Casos d'ús d'accés a l'aplicació

L'aplicació conté un mecanisme d'accés i autenticació per tal de poder escenificar els rols que s'han descrit i poder oferir un control dins del sistema. Per tal d'accedir a l'aplicació i els serveis que conté s'ha de procedir a realitzar algún d'aquests casos d'ús.

Accés a l'aplicació

Actor	Usuari
Entrada	Nom usuari, contrasenya
Sortida	-
Resum	L'usuari accedeix a l'aplicació
Curs típic d'esdeveniments	
Actor	Sistema
1. L'usuari entra al domini on es troba l'aplicació	
	2. El sistema redirigeix i mostra la pantalla de login.
3. L'usuari introdueix el seu usuari i contrasenya	
	4. El sistema comprova les dades i el redirigeix a la primera aula a la que té permisos o al perfil si no té cap aula assignada.
5 . L'usuari ha accedit a l'aplicació.	
Curs alternatiu	
	4. Les dades de l'usuari no són vàlides s'envia un missatge d'error
5. Apareix el missatge d'error en la zona de missatges del login	

Sortir de l'aplicació

Actor	Usuari
Entrada	-
Sortida	-
Resum	L'usuari es desconnecta i surt de l'aplicació

Curs típic d'esdeveniments	
Actor	Sistema
1. L'usuari selecciona sortir de l'aplicació des del menú superior	
	2. El sistema tanca la sessió de l'usuari i envia un missatge de redirecció al navegador cap a la pantalla de login
3. L'usuari és redirigit a la pantalla de login de l'aplicació.	

Curs alternatiu
-

Cercar aules

Actor	Usuari
Entrada	Nom aula
Sortida	Llistat d'aules (nom aula + subtítol aula + nom autor)
Resum	L'usuari realitza una consulta d'aules a través del nom i se li mostra una taula amb els resultats a través dels quals es pot clicar i intentar accedir a elles.

Curs típic d'esdeveniments	
Actor	Sistema
1 . L'usuari introdueix una consulta dins del camp de cerca	
	2. El sistema realitza una consulta a la base de dades utilitzant com a criteri els títols dels serveis i obté un llistat
3. A la mateixa pàgina es mostra una taula amb un llistat en la part inferior de la pàgina mostrant el títol, subtítol i autor. L'usuari pot clicar en els noms per intentar accedir a una aula.	
	4. El sistema redirigeix a l'acció d'accés a l'aula
4 . L'usuari es redirigit a la pàgina per accedir a l'aula.	

Curs alternatiu	
3. Es mostra una taula buida, l'usuari pot tornar a realitzar una altre cerca des de la mateixa pàgina.	

Accés a una aula

Actor	Usuari
Entrada	Clau de l'aula
Sortida	-
Resum	Un usuari accedeix a una aula amb el corresponent password
Curs típic d'esdeveniments	
Actor	Sistema
1. Usuari intenta accedir a una aula.	
	2. El sistema valida que no té permisos i el redirigeix a la pàgina d'accés
3. Es mostra la pàgina d'accés en la que introdueix el password de l'aula i envia les dades	
	4. El sistema valida correctament el password.
5. Es mostra la pàgina d'inici a l'aula	
Curs alternatiu	
	4. El password rebut no és correcte i no es permet l'accés a l'aula.
5. Es mostra a la zona d'errors el missatge de password no vàlid	

6.2.2 Casos d'ús de gestió de continguts

En aquests casos d'ús es troben les funcionalitats que estan directament relacionades i implicades en els continguts que es poden accedir i utilitzar dins les aules i els serveis que contenen. Les funcionalitats que s'han dissenyat aporten una flexibilitat més alta, com per exemple la internacionalització que es pot realitzar i utilitzar amb diferents aplicacions.

Accés a continguts d'un servei

Actor	Usuari
Entrada	Opció del servei
Sortida	-
Resum	L'usuari escull una opció del servei a utilitzar
Curs típic d'esdeveniments	
Actor	Sistema
1. L'usuari accedeix a la pàgina principal d'un servei	
	2. El sistema rep la petició i genera el llistat de continguts que estan disponibles per la instància de servei que s'ha d'executar, i realitza la comprovació amb les dades que ho indiquen a la base de dades.
3. Es mostren els continguts disponibles pel servei accedit, l'usuari clica per accedir a la opció desitjada	
Curs alternatiu	
No hi ha	

Canviar idioma aplicació

Actor	Usuari
Entrada	Idioma de l'usuari
Sortida	-
Resum	L'usuari canvia a les seves preferències l'idioma amb que es mostra l'aplicació

Curs típic d'esdeveniments	
Actor	Sistema
1. L'usuari accedeix a la seves preferències, canvia l'idioma que està desat i envia les dades	
	2. El sistema guarda les dades al perfil de l'usuari, i al generar els continguts de l'aplicació per ser visualitzats comprova l'idioma de l'usuari i els genera en l'idioma corresponent.
3. Es torna a mostrar el perfil en l'idioma que s'ha seleccionat.	
Curs alternatiu	
1. L'usuari canvi l'idioma fora de l'aplicació, en el login, o la cerca d'aules	
	2. El servidor rep com a paràmetre l'idioma amb la petició i canvia l'idioma en que es mostren les pàgines.
3. La pàgina corresponent canvi d'idioma.	

Canvi d'idioma d'un servei

Actor	Usuari
Entrada	Idioma
Sortida	-
Resum	L'usuari canvi l'idioma en el que es visualitza el servei i els continguts del servei

Curs típic d'esdeveniments	
Actor	Sistema
1. L'usuari accedeix a una opció dels continguts del servei i canvia l'idioma en la interfície del servei	
	2. El servidor rep una petició per part del servei demanant la informació de les dades amb els idiomes accessibles. No es canvia l'idioma de l'aplicació.
3. Els continguts canvien d'idioma així com la interfície del servei. La resta de l'aplicació no es veu afectada per la petició.	
Curs alternatiu	
1. L'usuari selecciona el mateix idioma, no es produeix cap petició.	

Afegir serveis

Actor	Professor
Entrada	Títol, captcha + paràmetres particulars de cada servei
Sortida	-
Resum	S'afegeix una nova instància de servei accessible a l'aula

Curs típic d'esdeveniments	
Actor	Sistema
1. L'usuari accedeix a la pàgina per seleccionar quin servei vol afegir	
	2. Al rebre la petició d'accedir a la pàgina d'afegir un servei genera un llistat dels diferents tipus de serveis que hi han disponibles per afegir a l'aplicació, si un servei conté diferents subtipus afegeix els subtipus al llistat.
3. Es mostra un llistat de serveis disponibles per afegir a l'aula. Es selecciona un per afegir.	
	4. Es rep el tipus de servei a introduir, segons el tipus o subtipus de servei que sigui es carreguen les opcions que hi hagin disponibles per aquell i es configura la vista i els elements depenent d'aquest tipus o subtipus.
5. Es mostra un formulari amb els camps i les opcions oferides pel tipus de servei seleccionat. S'introdueixen les dades del formulari i s'envia.	
	6. Es reben les dades del formulari i es validen correctament per tal de crear i enregistrar una nova instància del tipus/subtipus de servei escollit i s'afegeix a l'aula quedant accessible a través d'ella.
7. Es torna a la pàgina principal de l'aula, podent-se observar i accedir en el menú de serveis la nova instància creada.	
Curs alternatiu	
	6. Les dades no es validen correctament i s'envia un missatge d'error amb el formulari.
7. Es mostra un missatge d'error en la zona de missatges corresponent del formulari.	

Introduir imatges

Actor	Professor
Entrada	Imatge
Sortida	-
Resum	Un professor introdueix un fitxer en els continguts d'un servei de document.

Curs típic d'esdeveniments	
Actor	Sistema
1. Es clica en la opció de l'editor de text introduir una imatge.	
	2. El servidor rep la petició i la informació i configuració relacionada amb les carpetes que contenen imatges al servidor per seleccionar i on pujar de noves.
2. Es mostra una finestra on pot seleccionar una imatge que l'usuari vol afegir de les seves.	
	3. Es rep un fitxer que es puja al servidor i es retorna la adreça per poder ser visualitzada en l'editor.
4. Es mostra un missatge de fitxer pujat, s'accepta i es pot veure la imatge introduïda dins l'editor. S'envien les dades del servei.	
	5. Es crea la instància com en els altres casos i s'afegeix a l'aula.
6. Dins del servei document es pot visualitzar la imatge introduïda per l'usuari.	

Curs alternatiu	
	3. El servidor rep el fitxer que no compleix amb els requisits de grandària o formats vàlids i envia un error.
4. Es mostra un missatge d'error en la zona de missatges i es pot tornar a seleccionar un altre fitxer.	

Edició etiquetes

Actor	Professor
Entrada	
Sortida	-
Resum	S'introdueix dins d'una instància de servei etiquetes visibles

Curs típic d'esdeveniments	
Actor	Sistema
1. Dins del servei l'usuari clica en la interfície per afegir etiquetes, es mostra un formulari amb les dades a introduir.	
	2. Es rep una petició amb les dades del formulari, es validen i es comprova que l'usuari tingui els credencials necessaris. S'enregistra a la base de dades les etiquetes i s'envia un missatge de validesa de petició.

3. Es mostra en el servei l'etiqueta introduïda.	
Curs alternatiu	
	2. Es una petició no vàlida, es retorna un missatge d'error.
3. Es mostra el missatge d'error en l'editor.	

6.2.3 Casos d'ús de Gestió d'aules

Aquests casos d'ús són els es faciliten als professors per poder realitzar les tasques pròpies de gestió de les aules dins d'un entorn de gestió englobat per una escola. També entren dins dels sistemes i mecanismes que s'utilitzen per gestionar i controlar l'accés a les aules pels usuaris sense els quals no tindria sentit el sistema basat en privilegis per les aules i serveis.

Crear aules

Actor	Professor/Administrador
Entrada	Títol, subtítol, password, captcha
Sortida	-
Resum	Un usuari amb permisos omple les dades per crear una aula i poder accedir a ella.
Curs típic d'esdeveniments	
Actor	Sistema
1. Es mostra un formulari on l'usuari introdueix les dades i les envia	
	2. El sistema valida les dades introduïdes, enregistra la nova instància de l'aula, i retorna el servei de la nova aula.
3. Es redirigit a la pàgina principal de la nova aula creada.	
Curs alternatiu	
	2. El sistema valida les dades però no són correctes, envia un missatge d'error a l'usuari.
3. Es mostra en la zona d'errors del formulari l'error que s'ha produït en la validació de les dades del formulari.	

Editar clau de l'aula

Actor	Professor
Entrada	Clau antiga, Clau nova
Sortida	-
Resum	Es canvia la clau d'accés a l'aula
Curs típic d'esdeveniments	
Actor	Sistema
1. Es mostra un formulari on introduir clau antiga i la clau nova de l'aula.	
	2. Es reben les claus, es valida la clau antiga, i el format de la nova, en cas afirmatiu es substitueix la clau de l'aula per la nova.
3. Es retorna a la pàgina d'edició de dades de l'aula.	
Curs alternatiu	
	2. Les claus no són vàlides, s'envia un error.
3. Es mostra el formulari amb el missatge d'error en la zona corresponent.	

Accés a gestió d'aules

Actor	Professor
Entrada	-
Sortida	-
Resum	Accés a l'entorn per la gestió de les aules que ha creat
Curs típic d'esdeveniments	
Actor	Sistema
1. Es selecciona en el menú superior la opció gestió.	
	2. Es rep la petició, es comproven els credencials per mostrar les opcions de gestió d'aules pròpies.
3. Es mostra un menú amb les opcions per gestionar les aules del professor.	
Curs alternatiu	
No hi ha	

Llistar aules pròpies

Actor	Professor
Entrada	-
Sortida	Llistat d'aules(Títol d'aula + llista de permisos(usuari + permís))
Resum	Es mostra un llistat de les aules creades amb els permisos donats per cada una.
Curs típic d'esdeveniments	
Actor	Sistema
1. L'usuari accedeix a la opció dins de gestió d'aules de les seves aules.	
	2. Es realitza una consulta a la base de dades per obtenir les aules les quals el seu autor és l'usuari que demana la petició, i per cada una de les aules s'obtenen tots els permisos que hi ha a la base de dades per aquell servei.

3. Es mostra en una taula un llistat de les aules del professor, per cada de les aules es pot obrir seleccionant una fletxa un llistat dels usuaris als que se li han donat permisos dins la mateixa taula. Per cada usuari es presenten funcions de edició de permisos, i per les aules es permet activar/desactivar i eliminar cada una. També es mostra un botó per donar nous permisos a usuaris per les aules llistades.	
Curs alternatiu	
No hi ha	

Activar/desactivar aula

Actor	Professor
Entrada	Títol aula
Sortida	-
Resum	Activar o desactivar l'aula seleccionada permetent o no l'accés

Curs típic d'esdeveniments	
Actor	Sistema
1. Es selecciona la opció Activar o Desactivar en el llistat d'aules del professor o de les aules de l'escola.	
	2. Es rep la petició, i es modifica a la base de dades l'activació del servei.
3. S'actualitza el llistat d'aules canviant la opció a realitzar d'activar a desactivar o a la inversa.	
Curs alternatiu	
No hi ha	

Eliminar aula

Actor	Professor
Entrada	Títol aula
Sortida	-
Resum	S'elimina de l'aplicació l'aula seleccionada

Curs típic d'esdeveniments	
Actor	Sistema
1. Es selecciona la opció d'eliminar l'aula en el llistat d'aules.	
	2. S'esborra de la base de dades el servei de l'aula i la informació relacionada com els serveis que contenia i els permisos que hi havien assignats a altres usuaris per l'aula.
3. S'actualitza el llistat d'aules traient l'aula eliminada.	
Curs alternatiu	
No hi ha	

Donar permís d'accés a aula pròpia

Actor	Professor
Entrada	Correu, títol d'aula, permís
Sortida	Llista d'aules
Resum	Es dona un permís per una aula a un usuari que no en tenia.

Curs típic d'esdeveniments	
Actor	Sistema
1. Es selecciona donar permís en el llistat de les aules pròpies i es mostra un formulari en el qual es selecciona una de les aules disponibles i un permís dels disponibles al formulari, s'introdueix el correu d'un usuari de l'aplicació i s'envia.	
	2. Es rep la petició, es comproven les dades del formulari i s'enregistra a la base de dades el permís, el servei i l'usuari. S'envia un missatge de validesa.
3. El llistat d'aules s'actualitza afegint a la llista existent l'entrada pel permís donat a l'usuari dins l'aula seleccionada.	
Curs alternatiu	
	2. Si el permís ja existeix o l'usuari no existeix es retorna un missatge d'error.
3. Es mostra un missatge d'error al formulari.	

Editar permís d'accés a una aula pròpia

Actor	Professor
Entrada	Permís,usuari, aula
Sortida	Llistat d'aules
Resum	Es modifica el permís que té assignat un usuari per una aula del professor.

Curs típic d'esdeveniments	
Actor	Sistema
1. Es selecciona en el llistat la opció d'edició de permís i es mostra un formulari per canviar el permís d'accés, es poden donar permisos més o menys restrictius, o treure el permís assignat a l'usuari.	
	2. Es rep la petició i s'enregistren els canvis en la entrada corresponent de la base de dades pel permís seleccionat.
3. El llistat d'aules s'actualitza mostrant el permís modificat o sense ell si s'ha eliminat.	
Curs alternatiu	
No hi ha	

6.2.4 Casos d'ús de Gestió d'escola

Aquests casos d'ús són els que proporcionen les funcionalitats que formen l'entorn oferit als administradors de cada escola per poder gestionar els centres als que pertanyen. Dins dels casos d'ús que es descriuen s'ha d'afegir alguns descrits anteriorment com la activació, desactivació i eliminació d'aules, que també hi són presents en el llistat d'aules de l'escola, però que comparteixen els mateixos funcionaments i per aquest motiu no es tornen a especificar.

Accés a gestió d'escola

Actor	Administrador
Entrada	-
Sortida	-
Resum	Accés a l'entorn per la gestió de les aules que es troben dins d'una escola

Curs típic d'esdeveniments	
Actor	Sistema
1. Es selecciona en el menú superior la opció gestió.	
	2. Es rep la petició, es comproven els credencials per mostrar les opcions de gestió d'escola si es tracta d'un administrador.
3. Es mostra un menú i una benvinguda a l'entorn amb les opcions per gestionar les aules de tot el centre, els usuaris amb permisos i la informació relacionada amb l'escola.	

Curs alternatiu
No hi ha

Llistar aules d'escola

Actor	Professor
Entrada	-
Sortida	Llistat d'aules(Títol d'aula + subtítol + autor + estat))
Resum	Es mostra un llistat de les aules creades amb els permisos donats per cada una.

Curs típic d'esdeveniments	
Actor	Sistema
1. L'usuari accedeix a la opció dins de gestió d'escola d'aules del centre.	
	2. Es realitza una consulta a la base de dades per obtenir les aules les quals el seu servei pare és l'escola dins la que s'està.
3. Es mostra en una taula un llistat de les aules del professor que estan dins de l'escola, activar/desactivar i eliminar cada una.	

Curs alternatiu
No hi ha

Llistar professors

Actor	Administrador
Entrada	
Sortida	Llistat (usuari + nom)
Resum	Es mostren els professors de l'escola

Curs típic d'esdeveniments	
Actor	Sistema
1. Es selecciona en el menú de gestió de l'escola 'Professors'	
	2. Es realitza una consulta per obtenir els usuaris que tenen permisos d'escriptura al servei de l'escola i es retorna el llistat obtingut.
3. Es mostra un llistat dels professors de l'escola amb opció d'edició del permís per cadascun, i un botó per introduir nous professors a l'escola.	

Curs alternatiu
No hi ha

Afegir professor

Actor	Administrador
Entrada	Correu d'usuari
Sortida	Llistat
Resum	S'introdueix un professor a l'escola

Curs típic d'esdeveniments	
Actor	Sistema
1. Dins del llistat de professors es selecciona afegir un professor, es mostra un formulari on s'introdueix el correu d'un usuari de l'aplicació.	
	2. Es rep la petició, es comproven que l'usuari existeixi i que no tingui permís i s'enregistra el permís d'escriptura pel servei d'escola amb l'usuari rebut.
3. S'actualitza el llistat de professors i s'afegeix l'usuari introduït.	

Curs alternatiu	
	2. Es rep la petició amb un usuari que ja té permís o que no existeix, es retorna un missatge amb l'error.
3. Es mostra l'error corresponent en la zona de missatges del formulari.	

Treure professor

Actor	Administrador
Entrada	Correu d'usuari
Sortida	Llistat
Resum	S'elimina un professor de l'escola

Curs típic d'esdeveniments	
Actor	Sistema
1. Dins del llistat de professors es selecciona l'opció de treure un professor de la taula.	
	2. Es rep la petició, i s'elimina de la base de dades el permís d'escriptura pel servei d'escola amb l'usuari rebut.
3. S'actualitza el llistat de professors i desapareix l'usuari que s'ha seleccionat	

Curs alternatiu
No hi ha

Llistar administradors

Actor	Administrador
Entrada	
Sortida	Llistat (usuari + nom)
Resum	Es mostren els administradors de l'escola

Curs típic d'esdeveniments	
Actor	Sistema
1. Es selecciona en el menú de gestió de l'escola 'Administradors'	
	2. Es realitza una consulta per obtenir els usuaris que tenen permisos d'administració al servei de l'escola i es retorna el llistat obtingut.
3. Es mostra un llistat dels administradors de l'escola amb opció d'edició del permís per cadascun, i un botó per introduir nous professors a l'escola.	

Curs alternatiu
No hi ha

Afegir administrador

Actor	Administrador
Entrada	Correu d'usuari
Sortida	Llistat
Resum	S'introdueix un administrador a l'escola

Curs típic d'esdeveniments	
Actor	Sistema
1. Dins del llistat d'administradors es selecciona afegir un administrador, es mostra un formulari on s'introdueix el correu d'un usuari de l'aplicació.	
	2. Es rep la petició, es comproven que l'usuari existeixi i que no tingui permís i s'enregistra el permís d'administració pel servei d'escola amb l'usuari rebut.

3. S'actualitza el llistat d'administradors i s'afegeix l'usuari introduït.	
Curs alternatiu	
	2. Es rep la petició amb un usuari que ja té permís o que no existeix, es retorna un missatge amb l'error.
3. Es mostra l'error corresponent en la zona de missatges del formulari.	

Treure administrador

Actor	Administrador
Entrada	Correu d'usuari
Sortida	Llistat
Resum	S'elimina un administrador de l'escola
Curs típic d'esdeveniments	
Actor	Sistema
1. Dins del llistat d'administradors es selecciona l'opció de treure un administrador de la taula.	
	2. Es rep la petició, i s'elimina de la base de dades el permís d'administració pel servei d'escola amb l'usuari rebut.
3. S'actualitza el llistat d'administradors i despareix l'usuari que s'ha seleccionat	
Curs alternatiu	
No hi ha	

Editar dades escola

Actor	Administrador
Entrada	
Sortida	-
Resum	S'editen les dades del perfil de l'escola
Curs típic d'esdeveniments	
Actor	Sistema
1. Es selecciona en el menú de gestió d'escola la opció Editar Escola.	
	2. S'obtenen les dades de l'escola que estan relacionades amb el servei en el que s'està i es crea un formulari amb elles introduïdes.
3. Es mostra el formulari de l'escola amb les dades que hi ha guardades a la base de dades. S'edita la informació pertinent i s'envia el formulari.	
	4. Es rep el formulari i es valida, la informació del formulari s'actualitza a la base de dades.
5. Es redirigeix a la benvinguda de la gestió de l'escola.	
Curs alternatiu	
No hi ha	

Accés ajuda

Actor	Usuari
Entrada	-
Sortida	-
Resum	L'usuari accedeix als elements d'ajuda de l'aplicació

Curs típic d'esdeveniments	
Actor	Sistema
1. L'usuari accedeix al menú d'ajuda de l'aplica	
	2. S'envia a l'usuari a la opció escollida
Curs alternatiu	
No hi ha	

Editar permís d'accés d'admin/professor

Actor	Administrador
Entrada	Permís,usuari
Sortida	Llistat d'usuaris
Resum	Es modifica el permís que té assignat un usuari per l'escola

Curs típic d'esdeveniments	
Actor	Sistema
1. Es selecciona en el llistat l'usuari que es vol editar i es mostra un formulari per canviar el permís d'accés, es poden donar permisos més o menys restrictius.	
	2. Es rep la petició i s'enregistren els canvis en la entrada corresponent de la base de dades identificada per l'usuari, servei i permís antic.
3. El llistat d'aules s'actualitza traient del llistat a l'usuari que se li ha modificat el permís.	
Curs alternatiu	
	2. El permís ja existeix i es retorna un missatge d'error
3. Es mostra el missatge d'error en la zona de missatges del formulari.	

7 Model de dades

Al tractar-se d'un projecte realitzar a partir d'una aplicació ja existent el model de dades ja havia sigut dissenyat i estava implementat en la aplicació. En les bases de dades de l'aplicació estava reflectit també el model, tot i que s'han realitzat canvis durant la realització del projecte la major part de l'esquema, les classes que s'utilitzen, provenen del disseny original amb algunes adaptacions en el seu interior.

No obstant hi han hagut dos aportacions i canvis al model durant la realització del projecte, tot i aquesta part no es reflecteix en el disseny de la base de dades, que ha sigut la que ha preservat gran part dels dissenys originals. Però dins l'aplicació s'ha hagut d'introduir estructures de dades i classes al model que abans no estaven contemplades tot i que si que es trobava part de la informació a la base de dades.

Hi ha un altre conjunt de classes que s'utilitzen dins l'aplicació i que degut a canvis en el funcionament intern s'ha extret del seu interior. Aquesta informació que en l'actualitat ha augmentat la seva quantitat de dades, dins del codi de l'aplicació es trobava com a informació estàtica i d'ella depenen l'execució dels serveis.

Degut a l'ús d'un model ja existent a l'aplicació i no haver de realitzar grans canvis als dissenys de les bases de dades, com al fet d'utilitzar durant el projecte metodologies de desenvolupament ràpid d'aplicacions web, no es mostraran els diagrames de classes típics, es farà ús en el seu lloc dels esquemes dels models que s'utilitzen a Symfony per descriure i explicar el model de dades i les classes que s'han fet servir.

Com ja he remarcat la majoria del model ja estava implementat, el motiu de la seva exposició i descripció es perquè es pugui tenir una comprensió de l'àmbit en el que funciona l'aplicació i el conjunt de les dades amb les que he hagut de familiaritzar-me i tractar. Cal destacar que no només he hagut de tenir una visió de com funcionava el model de classes, sinó que durant el desenvolupament he hagut de tractar i afegir noves funcionalitats en moltes de les classes derivades dels models per poder obtenir les funcionalitats desitjades i fer un bon ús de la orientació a objectes, motiu de més per comentar el model de classes.

7.1 Classes de l'aplicació global

Usuari

Conté les dades del perfil de l'usuari tals com el seu nom i cognom, data de naixement, ciutat,... les dades que hi han la majoria són informatives, però s'ha de destacar web_lang que és el camp que conté l'idioma en el que es visualitzarà l'aplicació. El nom i cognom s'utilitza en algun llistat de permisos dels usuaris per poder identificar-los. Com en altres classes es pot observar un identificador de servei, aquest fa referència a la instància de la classe servei que el representa com a servei dins l'aplicació. Una altre referència és l'identificador de sf_guard_user, el qual és la classe de l'usuari pròpiament dita que veurem més endavant.

```
user:
  _attributes:          { phpName: User }
  id:
  sf_guard_user_id:     { type: integer, foreignTable: sf_guard_user,
                          foreignReference: id, required: true, onDelete: cascade }
  service_id:           { type: integer, foreignTable: service,
                          foreignReference: id, required: true, onDelete: cascade }
  first_name:           varchar(20)
  last_name:            varchar(20)
  native_lang:          { type: varchar(5) }
  web_lang:             { type: varchar(5) }
  birthday:             date
  country:              { type: varchar(2) }
  postalcode:           varchar(8)
  address:              varchar(255)
  city:                 varchar(255)
  region:               varchar(255)
  image:                varchar(255)
  about:                longvarchar
```

Comunitat

La classe que conté les dades d'una aula. En aquesta classe també hi trobem l'identificador del servei que representa. Conté altres dades pròpies de l'aula com el subtítol, i la clau perquè els usuaris anònims puguin accedir a l'aula.

```
community:
  id:
  service_id:           { type: integer, foreignTable: service,
                          foreignReference: id, required: true, onDelete: cascade }
  default_subservice:   { type: integer, foreignTable: service,
                          foreignReference: id, required: false, onDelete: cascade }
  subtitle:             varchar(255)
  password:             { type: varchar, size: 128 }
  algorithm:            { type: varchar, size: 128, required: true, default: sha1 }
  salt:                 { type: varchar, size: 128, required: true }
```

Entitat

La classe entitat representa el perfil d'una escola. Al igual que les altres classes conté informació relacionada amb el registre de l'escola.

```
entity:
  id:
  service_id:           { type: integer, foreignTable: service,
                          foreignReference: id, required: true, onDelete:cascade }
  entity_is_active:     { type: boolean, required: true, default: 0 }
  name:                 { type: varchar(255), required: true }
  president_name:       varchar(40)
  president_user:       { type: integer, foreignTable: sf_guard_user,
                          foreignReference: id, required: false }
  registry_code:        { type: varchar(12), required: true }
  registered_at:        { type: varchar(40), required: true }
  addressed_to:         varchar(60)
  web:                  varchar(255)
  country:              varchar(2)
  postalcode:           varchar(8)
  address:              varchar(255)
  city:                 varchar(255)
  region:               varchar(255)
  phone_number:         varchar(20)
  fax_number:           varchar(20)
  entity_email:         varchar(255)
  logo:                 varchar(255)
  about:                longvarchar
```

Plain

Aquesta classe representa un servei document. Es tracta de continguts introduïts a través d'un editor de text en el que es poden introduir altres elements com imatges i guardar-lo a la base de dades. Aquests continguts són accessibles a través del servei al que fa referència amb l'identificador corresponent dins de l'aplicació.

```
plain:
  id:
  service_id:           { type: integer, foreignTable: service,
                          foreignReference: id, required: true, onDelete: cascade }
  content:              longvarchar
```

Mapexpl

La classe que representa als serveis de zooms i rotacions. Conté informació per determinar quins són els continguts que seran accessibles, aquesta informació s'utilitzarà en l'aplicació per determinar si un teixit del cervell està disponible o no,

es guarda un valor al que després s'aplicaran màscares per determinar les seves opcions. La informació i les classes relacionades amb els continguts i la forma en la que es presenten a l'usuari a l'hora d'accedir es presentaran més endavant.

A diferència d'altres serveis aquest es classifica en diferents subtipus, aquesta necessitat va aparèixer durant el projecte perquè s'ofereixen diferents continguts presentats en forma de serveis diferents, però que en realitat internament utilitzen els mateixos mecanismes, l'única diferencia són els continguts que es poden mostrar entre els diferents subtipus de servei i la forma en que es presenten a l'usuari.

```
mapexpl:
  id:
  service_id:           { type: integer, foreignTable: service,
                          foreignReference: id, required: true, onDelete: cascade }
  availability_low:     integer
  availability_high:    integer
  type:                 varchar(20)
```

Bones

Aquesta classe és molt similar a l'anterior, defineix un servei de tres dimensions del cos humà. La informació que conté és la mateixa que a Mapexpl, a excepció del subtipus, ja que mapexpl és l'única que fa aquesta diferenciació.

```
bones:
  id:
  service_id:           { type: integer, foreignTable: service,
                          foreignReference: id, required: true, onDelete: cascade }
  availability_low:     integer
  availability_high:    integer
```

7.2 Model del plugin sfGuard

SfGuardPermission

La classe representa un tipus de credencial dins l'aplicació. El conjunt de permisos disponibles dins l'aplicació estan definits com a instàncies d'aquesta classe, si es vol afegir o treure algun tipus de credencial pel seu ús dins l'aplicació ha d'estar representat en aquesta classe.

```
sf_guard_permission:
  _attributes:    { phpName: sfGuardPermission }
  id:             ~
  name:           { type: varchar, size: 255, required: true, index: unique }
  description:    { type: longvarchar }
```

SfGuardUser

Aquesta classe representa a l'usuari i és la que s'utilitza per realitzar totes les accions que impliquen a l'usuari. Conté les dades essencials de l'usuari i que l'identifiquen: usuari i contrasenya, i per accedir al perfil que en aquest cas es la classe User que hem descrit, s'accedeix a través d'ella amb els mètodes corresponents.

Gràcies aquesta classe estan implementats els credencials, com es pot veure més endavant els permisos es relacionen amb sfGuardUser, ja que és la classe que proveeix del conjunt d'accions i mètodes relacionats amb la seguretat on es realitzen les comprovacions. Amb la introducció dels serveis però, aquest comportament ha sigut estès i molts cops recau en ella la comprovació dels permisos.

```
sf_guard_user:
  _attributes:    { phpName: sfGuardUser }
  id:             ~
  username:       { type: varchar, size: 128, required: true, index: unique }
  algorithm:      { type: varchar, size: 128, required: true, default: sha1 }
  salt:           { type: varchar, size: 128, required: true }
  password:       { type: varchar, size: 128, required: true }
  created_at:     ~
  last_login:     { type: timestamp }
  is_active:      { type: boolean, required: true, default: 1 }
  is_super_admin: { type: boolean, required: true, default: 0 }
```

SfGuardRememberKey

Proporciona la possibilitat de recordar l'accés d'un usuari. Aquesta és una altre funcionalitat que proporciona el plugin sfGuard, es combina introduint als formularis que es facin servir per realitzar els logins el camp necessari per poder activar aquest aspecte per un usuari.

```
sf_guard_remember_key:
  _attributes:    { phpName: sfGuardRememberKey }
  user_id:        { type: integer, primaryKey: true, required: true,
                    foreignTable: sf_guard_user, foreignReference: id, onDelete: cascade }
  remember_key:   { type: varchar, size: 32 }
  ip_address:     { type: varchar, size: 50, primaryKey: true }
  created_at:     ~
```

7.3 Model del plugin gpServices

Service

La classe representant dels serveis. Sota aquesta classe cau la majoria del pes del control de la correcta execució dels serveis de l'aplicació. Proporciona una interfície per abstraure els serveis i permet que l'accés a ells i la seva administració sigui més homogènia i s'integrin dins dels mecanismes de seguretat de l'aplicació totes les aplicacions que recull.

Els serveis contenen un identificador numèric, per identificar-los inequívocament. El títol és un paràmetre que es fa servir en tots els serveis, és el nom que es mostra als usuaris quan es llisten els serveis disponibles d'una aula. El paràmetre és el camp que s'utilitza per crear l'adreça del servei, perquè no es produeixi cap error existeix la restricció que dos serveis no poden tenir el mateix paràmetre dins del mateix servei pare, que en aquest cas és l'escola. D'aquesta manera els paràmetres són els que identifiquen a un servei en l'aplicació i la seva adreça és el resultat de la concatenació dels paràmetres dels serveis dins dels que es troba més el seu.

La resta de dades afecta també al comportament de l'aplicació, l'autor identificar l'usuari que ha creat el servei, i s'utilitzarà per comprovar els permisos, ja que l'autor té tots els permisos sobre els seus serveis, i també es pot utilitzar per trobar i accedir al serveis que ha creat un usuari i d'aquesta forma poder gestionar-los. El camp parent determina quin és el servei pare sota el qual s'executa, d'aquesta manera es pot recórrer els serveis fàcilment i realitzar comprovacions o generar la seva adreça.

```
service:
  #Unique parameter for a parent service or, if has not parent, for a supported_service_id
  id:
  service_type_name:    { type: varchar(32), required: true }
  parameter:            { type: varchar(32), required: true }
  title:                { type: varchar(75), required: true }
  author:               { type: integer, foreignTable: sf_guard_user, foreignReference: id, required: true }
  visits:               bigint
  created_at:
  last_visit_at:        timestamp
  is_active:            { type: boolean, required: true, default: 1 }
  parent:               { type: integer, foreignTable: service, foreignReference: id }
```

UserPermissionService

Estableix relacions entre permisos, usuaris i serveis. Quan es vol donar algun permís d'accés a un usuari (que no sigui anònim) es crea una instància amb els identificadors que referencien a les corresponents instàncies de les classes involucrades. No només s'utilitzen per controlar l'accés als serveis, també poden determinar a quin servei accedirà l'usuari quan es entri a l'aplicació, i en la cerca d'usuaris amb permisos que hagin sigut donats en els llistats de gestió d'aules.

```
user_permission_service:
  sf_guard_user_id:       { type: integer, foreignTable: sf_guard_user,
                            foreignReference: id, required: true }
  sf_guard_permission_id: { type: integer, foreignTable: sf_guard_permission,
                            foreignReference: id, required:true }
  service_id:
  _uniques:
    index:   [sf_guard_user_id, sf_guard_permission_id, service_id]
```

7.4 Model d'etiquetes

Hi han alguns serveis, els de les classes Mapexpl i Bones, que ofereixen la possibilitat de mostrar etiquetes amb informació en el seu interior, dins les imatges que es veuen als serveis. El que s'ha creat i introduït dins l'aplicació durant el projecte és el concepte de classe per la informació que s'utilitza i està relacionada amb les etiquetes.

Per tractar la informació d'aquesta forma s'han creat un conjunt d'accions i de funcionalitats en l'interior de l'aplicació que han integrat i renovat tota la lògica que hi ha al darrere de les interaccions i que es realitzen entre les aplicacions que s'executen al cridar als serveis i el servidor de l'aplicació, que fins ara treballaven en entorns diferents amb uns models de classes aïllats i que eren agnòstics entre ells.

Ara tota la lògica de les tres capes es basa en la utilització sota el framework de les classes que es presentaran a continuació.

Photolabel

És la representació d'una part de les etiquetes que es fan servir en serveis de Zooms. Contenen les dades comunes i que no varien en una mateixa etiqueta per tal de poder fer una separació amb els continguts que es mostren i que poden ser diversos.

El camp line_id l'explicarem més endavant, però es tracta d'un dels elements que identifiquen i relaciona les etiquetes amb una de les opcions accessibles des dels serveis.

```
photolabel:
  _attributes: { idMethod: native }
  id: { type: INTEGER, required: true, autoIncrement: true, primaryKey: true }
  line_id: { type: VARCHAR, size: '25', required: true }
  tag_x: { type: SMALLINT, required: true }
  tag_y: { type: SMALLINT, required: true }
  scale: { type: FLOAT, required: true }
  ap_min: { type: FLOAT, required: true }
  ap_max: { type: FLOAT, required: true }
  user_id: { type: INTEGER, required: true, foreignTable: service, foreignReference: id, onDelete: cascade }
```

Photolabelrang

És la classe que guarda els continguts d'una etiqueta de Zooms, complementa a Photolabel. La informació que es guarda típicament són cadenes de text en un idioma per cada instància. Aquestes dades es troben separades perquè una etiqueta pot haver-se introduït en diversos idiomes, per tant per cada instància de Photolabel poden haver tantes d'aquesta classe com idiomes siguin suportats a l'aplicació.

```
photolabelrlang:
  photolabelid: { type: INTEGER, required: true, primaryKey: true }
  langid: { type: TINYINT, required: true, primaryKey: true, foreignTable: lang, foreignReference: id}
  name: { type: VARCHAR, size: '50', required: true }
  description: { type: LONGVARCHAR }
```

Rotationlabel

Aquesta classe es en part equivalent a Photolabel, conté la informació bàsica de les etiquetes que s'utilitzen en els serveis de rotacions, i és informació que no s'ha de repetir en les altres classes per a cada instància de Rotationlabel, ja que el contingut no canvia.

```
rotationlabel:
  _attributes: { idMethod: native }
  id: { type: INTEGER, required: true, autoIncrement: true, primaryKey: true }
  line_id: { type: VARCHAR, size: '25', required: true }
  user_id: { type: INTEGER, required: true, foreignTable: service, foreignReference: id, onDelete: cascade }
```

Rotationlabelrlang

És idèntica en comportament i atributs a la classe Photolabelrlang.

```
rotationlabelrlang:
  rotationlabelid: { type: INTEGER, required: true, primaryKey: true }
  langid: { type: TINYINT, required: true, primaryKey: true, foreignTable: lang, foreignReference: id }
  name: { type: VARCHAR, size: '50', required: true }
  description: { type: LONGVARCHAR }
```

Subrotationlabel

A diferència de les etiquetes dels Zooms, en les rotacions per facilitar la visualització quan s'introdueix una etiqueta es fa que aparegui en la imatge anterior i la posterior quan es fa la rotació, per tal que no sigui tan fàcil perdre-la de vista.

Per tant per cada instància de Rotationlabel existeixen tres de Subrotationlabel que contenen els atributs que fan referència a la seva posició en les imatges on apareixeran.

```
subrotationlabel:
  idrotationlabel: { type: INTEGER, required: true, primaryKey: true }
  state: { type: TINYINT, required: true, primaryKey: true }
  tag_x: { type: SMALLINT, required: true }
  tag_y: { type: SMALLINT, required: true }
  scale: { type: TINYINT, required: true }
```

Tresdlabel

És la classe que conté informació de les etiquetes del serveis de tres dimensions. El comportament és el mateix que per la classe Photolabel excepte per l'atribut url que no contenen les altres i que es fa servir per identificar el fitxer de la imatge en el que es visualitzaran les etiquetes.

```
tresdlabel:
  _attributes: { idMethod: native }
  id: { type: INTEGER, required: true, autoIncrement: true, primaryKey: true }
  url: { type: VARCHAR, size: '25', required: true }
  tag_x: { type: SMALLINT, required: true }
  tag_y: { type: SMALLINT, required: true }
  user_id: { type: INTEGER, required: true, foreignTable: service, foreignReference: id, onDelete: cascade }
```

Tresdlabelrlang

És la classe equivalent a Photolabelrlang i Rotationlabelrlang, té els continguts de les etiquetes en cada idioma, i diferents continguts poden apuntar a la mateixa.

```
tresdlabelrlang:
  tresdlabelid: { type: INTEGER, required: true, primaryKey: true }
  langid: { type: TINYINT, required: true, primaryKey: true, foreignTable: lang, foreignReference: id }
  name: { type: VARCHAR, size: '50', required: true }
  description: { type: LONGVARCHAR }
```

Lang

És la classe que identifica els idiomes de les aplicacions, s'utilitza per creuar informació dels idiomes amb les instàncies que proporcionen els continguts de les etiquetes.

```
lang:
  id: { type: TINYINT, required: true, primaryKey: true }
  name: { type: VARCHAR, size: '15', required: true }
  symbol: { type: VARCHAR, size: '5', required: true }
  _uniques: { symbol: [symbol] }
```

7.5 Model derivat de la configuració

Les classes que es descriuran a continuació no es poden trobar als esquemes de l'aplicació, el motiu pel qual no apareixen és perquè les instàncies d'aquestes classes no necessiten d'una persistència a la base de dades, ja que són utilitzades pel funcionament d'alguns mecanismes dels serveis.

Una altra de les característiques que comparteixen aquestes classes és que la informació que contenen no varia i és per aquest motiu que no es troben a la base de dades, per guardar dades la base de dades no és l'única solució, i al no necessitar les propietats de les transaccions i consultes que ens pot aportar aquestes tecnologies es obvi que no resulta ser la millor. Els accessos que es volen realitzar seran de tipus global, obtenint totes les dades, o bé aleatori per obtenir algun element en concret, aquest és un altre motiu més per no utilitzar bases de dades, al no haver de realitzar consultes complexes no s'obtindria una millora de rendiment respecte a realitzar peticions i consultes a un servidor de base de dades.

Al no formar part de l'esquema de base de dades, aquestes classes no depenen ni estan mantingudes per cap dels elements de Symfony, són definicions pròpies i personalitzades que no es veuen afectades per cap canvi que es pugui realitzar dins l'aplicació, ja que dins l'aplicació són aquestes classe i la seva estructura les que són utilitzades per alguns mecanismes i parts de l'aplicació.

Tot i que les classes no estan directament relacionades amb altres que s'utilitzen dins del framewok, pel seu disseny s'ha utilitzat el mateix esquema del model que es genera dins de Symfony i que es fa servir en les classes de l'aplicació. Per tant per cada una de les classes podem trobar la classe objecte per crear i utilitzar les instàncies de la classe, i la classe Peer que s'encarrega d'accedir a les dades per poder crear les instàncies. D'aquesta forma a simple vista no hi ha una diferenciació de la resta del model, i utilitzant la nomenclatura comuna pels mètodes de les classes es proporciona un accés homogeni i estructurat per tota l'aplicació que fan que l'accés a aquestes dades sigui transparent i no s'observi una diferència entre les classes que emmagatzemen a la base de dades i les que no ho fan.

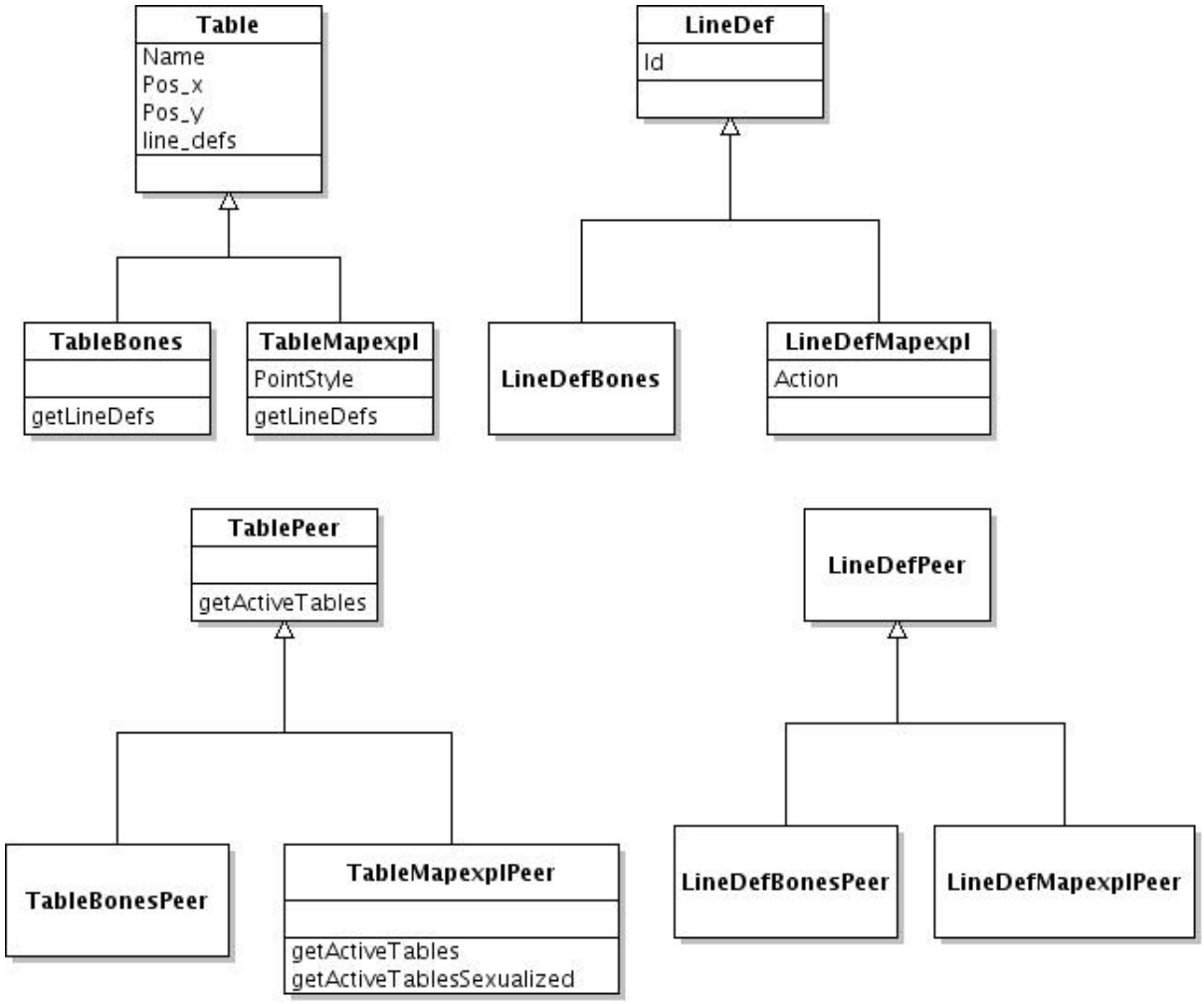
ServiceType

En la classe Service s'ha pogut veure com hi ha un camp que s'anomena com service_type i que conté un string, tot i que no es pot explicitar en l'esquema, el valor d'aquest camp és l'utilitzat com a referència d'elements d'aquesta classe.

ServiceType defineix el tipus de servei i intervé en el mecanisme de gestió de serveis. Conté tota la informació relacionada amb el funcionament d'un tipus de servei en concret. Aquesta informació és determina durant el desenvolupament, i tot i que pot haver algun canvi, un cop s'utilitza un disseny i arquitectura en l'aplicació es poc probable que es realitzin canvis, els canvis que es poden esperar són els de la forma en que s'executaran les accions, els quals no repercuteixen en les dades que reflecteix ServiceType.

Les classes definides a continuació segueixen l'esquema anteriorment comentat, però en aquest cas es presentarà un conjunt interrelacionat. Tots els serveis contenen un conjunt de dades sense els quals no poden executar-se. En alguns serveis a part dels continguts que es poden accedir i modificar, necessiten d'uns conjunts de dades que podrien ser propis de la configuració, ja que configuren les opcions accessibles dins dels serveis.

Aquests serveis presenten una forma similar a l'hora d'accedir a aquestes opcions, és per aquest motiu que s'ha dissenyat una jerarquia de classes per agrupar els comportaments comuns i una part específica per cada tipus de servei degut a les diferències que hi han entre les estructures de dades a les que s'accedeix en cada una de les classes. D'aquesta forma es pot tenir una estructura de dades molt similar per diferents serveis amb una sintaxis homogènia pel seu ús en l'aplicació.



Un dels aspectes a remarcar i que tenen en comú aquestes classes és el seu cicle de vida. En un principi les dades es troben guardades en unes estructures accessibles a través del framework de l'aplicació, els accessos que es produeixin a aquestes dades només derivaran en la creació d'instàncies d'aquestes classes quan sigui necessari el seu ús per part d'algun servei. En el cas que no siguin necessàries s'obviaran, el resultat serà el mateix que quan s'executen consultes per obtenir les instàncies d'altres classes del model.

LineDef

Serveis com Bones i Mapexl basen el seu funcionament en unes línies, aquestes són una serie d'imatges o de localitzacions dins del cos o d'una planta sobre les que els serveis utilitzen les seves funcionalitats. Més endavant es detallarà el funcionament i mecanismes implementats per accedir a ells i veure com afecten als serveis que els fan servir d'hostes i a través dels que s'accedeix.

Cada una d'aquestes línies conté paràmetres diferents, ja que una sèrie pot tenir 25 imatges, i una altre 30, i els serveis que s'executen necessiten saber com poder accedir als recursos i com utilitzar-los per la seva execució. Aquesta classe ofereix el mètodes necessaris per crear les instàncies i passar-les als serveis perquè accedeixin als seus paràmetres quan ho necessitin.

LineDefPeer

Les classes Peer s'utilitzen per obtenir les línies de les seves estructures de dades on es guarden i retornar els objectes corresponents, permeten emular els tipus d'accés abans comentats, obtenint tots els elements que hi hagi per un tipus de línies, o bé especificant a través de l'identificador de la línia una concreta a la que es vol accedir.

Table

Altres dades que utilitzen els serveis són les denominades taules, aquestes classes van ser dissenyades per tal de oferir una manera còmode i dinàmica de generar les opcions que es podran visualitzar i accedir per cada instància de servei. Sabent que poden haver diferents instàncies d'un servei i oferir diferents línies en cadascun, per cada una d'elles s'haurà d'obtenir les seves opcions i mostrar-les com pertoquin a l'usuari.

El concepte de taula prové de la forma en que es presenta l'accés a les diferents opcions dels serveis, cada taula estarà relacionada amb una localització dins les zones a les que es pugui accedir en un servei com pot ser el cap de l'ésser humà. Però aquesta localització no és on s'accedirà, sinó que representa un contenidor on es poden trobar línies disponibles, com pot ser un zoom d'un teixit del cervell, o la rotació d'aquest. Per establir aquesta relació entre localitzacions i línies s'utilitza la classe Table que conté informació relacionada amb el nom a visualitzar, la posició de la localització en la plantilla i a més contindrà un llistat de les línies a les que es

podrà accedir que són necessàries per generar les rutes corresponents a les opcions dels serveis.

TablePeer

Les classes Peer de les taules són les encarregades de desencadenar tot un seguit d'accions que realitzen la feina necessària per obtenir les taules amb les línies disponibles per cada instància del servei. Des del punt de vista de l'interior de l'aplicació les crides a consultes d'aquestes classes són el punt de partida per generar les opcions i poder visualitzar-les.

Cada una de les classe proporciona les seves consultes específiques per tractar amb les classes corresponents de línies que utilitzen els serveis als que estan orientats. Existeix una relació entre les classes Peer de les taules i les classes objecte dels serveis, ja que per realitzar les consultes han d'intervenir les instàncies dels serveis a l'hora de comprovar quines són les opcions que els hi són disponibles, aquesta relació s'estableix a través dels únics camps que poden tenir les classes Peer i que són utilitzats per guardar duran les execucions de les funcions que s'executaran a través de les consultes, les instàncies dels tipus de serveis a les que han d'accedir per fer les comprovacions necessàries.

8 Disseny i Implementació

En l'apartat anterior s'han les tecnologies utilitzades durant la realització del projecte, dins del contingut de la més important de totes, Symfony, s'ha descrit quines han sigut les metodologies de treball i l'arquitectura que defineix Symfony per les aplicacions que es desenvolupen utilitzant-lo.

S'han descrit i tractat els aspectes relacionats amb l'arquitectura, quins són els components i mòduls sobre els que s'erigeix l'aplicació, la relació entre cada un dels elements estructurals de l'aplicació i el framework, com es funcionen i es produiran les comunicacions dins de la mateixa aplicació, i la forma en la que es defineixen la interfície i els procediments que hi han al darrere. Per tant molts aspectes del disseny utilitzat i de la implementació han quedat coberts en aquell capítol. Només queda descriure els mòduls que s'han implementat per complir els requeriments que s'han anat detallant durant el projecte.

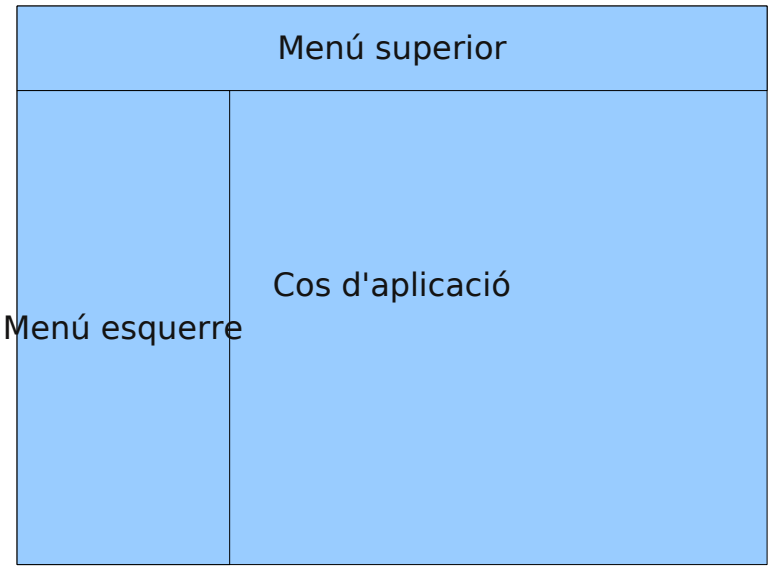
8.1 Implementació

A continuació es presentaran els mòduls i entorns que s'han implementat en l'aplicació. Dins de cada apartat que es descrigui es parlarà sobre els mecanismes i components que interactuen dins del framewok que han sigut implementats, així com les classes del model i estructures de dades addicionals que s'utilitzen i proporcionen part de les funcionalitats que completen l'execució.

En els aspectes que s'hagin realitzat canvis en comptes d'afegir nous components es comparà la implementació anterior amb l'actual per aclarir les diferencies i el motiu que han portat a realitzar-les. Per tal de no fer la lectura més complicada, s'explicaran els aspectes de la implementació sense introduir grans pàgines o blocs de codi, només s'introduiran crides a funcions i noms dels corresponents elements, per veure el codi del que s'està parlant s'afegeixen als annexes amb exemples de funcionalitats completes de codi del que es parlarà.

8.1.1 Estructura de la vista

Respecte a l'estructura de la pàgina, la gran majoria tenen el mateix estil, es troba localitzada en el template de l'aplicació 'layout.php'. Els elements que hi han dins són: menú superior, menú lateral esquerre i el cos de l'aplicació. L'esquema quedaria així:



Menú superior

El menú superior és sempre present a l'interior de l'aplicació, i no canvia el seu contingut sigui quina sigui la situació. Els elements que trobem són els que es poden veure a la imatge.

- Logo aplicació.
- Missatge de salutació a l'usuari.
- Combobox per anar a una aula d'entre les que són disponibles. Per generar les aules disponibles per l'accés a l'usuari entre les opcions del combobox es combina una funció del helper Form de symfony object_select_tag(), a la qual se li passa com a paràmetres la classe, el seu mètode i el camps que volem utilitzar dels resultats de la consulta com a opcions en el desplegable.
- Enllaç a gestió. Aquest enllaç depèn dels permisos que l'usuari tingui assignats, es realitza una crida per saber els permisos que té assignats dins

l'entitat i en cas de tenir-ne es mostra l'enllaç que redirogeix a la pàgina de benvinguda a la gestió.

- L'enllaç al perfil sempre es mostra a l'usuari.
- Menú d'ajuda, conté les opcions: Blog, Instruccions, Sobre GenomEdu i Avisos legals. El menú està controlat per un Javascript que crea uns efectes de desplegament. El mecanisme es basa en dos funcions de jQuery, click() i hover(). Amb aquestes funcions es defineix que quan es produeixi un canvi en l'efecte de desplegament, si està desplegat es contrau, i si no ho està es contrau. La funció hover() està per controlar que si l'usuari surt de la zona del menú aquest es replegui automàticament. Dins del menú el blog i instruccions són enllaços a les pàgines corresponents, però els dos últims enllaços són textos introduïts als templates que mitjançant la funció de jQuery dialog() crea un diàleg i el mostra en el moment en que es clica sobre l'item.
- Enllaç per desconnectar-se de l'aplicació.

Menú lateral

El menú lateral consta d'un slot dins el layout i de dos parcials que s'escolliran segons l'entorn en el que ens trobem, per això els templates d'un entorn introduiran al seu codi el seu corresponent parcial en l'slot del menú. Les opcions són o bé un menú de l'aula o bé el de l'escola.

Menú aula

Si l'usuari es troba dins l'àmbit d'una aula el parcial que s'introdueix dins del slot del layout és el '_communitymenu.php' que conté el menú de l'aula. El menú de l'aula conté tres parts en el seu interior, la primera es visible sempre i es tracta d'un enllaç a l'aula dins la que s'està, d'aquesta manera si un usuari entra en algun dels serveis pot tornar a l'inici de l'aula amb només fer un clic a l'enllaç. D'una altra manera havia de desplaçar-se enrere per poder tornar a l'inici.

El segon element en ordre d'aparició al menú no sempre és visible, només apareix si es té credencials de professor o superior. Aquesta part conté les opcions per editar l'aula: informació de l'aula, activar/desactivar serveis i afegir nous serveis.

L'últim element del menú sempre és visible ja que són un llistat dels serveis que conté l'aula i que es poden accedir siguin quin sigui el rol de l'usuari.

Menú Escola

Si en comptes d'estar dins d'una aula, un usuari professor o administrador es troba dins de l'entorn de gestió que ha pogut accedir a través de l'enllaç del menú superior, el menú lateral canviarà i s'utilitzarà el parcial '_entitymenu.php'.

En la primera part del menú es mostren els enllaços relacionats amb la gestió de l'escola, i només són disponibles si es tenen credencials d'administrador dins l'escola. Els enllaços que mostra són:

- Aules. Mostra el llistat d'aules de l'escola.
- Administradors. Mostra llistat d'administradors de l'escola.
- Professors. Mostra un llistat de professors de l'escola.
- Editar l'Escola. Edició de les dades del perfil de l'escola.
- Inscripcions. En construcció!

En el cas que l'usuari sigui professor, o administrador, ja que té els mateixos privilegis que té un professor però ampliats, es mostraran les opcions del grup Gestió d'Aula, que dona accés a funcionalitats per gestionar les aules que un usuari ha creat dins l'escola. Les opcions disponibles són:

- Aules creades. Proporciona un llistat de les aules i els permisos donats al usuaris sobre elles.
- Afegir aula. Mostra el formulari per crear una aula nova.

Dins dels parcials que defineixen els menús, s'activen les opcions que s'han descrit comprovant que els usuaris que estan dins l'aplicació tinguin els credencials necessaris per realitzar les accions amb la funció hasCredential() que ja s'ha comentat.

Contingut

El contingut de l'aplicació és la part que es va omplint amb les plantilles que generen les accions dels mòduls corresponents als que s'accedeix, i que s'omplen des del layout de l'aplicació a través de la variable que té els continguts generats per les peticions `<?php echo $sf_content ?>`.

8.1.2 Accés a l'aplicació

Login

El login al igual que el layout de l'aplicació ha sofert una gran renovació, els estils que s'han introduït no són propis, sinó que es feina d'un altre membre que va proporcionar un nou estil a l'aplicació. Encara que el disseny dels estils no sigui propi no significa que no hagi portat feina, ja que els disseny venia especificat en plantilles en HTML pur. Els aspectes relacionats amb els blocs dins del document, i els elements del formulari difereixen per tant de com es generen en una plantilla amb PHP i Symfony i ha calgut realitzar una sèrie de modificacions per poder adaptar el nou disseny i fer-lo compatible amb l'aplicació.

En el formulari d'accés anterior, la visualització es feia amb la tècnica de mostrar-lo directament amb una sentencia 'echo' de PHP. Ara en comptes de mostrar-lo directament com el nou disseny defineix una nova estructura d'elements a la vista, s'havia de repartir els camps en diferents blocs HTML i especificar uns atributs específics de cada camp per tal que els estils s'apliquessin correctament. Això va comportar un redisseny del template afegint la nova estructura de la plantilla.

Un altre aspecte que es va haver d'adaptar va ser l'enviament del formulari. En la plantilla rebuda per implementar el nou disseny, el control dels elements del formulari es feien a través de Javascript. Aquesta característica va comportar problemes amb alguns botons que eren controlats amb Javascript per realitzar crides AJAX, però que no servien per fer el SUBMIT del formulari de Symfony al servidor.

També es van haver de canviar les crides AJAX per la interacció dels formularis que proporciona Symfony, i es van simular els efectes que es produïen en cas d'error combinant el propi jQuery que ja s'utilitzava en l'original amb una comprovació dels errors del formulari complementada amb una crida a una funció Javascript que funcionés com a activador de l'efecte de jQuery per mostrar l'error de la mateixa manera.

Login aula

En el cas del login de l'aula es va aprofitar l'esquema de la plantilla i els estils que s'havien utilitzat perquè siguessin el més semblants possibles.

En el cas dels formularis s'està parlant de la part de la vista on es mostren, però no es parla de la part del servidor. El moment en que es creen els formularis dins les accions i les validacions quan es reben enviats pels usuaris són molt semblants en tots, aquest és el motiu pel qual no s'han de descriure, ja que quan es veu el mecanisme d'un formulari típic, s'han vist tots. El procediment resulta el mateix canviant la classe del formulari que s'utilitza i l'objecte que s'enregistra si s'ha de guardar algunes dades.

En el cas del login l'acció, es realitza a través de l'acció signin de sfGuardAuth del plugin sfGuard, com s'ha explicat en l'apart on es descriu el funcionament dels plugins, es poden sobreescriure les accions i templates.

En el nostre cas només hem sobreescrit el template introduint la plantilla personalitzada tal com s'ha descrit. El codi de l'acció no s'ha hagut de re-definir. Aquesta acció s'encarrega d'obtenir la classe que s'utilitza com a formulari dins l'aplicació i que es troba guardada com un paràmetre de l'aplicació dins el fitxer app.yml i que nosaltres tenim definit com a sfGuardFormI18NSignIn.

Cercar aula

Aquesta acció denominada 'search' està ubicada dins del mòdul 'Community', que representa el mòdul de les aules. S'ha introduït dins del mòdul perquè les consultes que es realitzen són sobre serveis del tipus Community, i per tant les funcions que s'executen estan relacionades amb aquesta classe. Com és una acció que no necessita de credencials ni de autenticació d'usuari, ja que es troba accessible des de la pàgina de login a través d'un enllaç de cerca d'aules, no s'ha de realitzar cap comprovació addicional que no tingui a veure amb el formulari dins l'acció.

El formulari que s'utilitza es diu 'CommunitySearchForm', és un formulari molt bàsic que hereta del formulari base de Symfony sfForm i només conté un widget per la entrada de text.

A la plantilla es torna a utilitzar un esquema molt semblant al del login, però en aquest cas s'afegeix un bloc de codi que en cas de que el condicional que comprova si hi ha una variable amb els resultats es compleix, s'activarà l'execució d'aquest codi que procedirà a generar una taula amb el llistat de les aules que s'han rebut com a resultat de la consulta.

Els objectes que es retornen són serveis els quals proveeixen de mètodes ganxo per poder accedir a les funcions de les classes del tipus de servei al que pertanyen. La taula conté:

- Títol de l'aula en forma d'enllaç a la pàgina inicial de l'aula.
- Subtítol de l'aula.
- Nom complet de l'usuari responsable de l'aula que es llista.

Les funcions ganxo que es proporcionen amb els serveis estan implementades perquè l'accés sigui senzill i es pugui realitzar de forma neta cap als paràmetres de les classes relacionades. D'aquesta manera s'evita haver de realitzar un processament de tots els objectes que s'hagin de retornar i crear uns arrays associatius amb els paràmetres des de la capa del controlador.

Un element que podem trobar en diverses taules com la que es fa servir en el llistat de resultats de la cerca d'aules són els paginadors. Per poder proveir taules amb grandàries acceptables i llegibles s'utilitza un objecte paginador que proporciona Propel.

El funcionament és senzill i està designat per utilitzar-se amb classes del model generades amb Propel. Per tal de configurar-lo s'ha d'indicar la classe sobre la que es vol fer les cerques, el número màxim d'elements que apareixeran en una pàgina de la taula i el criteri que es farà servir per realitzar les consultes al model. El paginador va realitzant consultes cada cop que arriba la petició, per defecte en la primera petició no es trobarà amb un paràmetre 'page' que s'envia en les posteriors peticions per navegar a través de les llistes i és el que indica si la petició és nova o bé s'està desplaçant a través de la taula i quina és la següent pàgina a mostrar.

Dins del template només s'hauran de mostrar els indexes del paginador que a més es mostraran com a enllaços a la mateixa acció amb el paràmetre corresponent del número de l'índex al que representa per poder realitzar la navegació a través de la taula.

7.1.3. Mòdul Entitat

En l'anterior versió, l'aplicació no tenia la mateixa noció d'escola que hi ha ara a l'aplicació. Abans només es podia afegir, i no era ni obligatori, una informació relacionada amb una entitat a la que l'usuari podia pertànyer, aquesta informació es

podia introduir i editar des del formulari de l'aula. No tenia cap efecte sobre l'aula o sobre l'usuari, el seu valor era purament informatiu.

Per la nova versió es volia introduir la figura d'escola com a entitat a través de la qual es podrien crear aules, el motiu d'aquest canvi era que es vol proveir d'un entorn per la gestió dels usuaris que pertanyin a una entitat, com podrien ser els professors d'una escola o universitat que decideix adquirir un compte i registrar-se per utilitzar l'aplicació dins la seva escola.

Aquesta nova estructura xocava amb l'anterior, ja que en aquells moments l'aula era el servei arrel per la resta de serveis oferits dins l'aplicació(exceptuant l'usuari que és un cas a part). La part positiva era que el sistema de serveis que es tenia no estava entrellaçat amb els tipus de serveis que hi han disponibles a l'aplicació. Això vol dir que el fet d'afegir un nou tipus de servei al sistema no afectaria al funcionament dels mecanismes en general, les dades que hi havien i que definien els serveis de l'aplicació i la seva informació era el que s'havia de modificar.

Així el primer pas era modificar les dades dels serveis, afegint una entrada per Entitat, definint la informació relacionada amb el seu mòdul, i accions, i quins són els seus serveis fills. Les dades d'altres serveis també es va haver de modificar, ja que en elles es reflecteix la jerarquia que hi ha entre ells. En el cas de la comunitat va haver de passar a ser el servei immediatament inferior a l'entitat.

Aquestes dades van passar a estar definides dins del fitxer servicetypes.yml dins la configuració de l'aplicació. Aquest fitxer junt amb d'altres dels que es parlarà més endavant s'integren dins d'app.yml amb inclusions de PHP, la separació facilita la gestió de les dades i la seva organització sense afectar al fitxer que l'aplicació rebrà i accedirà.

Dins del model de dades es va haver de realitzar canvis en Entity i Community, ja que les aules contenien un camp que apuntava a una entitat en el cas que s'haguessin introduït les dades del formulari. Aquest camp va haver de desaparèixer, ja que la relació entre entitat i aula es passaria a establir a través del camp que identifica el pare d'un servei. Per poder establir les instàncies d'entitat com a serveis també es va haver d'afegir un camp que apuntaria a la instància de servei concreta corresponent.

Per tal d'oferir un entorn amb totes les funcionalitats que es demanaven per la gestió de l'escola es va haver de crear un nou mòdul que les agrupés, ja que els mòduls existents pertanyien cadascun a un tipus de servei, i l'escola al no haver existit com a servei, i només tenir com a propòsit omplir un formulari opcionalment, no en tenia.

Al afegir un entorn nou i noves funcionalitats enfocades a un nou grup d'usuaris es va haver de modificar també l'esquema de credencials que hi havia i que identificaven els rols dels usuaris que hi havien presents a l'aplicació. Fins al moment s'utilitzava sobre les aules els permisos de lectura i escriptura, aquests permisos però no afectaven als serveis com els zooms i rotacions ja que eren externs, dels quals es va derivar un altre problemàtica de permisos que més tard es parlarà.

Com que les accions a realitzar en el nou mòdul eren diferents, ja que no es tractava d'accedir a serveis per afegir contingut o visualitzar-los, es va utilitzar el permís d'escriptura i afegir-ne un permís de nou als que ja hi havien dins del plugin sfGuardPermission per tal de poder diferenciar als professors dels administradors d'una escola.

L'esquema de permisos que es va derivar d'aquestes canvis va acabar en:

- ServiceRead: Utilitzat en aules per dona accés a l'aula i els seus serveis, sense poder modificar continguts.
- ServiceWrite: Utilitzat en les aules i les escoles, pel primer cas defineixen el rol de professor en una aula, i el segon el rol de professor dins d'una escola. Un usuari pot tenir permisos per modificar el contingut dins d'una aula, però no en l'escola. Això implicarà que no podrà crear aules noves ni editar els permisos de les que tingui permís per gestionar-les, ja que no se li permetrà l'accés al mòdul d'escola.
- ServiceAdmin: Utilitzat només dins de l'escola i identifica a un usuari com a administrador d'aquesta.

Com a últim comentari dels efectes de la implementació d'escola com a servei arrel de l'aplicació, s'ha de tenir en compte que aquest fet no pot ser evadit, per tant tot usuari de l'aplicació es trobarà dins d'alguna escola. La solució proposada per poder introduir usuaris i aules que no estiguin lligades a cap escola o entitat educativa és tenir dins l'aplicació unes escoles genèriques en les que agrupar aquests usuaris. En

aquests casos, els usuaris mai rebran permisos d'administradors, ja que es tractarà d'usuaris que només hauran de gestionar les seves pròpies aules.

Accions del mòdul Escola

Editar Escola

L'acció 'edit' del mòdul s'encarrega de mostrar el formulari de la informació de l'escola, encapsula les tasques de creació, validació i enregistrament del formulari. El seu comportament és idèntic als formularis ja vistos, en aquest cas s'utilitza el formulari EntityEditForm que hereta del formulari EntityCreateForm que és el formulari que s'utilitza per crear les escoles i conté tots els camps de les dades introduïdes per una escola. Com ja s'ha descrit en el procediment de creació d'un formulari, al crear la instància del formulari se li passa com a paràmetre el servei de l'escola per tal que s'omplin els camps i a l'usuari se li mostri la informació que hi ha guardada.

Llistat d'aules de l'escola

L'acció 'communities' s'encarrega de fer el llistat d'aules que pertanyen a l'escola en la que es troba l'usuari. La lògia de l'acció crea un criteri per la cerca basat en el camp pare dels serveis que ha de coincidir amb l'identificador del servei arrel que serà l'escola. Amb el criteri es crea el paginador perquè els resultats de la taula estiguin limitats com en el cas de la cerca.

La plantilla rebrà un conjunt de serveis accessibles a través del paginador, i realitzarà un recorregut accedint per cada servei al seu títol, el subtítol que ve a ser com una descripció, i el nom complet de l'autor del servei. També s'afegeixen enllaços a les accions per eliminar i activar o desactivar una aula.

Aquestes accions també del mòdul d'escola s'anomenen Erase per l'acció d'eliminar una aula, i Disenable per activar o desactivar una aula depenent del seu estat. Aquestes accions tenen una lògica molt simple, reben com a paràmetre l'identificador de l'aula que serà l'objectiu de l'operació, i en el cas de l'eliminació es realitzarà:

En el cas de l'activació o desactivació d'una aula, amb el paràmetre, s'obtindrà el servei corresponent i es canviarà l'estat de l'aula:

Es descriuen aquestes accions perquè apareixen més d'un cop en els llistats d'aules en la gestió de l'escola, d'aquesta manera es veu que només tenen com a requisit rebre un identificador que es pot passar des de qualsevol acció.

Creació d'Escola

L'acció Create del mòdul d'escola és un cas especial. El seu funcionament és com el de l'edició però canviant la classe de formulari utilitzada. Es tracte d'un cas especial perquè fins al moment la creació d'escoles està limitada al contacte a les persones interessades. Les escoles que hi han creades dins l'aplicació actualment han sigut creades sota demanda, i han sigut les necessàries per poder crear les aules corresponents obertes als usuaris.

Llistat administradors

L'acció Admin del mòdul escola ofereix les funcionalitats relacionades amb el llistat d'administradors de l'escola. La lògica de la part de la capa del controlador de l'acció es divideix en dos parts. En la primera part, la inicial, és quan s'intenta accedir per mostrar el llistat generant el paginador per una cerca amb un criteri de cerca sobre UserPermissionService amb l'identificador del servei de l'escola, i el permís ServiceAdmin que identificarà als administradors de l'escola.

A la plantilla es mostra la taula corresponent amb el username i nom complet de l'usuari de cada una de les entrades retornades de la consulta realitzada. Al costat de les dades de cada fila dins la taula es mostra un enllaç a la edició del permís, que permetrà canviar el permís de administrador a professor de l'escola només. I com a últim element de la plantilla, es mostrarà un botó per donar un permís d'administració dins l'escola a un usuari de l'aplicació.

Quan es faci clic en l'enllaç d'edició de permís s'obrirà el diàleg que permetrà editar el permís o eliminar-lo. El formulari es troba dins la plantilla i la seva definició és el de qualsevol altre formulari HTML, la diferencia es que es torna a utilitzar la funció dialog() de jQuery abans mostrada, que amaga el formulari i fa que només aparegui el popup amb el formulari al seu interior quan es realitzi el clic. Dins la definició del diàleg es troben les accions que realitzen els botons que s'afegeixen, aquestes utilitzen crides AJAX al servidor per realitzar els canvis en els permisos.

Aquestes crides es defineixen amb les funcions .ajax() de jQuery, en elles es poden definir els paràmetres que s'envien, el mètode, si es POST o GET, la URL on es realitzarà i les accions a realitzar un cop hagi retornat la crida. En tots els casos quan la crida retorna havent-se executat sense cap problema greu es comprova el valor de les dades retornades, i si es troba un missatge d'error, es procedeix a mostrar-lo. En el cas que no sigui així es realitza la manipulació corresponent a la taula.

En la banda del servidor es poden descriure dins del mòdul de l'escola diferents accions per les peticions AJAX, en el cas que sigui una edició del permís, la url apuntarà a l'acció 'Permedit', en el cas que la petició sigui per eliminar un permís la petició es fa arribar a l'acció 'Permdel'. La separació d'aquestes accions es realitza perquè hi ha més d'una acció des de la que es podran eliminar i editar els permisos d'usuaris que es llistin, com es veurà més endavant.

En el cas donar un permís d'administrador a un usuari, també es mostrarà un diàleg amb un formulari en el que introduir el correu de l'usuari a qui se li vol donar el permís. La petició AJAX que es realitzarà serà enviada a la mateixa acció de llistat d'administradors i serà on s'executi l'altre part de l'acció que estava separada i que s'havia parlat. La funcionalitat d'afegir un nou permís no s'ha separat com les altres, perquè fa referencia a un permís d'administració, que és del que s'encarrega aquesta mateixa acció, gestionar els administradors.

En la banda del servidor quan l'acció Admin rep la petició i comprova que es tracta d'una petició POST obté l'usuari que s'ha rebut com a paràmetre i fa les comprovacions que acaben en un missatge de retorn d'error:

- Usuari existeix, vol dir que és un usuari registrat
- No té el permís assignat encara

Per fer les comprovacions es realitzen les consultes, una per comprovar que existeix una instància de sfGuardUser amb el username rebut, i una altre a UserPermissionService en la que l'usuari ja tingui el permís 'ServiceAdmin' per l'escola.

Depenent del resultat s'envia un missatge o una altre, per saber com ha acabat l'execució de la petició. Un cop retornada la petició, es comprova el missatge, si la petició s'ha realitzat correctament s'insereix utilitzant jQuery dins la taula en la que es mostrarà una nova entrada amb les dades de l'usuari. En cas de no haver-se

realitzat la operació correctament s'utilitzarà el missatge d'error per mostrar-lo en la zona de missatges d'error del diàleg.

Llistat Professors

L'acció Teachers del mòdul que correspon al llistat de professors i l'edició de permisos d'aquests, és l'acció anàloga de llistar administradors, comparteix la majora de característiques i lògica, incloent les peticions AJAX i les accions que es criden. El canvi a destacar són els tipus de permisos que s'utilitzen en aquest cas, seran ServiceWrite a l'escola. Com el funcionament de l'acció, template i crides és tan semblant no es tornarà a descriure.

Llistat d'aules del professor

L'acció 'Teaching' del mòdul de l'escola s'encarrega de llistar les aules d'un professor amb els permisos que ha donat als usuaris per a aquestes aules. La lògica dins l'acció és molt semblant a les anteriors accions, canviant el criteri de selecció, al que s'afegeix a més de la restricció d'haver de ser de l'escola, que l'identificador de l'autor sigui el mateix que el de l'usuari que ha realitzat la petició. Per tant l'única diferència seria l'accés a la instància de sfGuardUser per obtenir l'identificador i afegir-lo al criteri de cerca.

Els elements d'interacció dins la vista són semblants a les accions, contenen enllaços per editar els permisos i un botó per accedir als diàlegs que contenen els formularis corresponents. El format de la taula varia però, ja que inicialment es mostra un llistat de les aules, però per cada aula que tingui permisos donats a altres usuaris es veurà una fletxa al principi que servirà com element per desplegar un llistat dins la mateixa taula mostrant els usuaris i els permisos que s'han donat per aquella aula.

Aquest llistat es genera gràcies a que a la plantilla a l'hora de generar les entrades a la taula, per cada una de les aules que s'ha d'introduir en ella, es realitza una crida a una funció del servei que s'encarrega de retornar els usuaris que tenen permisos i que no són l'autor. Un cop es tenen els usuaris amb permisos per l'aula, s'afegeixen seguidament de l'entrada a la taula per l'aula, les dels usuaris amb permisos, i afegint un codi en Javascript es produeix l'efecte de desplegament dins la taula al fer clic sobre cada aula.

Crear aula

L'acció per crear una aula en la versió anterior es trobava dins el mòdul d'aules, el motiu del seu desplaçament cap al mòdul de l'escola és perquè precisament depèn d'ella per crear-se. Ara quan es crea una aula, a més de crear el formulari que es mostrarà se li assigna un servei durant la creació, aquest procés s'ha de realitzar perquè són dades que l'usuari no introdueix al formulari, i sinó quan el formulari es guardés no es podria validar correctament, a més és informació estrictament necessària pel funcionament correcte del sistema de serveis.

Aquesta part del codi s'executa al principi de l'acció, i el que fa es crear una instància de servei buida a la que li assignarà com a servei pare l'escola que es troba com a arrel a la pila de serveis, l'última sentència és la que crea el formulari, a partir d'allà es mostra, mostrant-se un formulari per introduir el títol i subtítol de l'aula, al que a més s'ha d'introduir el password que es voldrà fer servir per accedir. La resta del comportament del formulari es com en els altres casos.

8.1.4 Implementació mòdul Mapexpl

El mòdul Mapexpl de l'aplicació conté les accions relacionades només amb els serveis de tipus Mapexpl, aquests són el serveis que ofereixen rotacions i zooms de diferents teixits i parts del cos, o d'altres formes de vida com les plantes.

El mòdul ja existia en la versió anterior les accions que contenia eren:

- Index. La pàgina principal d'un servei Mapexpl on es mostraven els teixits que estaven disponibles per visualitzar de l'ésser humà.
- Rotation. L'acció que s'accedia quan es volia visualitzar la rotació d'algun òrgan.
- Zoom. L'acció a la que s'accedia quan es volia visualitzar el zoom d'algun teixit.

Tot i que ja hi havien algunes accions dins del mòdul, a apart de les que s'han afegit de noves, l'execució de les que existien han sigut renovades casi al complet.

Acció Index

La versió anterior de l'acció index pràcticament no contenia cap mena de lògica de negoci dins de l'acció, es limitava a passar com a paràmetre el servei que s'executa i afegir els javascript i fulles d'estil.

En la plantilla de l'índex hi havia un llistat de teixits, i per cada un d'ells es realitzava una crida amb el servei Mapexpl per comprovar la seva disponibilitat en aquell servei. La funció en aquells sempre retornava com a cert per la disponibilitat dels teixits, tampoc tenia lògica implementada.

A diferència de la versió anterior, en la actualitat les comprovacions de disponibilitat ja no es troben dins la plantilla, i en el template no apareix un llistat extens de totes les localitzacions que s'ha de comprovar si estan disponibles. Aquest resultava un gran problema a l'hora d'afegir nous teixits, ja que al realitzar un canvi s'havia de buscar entre totes les sentències i condicionals de comprovacions de disponibilitat i afegir o treure en el punt adient, i sinó crear un bloc nou on posicionar-lo, el manteniment resultava molt pesat i lent.

Quan l'acció s'inicia, a part d'afegir les fulles d'estil i Javascripts que s'afegien anteriorment, es crida a la classe del model TableMapexplPeer per realitzar una consulta i obtenir els teixits disponibles que s'han de mostrar. L'acció també s'encarrega de comprovar de quin subtipus de servei Mapexpl es tracta, ja que depenent si és un servei de plantes, o humà s'haurà de tenir en compte el sexe de l'espècie o no, per aquest motiu existeix una consulta getTablesActives(...) i getTablesAcivesSexaualized(...) que tracten les línies depenent si tenen atributs relacionats amb el gènere de l'espècie on han d'aparèixer o no.

Aquestes consultes recorren l'estructura de dades del fitxer app.yml accedint primer a les taules que hi han definides per l'espècie que tracten, i per cada taula accedeixen a les dades de les línies de teixits que poden anar dins de la taula fent consultes a la instància de Mapexpl del servei per determinar si la línia es accessible o no. En el cas que si que ho sigui es crea una instància de LineDefMapexpl, que conté les dades de la línia i s'afegeix a un array de línies que s'introduirà al final a la instància creada de TableMapexpl.

L'acció rep aquest conjunt de taules que seran passades a la vista perquè pugui generar les taules de les línies de teixits que són disponibles en el servei.

L'avantatge de passar un conjunt de taules i línies encapsulats en les seves classes és que la vista queda molt simplificada. Per una banda la vista utilitza el sub-tipus de servei Mapexpl per determinar quin és el parcial que ha d'introduir en el cos de l'aplicació, ja que no és el mateix mostrar les localitzacions en dos cossos d'ésser humà que en les figures de tres plantes, cadascuna té els seus elements i la seva disposició a la pàgina.

Un cop s'ha inclòs el parcial corresponent, es pot accedir a les dades de les taules fent un recorregut de l'array rebut i per cada una accedir a les dades utilitzant les funcions Getters que tenen definides dins d'un bucle. Quan s'ha acabat de recórrer totes les taules ja es troben totes posicionades en el seu lloc i llestes per ser visualitzades.

L'estructura de les línies i taules que es fa servir és una representació plana i senzilla de la informació que s'ha d'utilitzar per posicionar-les a l'acció index, contenen la representació amb camps com el nom que ha d'aparèixer, la posició, si és exclusiva d'un sexe, ... L'estructura de les utilitzada per guardar la seva informació, conté les dades que en la versió anterior, era calculada, en aquest cas el número d'imatges, i per una altre banda el valor que faltava definir per poder realitzar la funció isAvailable() de Mapexpl que s'encarrega de realitzar la màscara per determinar l'accessibilitat.

Les accions que venen a continuació són molt similars, el motiu perquè s'hagin hagut de mantenir separades és perquè accedeixen a recursos diferents, cada acció ha d'accedir a classes i taules de l'aplicació diferents, però el funcionament és molt similar.

Acció Zoom/Rotation

En la versió antiga es cridava a un script PHP extern que s'introduïa dins d'un iframe que es creava al template. L'iframe encara es manté a la versió actual, això es degut a problemes de posicionament de l'aplicació Javascipt que s'executa dins d'ell. Però actualment en comptes de cridar a un script PHP que realitza totes les crides a la base de dades i interaccions de la banda del servidor, es crida a una acció que ha integrat totes aquestes operacions dins del mòdul i per tant aprofita l'estructura i mecanismes de Symfony.

Accions Windowzoom / Windowrotation

Aquestes són les accions que es criden i s'executen dins de l'iframe cridat per les accions anteriors. Les accions són noves, i adapten tot el contingut que hi havia dins de l'antic scipt PHP. L'script contenia tant lògica de negoci que s'encarregava d'obtenir els paràmetres i realitzar les operacions que rebia per peticions de l'aplicació javascript, com també contenia la part de la vista on s'afegien tots els javascript, fulles d'estil i s'inicialitzava l'aplicació amb l'identificador del teixit/òrgan corresponent.

Aquesta part ha sigut separada, per una banda està el template en el que s'afegeixen els includes dels Javascript i fulles d'estil necessàries per l'aplicació. També s'encarrega d'obtenir les dades de la línia que se li ha passat des de la capa del controlador i amb ells incialitza els paràmetres per l'aplicació.

A la part de la capa de controlador s'han afegit només les crides inicials que es realitzen per passar els continguts de l'aplicació, que són les etiquetes. Si recordem l'apartat on es descriu el model de dades, s'expliquen un conjunt de taules relacionades amb unes etiquetes per les aplicacions de zooms, rotacions i 3D, en aquest cas s'accedeix a les de zooms i rotacions.

Quan l'acció rep una petició obté de l'app.yml les dades relacionades amb la línia que rep com a paràmetre, aquests paràmetres seran passats al template perquè siguin accessibles a l'aplicació.

Es realitzaran les consultes a la base de dades utilitzant com a criteris l'identificador de la línia per obtenir només les etiquetes d'aquell teixit, i l'identificador de servei. La política utilitzada per la selecció d'etiquetes dins del serveis està definida per l'identificador de servei. En cada aula, haurien d'aparèixer les etiquetes que els gestors de l'aula han introduït, però a més, l'aplicació té per defecte un conjunt d'etiquetes per diversos teixits, aquestes etiquetes estan disponibles per totes les aules de l'aplicació web.

Per no haver d'afegir un camp extra a totes les classes d'etiquetes del model es va decidir utilitzar un identificador especial per aquestes etiquetes globals, aquest identificador fa referència a una aula específica de l'aplicació en la que s'introdueixen aquestes etiquetes i que només tenen accés els encarregats d'introduir etiquetes a l'aplicació. D'aquesta manera no s'ha de realitzar cap tasca

de gestió i totes les etiquetes que s'afegeixin o canviïn en aquella aula es podran veure reflectides immediatament en la resta d'aules.

El resultat de les consultes amb les etiquetes es processen per ser introduïts en arrays utilitzant el format jSON, perquè des de l'aplicació puguin ser utilitzades i mostrades.

Accions tagsZoom / tagsRotation

Aquestes accions són les que s'encarreguen de les peticions que genera l'aplicació en Javascipt, els events que provoquen aquestes peticions poden ser:

- Canvi idioma en el selector de l'aplicació javascript.
- Inserció d'una etiqueta
- Edició d'una etiqueta
- Eliminació etiqueta

Aquests aspectes es tenen en compte quan es rep una petició a l'acció. Al iniciar l'execució de l'acció es comproven que hi hagin els paràmetres necessaris, quin tipus de petició es realitza, i si es tenen els permisos necessaris. S'ha de tenir en compte que per les accions dins de cada mòdul es defineixen uns credencials que són necessaris per poder executar-les, en el cas de Mapexpl només es demana que el credencial sigui ServiceRead o d'altre forma no podrien accedir els usuaris normals o anònims. És per aquest motiu que dins de les accions s'han de realitzar les comprovacions de permisos quan es volen realitzar operacions i no consultes a les etiquetes.

Per acabar l'acció genera el missatge de retorn en format JSON, ja sigui un missatge d'error, o retornant l'identificador de l'etiqueta que s'ha realitzat la operació i un missatge d'haver realitzat l'acció correctament. En el cas de tractar-se d'una petició per canviar l'idioma de les etiquetes, es realitzarà una consulta per obtenir les etiquetes en l'idioma demanat i es retornaran com en les accions anteriors.

8.1.5 Implementació Mòdul Bones

El mòdul Bones conté les accions relaciones amb un servei per visualitzar Tags en tres dimensions i poder realitzar captures en el visualitzador per després afegir etiquetes. El servei quan passa a executar-se consta d'un applet que s'encarrega de les visualitzacions dels Tags, i que ja estava implementat. Dins l'applet s'ofereix la possibilitat de realitzar captures al visualitzador amb les que després es podran gestionar i mostrar les etiquetes que hagin sigut introduïdes en elles. En la versió anterior el comportament del mòdul diferia encara més que el de Mapexpl. Només contenia l'acció index per accedir la pàgina a la que accedir a l'aplicació.

Acció index

La versió anterior de l'acció només introduïa dins l'acció un javascript i una fulla d'estil, com en l'acció index de Mapexpl, però a deferència d'aquest no havien continguts, només es deixava accessible un punt que feia d'enllaç per descarregar i executar un Java Web Start.

L'acció que s'ha implementat en la nova versió és molt semblant a l'actual de Mapexpl també, això és degut a que amb el disseny i implementació de les taules i línies s'ha buscat que la forma en que es realitzen les operacions, encara que variïn i cada mòdul tingui la seva especialització i alguns camps variïn, els esquemes i patrons a seguir siguin els mateixos.

L'acció s'encarrega també d'obtenir les taules a través de TableBonesPeer per després accedir a elles de la mateixa forma que es fa a Mapexpl i recórrer l'array de taules per mostrar-les a la plantilla. En aquest cas encara és més senzill, ja que no cal fer cap diferenciació entre sexes, o diferents espècies. Les línies i taules que hi han a la estructura de dades app.yml per Bones només conté línies i taules d'un grup, el de l'ésser humà i que són les que s'accedeixen i es mostren.

Acció 3dshow

A aquesta s'acció s'arriba a través de l'índex, quan es clica en una de les localitzacions per ser visualitzades. Amb la petició arriba l'identificador de la línia a la que es vol accedir i que s'utilitzarà per accedir a les línies de l'app.yml per

configurar certs paràmetres necessaris per poder executar l'applet del servei correctament:

- Url del model. Els models són un conjunt d'imatges que es troben en un directori del servidor i al que s'ha d'indicar la URL perquè l'applet que s'executarà pugui accedir al directori i obtenir les imatges necessàries per funcionar.
- Rang. És un paràmetre que determina quantes imatges ha de carregar per visualitzar el model.
- Idprofile. És necessari per saber quin perfil d'usuari és, com l'applet no pot accedir a l'usuari i als seus credencials, se li passa com a paràmetre un enter que determina si té permisos d'escriptura per realitzar captures i guardar-les, o si només té permisos de lectura i les captures es guardaran localment en el sistema de fitxers de l'usuari.
- Identificador del servei. L'applet quan crea una captura genera un nom amb alguns caràcters aleatoris, el nom del model i l'identificador del servei, sense els quals no es podrien identificar i carregar imatges que s'hagin realitzat i guardat per l'aula.

En la vista d'aquesta acció també s'introdueix un botó que activa un editor d'etiquetes per la captura que es tingui seleccionada. L'editor es igual que els de Mapexpl.

Acció Tags3d

Aquesta acció es equivalent a Tagszoom/rotation de Mapexpl, l'únic aspecte que canvia són els camps que s'introdueixen com a criteri de cerca, per les diferencies entre les classes d'etiquetes, però la mecànica, i el número de consultes, el tractament d'errors, i la resposta és la mateixa, per aquest motiu no es tornarà a descriure.

9. Consideracions Finals

Durant la realització del projecte, l'ús del framework Symony ha suposat una gran ajuda pel desenvolupament de l'aplicació. L'estructura, l'organització en capes i la modularitat dels components que intervenen en cada una de les capes ha resultat molt adient per l'aplicació en la que s'ha treballat per la seva orientació a oferir serveis a l'usuari.

Tenint en compte que l'aplicació ja existia quan vaig començar el projecte i la seva renovació interior, és molt probable que els següents passos a seguir en les futures revisions estiguin orientats a la introducció de nous serveis per oferir a l'usuari, tasca que ja s'ha realitat amb el mòdul entitat, i tot i que es modificava la jerarquia de serveis la resta de mòduls no s'han hagut de refer. Per tant, en el cas d'afegir un nou servei dins d'una aula s'espera que no provoqui cap canvi en el codi actual.

Com a punt negatiu que afecta al framework actualment utilitzat s'ha de destacar les versions de software que s'utilitzen, en el cas de Symfony és la 1.1, quan actualment s'utilitza la 1.4. La versió actual no està mantinguda encara que existeixen plugins i suport per part de la comunitat d'usuaris que encara l'utilitzen. S'hauria de tenir en compte que els nous plugins surten només per les noves versions, i els que s'utilitzen actualment poden haver rebut revisions i haver millorat el seu funcionament i/o estendre les funcionalitats que ofereixen, aquestes millores i canvis però, només arriben per les noves versions que estan mantingudes.

No obstant pot haver un debat sobre si realitzar l'actualització a una de les versions estables actuals o esperar a la versió 2 que sortirà cap a finals d'aquest any, que promet una millora i renovació molt important, no solament de rendiment respecte l'última versió. La discussió es genera pel fet que es produirà una remodelació del nucli, i l'actualització de l'aplicació pot acabar esdevenint en una re-definició d'aquesta. Com les preferències del projecte han sigut aportar noves funcionalitats i sembla que continuarà sent així, és molt provable que s'opti per una actualització a les versions actuals.

També cal destacar que tal i com Symfony ha anat evolucionant, Propel també ho ha seguit fent, actualment es troben en la versió 1.5 i les millores en rendiment mostren un resultats molt superiors en les operacions realitzades, la diferencia es pot observar respecte anteriors versions, especialment 1.4 respecte a les anteriors, i amb l'última versió estable de l'ORM competidor en Symfony, Doctrine 1.2. Una

actualització de Propel aniria lligada a l'actualització de la versió de Symfony, ja que cada versió està dissenyada per treballar amb una versió de l'ORM en concret.

L'actualització a una nova versió de Symfony no s'ha pogut realitzar pel cost que hagués representat fer un anàlisis de les diferències en el funcionament i les APIs de les diferents versions ja que hi han diversos elements incloent-ne dels formularis que s'han quedat obsolets, i un cop fet l'anàlisi i aplicats els canvis s'hauria d'entrar en una fase de proves que demandava d'una gran quantitat de temps.

Dins de l'aplicació, s'ha presentat com a alternativa al funcionament del model de la vista actual la possibilitat d'utilitzar crides AJAX per recarregar només el contingut de l'aplicació que ha de canviar quan un usuari es mou a través de l'aplicació. Aquest comportament tindria sentit tenint en compte que hi han certs elements de la vista que no varien durant tota l'aplicació com és el menú superior. Hi han uns altres però que poden variar, concretament quan l'usuari faci un canvi de l'entorn en el que es troba a l'aplicació, aquests canvis d'entorn afecten als menús laterals, que segons si ens trobem dins l'àmbit d'una aula o la gestió d'una escola varien les opcions que contenent carregant diferents parcials.

L'adaptació no hauria de ser molt dramàtica, ja que Symfony proveeix de Helpers per realitzar crides AJAX per obtenir una part de la vista i substituir-la en la zona que se li especifiqui, i en el cas de necessitar funcionalitats més elaborades o el seu ús amb jQuery també hi han plugins disponibles per proporcionar els Helpers necessaris.

Un altre aspecte a millorar en un futur poden ser les funcionalitats de l'aplicació per poder gestionar estadístiques de l'ús dels usuaris de l'aplicació i els seus serveis actualment. Es basen primer en l'ús d'StatCounter com a sistema d'estadístiques per seguir les connexions realitzades a l'aplicació, i per una altre banda s'accedeix als logs que registren informació dels accessos a les aules per part dels usuaris quan han accedit.

Per estendre la informació que es pot oferir per part de l'aplicació amb el sistema de logging s'havia plantejat crear una base de dades amb SQLite on emmagatzemar missatges i informació sobre els usuaris, els serveis i el temps, que després es podria utilitzar per obtenir la informació selectivament. Fent ús de les consultes a base de dades es proporcionaria un entorn molt més flexible en quant a la informació que es podria mostrar oferint una llegibilitat molt més alta que no pas la

d'uns fitxers de logs i que acompanyada amb una interfície com a backend l'aplicació podria tenir el seu propi sistema d'estadístiques i de manteniment sense haver de recórrer a eines externes.

Tots aquests canvis o actualitzacions que es descriuen no formaven part del desenvolupament inicial que es va plantejar durant el projecte, són idees que han anat apareixent durant el projecte i que s'han discutit perquè es creu que poden aportar millores al funcionament de l'aplicació, ja que no afecten directament a l'usuari, no obstant no s'ha descartat que no s'arribin a introduir en un altre nova fase de desenvolupament de l'aplicació.

10 Conclusions

Al final del projecte s'ha aconseguit realitzar els objectius i implementar les funcionalitats que s'havien plantejat per introduir dins l'aplicació web. Una de les parts que ha tingut més importància dins del projecte ha sigut la fase d'aprenentatge del framewok Symfony, ja que s'ha hagut de conèixer el funcionament del framework i adaptar-me a les metodologies de desenvolupament que s'utilitzen. L'experiència no podria haver sigut més satisfactòria, el framework compleix amb tot el que promet, un desenvolupament ràpid i àgil d'aplicacions, i dels seus elements permetent oblidar-se dels aspectes repetitius i centrar-se en la implementació.

La part més costosa ha sigut assimilar l'estat en que es trobava el desenvolupament de l'aplicació, al existir un disseny i una implementació el següent pas va ser comprendre com funcionava internament l'aplicació. La falta de documentació per part del disseny que s'havia implementat va resultar un handicap, havent de realitzar un procés d'anàlisi i de proves amb el codi de l'aplicació fins arribar a tenir una visió completa de l'aplicació amb el màxim nivell de detall.

Durant el desenvolupament del projecte tot i que els objectius s'han anat complint tal i com es definien, en alguns casos podent variar la implementació interna que inicialment s'esperava que donaria la resposta al problema plantejat, m'he trobat en amb moments en que els plantejaments de les etapes i la seva duració que s'havien establert per realitzar alguns objectius s'han realitzat en més temps de l'esperat, havent d'intensificar en algunes parts el desenvolupament per no allargar les dates establertes. Aquest problema ha vingut derivat per algunes especificacions massa dèbils o poc exigents en alguns casos, i que per tant portaven a unes estimacions i valoracions prèvies errònies que provocaven haver de realitzar correccions en el calendari per equilibrar les diferències entre fases.

Per acabar, destacar que com en la realització del projecte, l'ús d'un framewok com Symfony està altament recomanat si es vol dur a terme un projecte de certa envergadura, l'experiència que he tingut ha sigut molt bona, tant que actualment segueixo i participo dins la comunitat per tal de resoldre dubtes relacionats amb el framework i per tal d'obtenir noves idees amb les que després experimentar utilitzant Symfony i poder aportar futures millores a l'aplicació o altres projectes.

Aquest aspecte ha sigut de gran ajuda, ja que inicialment un pot no veure fins on poden arribar les aplicacions dels mecanismes utilitzats, i els coneixements que es poden arribar a adquirir a partir dels problemes i solucions, dins la comunitat són inimaginables.

11 Bibliografía

HTML

HTML Dog: The Best-Practice Guide to XHTML and CSS, Patrick Griffiths

,New Riders, November 22, 2006, ISBN-10: 0-321-31139-6

W3Schools : http://www.w3schools.com

CSS

CSS3 : http://www.css3.info/

Learning Web Design, Third Edition, Jennifer Niederst Robbins, O'Reilly Media, Inc., June 29, 2007, ISBN-13: 978-0-596-52752-5

Core CSS: Cascading Style Sheets, 2nd Edition, Keith Schengili-Roberts,

Prentice Hall, September 23, 2003, ISBN-10: 0-13-009278-9

Javascript

Pure JavaScript, Second Edition,R. Allen Wyke; Jason D. Gilliam; Charlton Ting; Sean Michaels, Sams, August 15, 2001, ISBN-10: 0-672-32141-6

Learning JavaScript, 2nd Edition, Shelley Powers, O'Reilly Media, Inc.

December 16, 2008, ISBN-13: 978-0-596-52187-5

jQuery

jQuery : http://jquery.com/

jQueryUI: http://jqueryui.com/

YUI CSS

YUI: http://developer.yahoo.com/yui/3/cssreset/

PHP

Learning PHP 5, David Sklar, O'Reilly Media, Inc., June 25, 2004

ISBN-13: 978-0-596-00560-3

PHP Cookbook, 2nd Edition, Adam Trachtenberg; David Sklar

O'Reilly Media, Inc., August 25, 2006, ISBN-13: 978-0-596-10101-5

Learning PHP & MySQL, 2nd Edition, Michele E. Davis; Jon A. Phillips

O'Reilly Media, Inc., August 17, 2007, ISBN-13: 978-0-596-51401-3

PHP Manual : http://www.php.net/manual/en/

Symfony

Documentació de Symfony: http://www.symfony.es/documentacion/

Tutorial Jobeet: http://librosweb.es/jobeet_1_3/

PDO: http://es.php.net/pdo

Propel: http://www.propelorm.org/wiki/

Rendiment de Propel: http://propel.posterous.com/how-fast-is-propel-15

MySQL

MySQL®, Fourth Edition, Paul DuBois, Addison-Wesley Professional, 29-AUG-2008

MySQL Stored Procedure Programming, 1st Edition, Guy Harrison; Steven Feuerstein; O'Reilly Media, Inc; March 28, 2006; ISBN-13: 978-0-596-10089-6

SQLite

The Definitive Guide To Sqlite, Mike Owens,, Apress, 25/05/2006

12 Annexes

12.1 Consultes amb Propel i Criteria

```
public function executeTagszoom($request)
 {
        $this->forward404Unless($services = ServicePeer::getCurrentServicesStack());
        $mapexpl = array_pop($services);
        $user_id=$mapexpl->getId();
        $sf_user = $this->getUser();

        if($sf_user->hasCredential('ServiceRead'))
        {
         $text=$request->getParameter('text');
         $this->settext=isset($text);

         //Obtenir els paràmetre
         $id=$request->getParameter('id');
         $type=$request->getParameter('type');
         $line_id=$request->getParameter('line_id');
         $scale=$request->getParameter('scale');
         $ap_min=$request->getParameter('ap_min');
         $ap_max=$request->getParameter('ap_max');
         $pos_x=$request->getParameter('tag_x');
         $pos_y=$request->getParameter('tag_y');
         $langs=$request->getParameter('language');
         $culture=$request->getParameter('lang');

         //Construim la resposta JSON
         if($type=="tags-lang") {
                $call = new Criteria();
                $call->add(PhotolabelPeer::LINE_ID,$line_id);
                $call->add(PhotolabelPeer::USER_ID,array($user_id,'14'),CRITERIA::IN);
                $call->addAscendingOrderByColumn(PhotolabelPeer::ID);
                $photoresult=PhotolabelPeer::doSelect($call);

                $arrayLabels=array();

                foreach($photoresult as $photo)
                {
                        $user_id=$photo->getUserId();
                        $line_id=$photo->getLineId();
                        $pos_x=$photo->getTagX();
```

```
$pos_y=$photo->getTagY();
$scale=$photo->getScale();
$ap_max=$photo->getApMax();
$ap_min=$photo->getApMin();
$id=$photo->getId();

$c2=new Criteria();

$subSelect = "photolabelrlang.PHOTOLABELID IN (
                    SELECT
                        photolabelrlang.PHOTOLABELID
                    FROM
                        photolabelrlang, lang
                    WHERE
                        lang.ID = photolabelrlang.LANGID
                     AND
                        photolabelrlang.PHOTOLABELID =".$id."
                     AND
                        lang.SYMBOL ='".$culture."')";

$c2->addJoin(PhotolabelrlangPeer::LANGID,LangPeer::ID, Criteria::INNER_JOIN);
$c2->add(PhotolabelrlangPeer::PHOTOLABELID, $id);
$c2->addAnd(PhotolabelrlangPeer::PHOTOLABELID,$subSelect,Criteria::CUSTOM);
$c2->addAscendingOrderByColumn(LangPeer::ID);
$joinresult=PhotolabelrlangPeer::doSelectJoinLang($c2);

if($joinresult){
        $arrayLangs=array();

        foreach($joinresult as $ph)
        {
                $symbol=htmlentities($ph->getLang()->getSymbol());
                $desc=htmlentities($ph->getDescription());
                $name=htmlentities($ph->getName());
                $itemArray=array("lang"=>$symbol,"name"=>$name,"desc"=>$desc);
                $arrayLangs[]=$itemArray;
        }

        $label=array("id"=>htmlentities($id
         ),"line_id"=>$line_id,"ap_min"=>$ap_min,"ap_max"=>$ap_max,
         "scale"=>$scale,"pos_x"=>$pos_x,"pos_y"=>$pos_y,
        "user_id"=>$user_id,"langs"=>$arrayLangs);

        $arrayLabels[]=$label;
```

```
                }
            }
            $arrayResponse=array("status"=>"ok","tags"=>$arrayLabels);

            //Construeixo la resposta
$jsonResponse2 = json_encode($arrayResponse);
$this->jsonResponse = str_replace("\\n", "\\\\n", $jsonResponse2);
        }
        else if($sf_user->hasCredential('ServiceWrite')){
        //Si és una inserció
        if($type=="ins")
        {

            //Afegeixo la etiqueta principal
            $phlabel=new Photolabel();
            $phlabel->setLineId($line_id);
            $phlabel->setTagX($pos_x);
            $phlabel->setTagY($pos_y);
            $phlabel->setScale($scale);
            $phlabel->setApMin($ap_min);
            $phlabel->setApMax($ap_max);
            $phlabel->setUserId($user_id);
            $phlabel->save();
            $phid = $phlabel->getId();

            //La guardo a la taula d'etiquetes relacionada amb llenguatge
            //Parsejo la llista d'idiomes
            $taulaIdiomes=explode(',',$langs);
            //Per cada idioma
            for($i=0;$i<sizeof($taulaIdiomes);$i++)
            {
                //Obtinc el symbol de l'idioma
                $symbolIdioma=$taulaIdiomes[$i];

                //Obtinc l'identificador de l'idioma
                $critLang = new Criteria();
                $critLang->add(LangPeer::SYMBOL, $symbolIdioma);
                $result=LangPeer::doSelectOne($critLang);
                $langId=$result->getId();
                //Obtinc els camps associats a l'idioma
                $labName= utf8_decode($request->getParameter('name_'.$symbolIdioma));
                $labDesc= utf8_decode($request->getParameter('desc_'.$symbolIdioma));
                //Afegeixo els valors a la taula PhotoLabelRLang
                $phrlang=new PhotolabelrIang();
```

```
                $phrlang->setPhotolabelid($phid);
                $phrlang->setLangid($langId);
                $phrlang->setName($labName);
                $phrlang->setDescription($labDesc);
                $phrlang->save();
        }

        //Construdeixo la resposta
        $this->jsonResponse ='{"status":"ok", "tag_id":"'.$phid.'"}';
}else if(PhotolabelPeer::RetrieveByPk($id)->getUserId()==$user_id ) {
 //Entra si:  1)No es etiqueta nostre  2)es etiqueta nostre pero es un superAdmin
 //Si és una modificació
 if($type=="mod")
 {
        //Modifico l'etiqueta a la base de dades
        //Modifico la taula d'etiquetes
        $phlabel=PhotolabelPeer::RetrieveByPk($id);
        //$subrot=SubrotationlabelPeer::RetrieveByPk($id,$state);

        $phlabel->setApMin($ap_min);
        $phlabel->setApMax($ap_max);
        $phlabel->setTagX($pos_x);
        $phlabel->setTagY($pos_y);
        $phlabel->setScale($scale);
        $phlabel->save();

        //La modifico a la taula d'etiquetes relacionada amb llenguatge
        //Esborro les anteriors tuples d'etiquetes associades als llenguatges
        $critDel= new Criteria();
        $critDel->add(PhotolabelrlangPeer::PHOTOLABELID,$id);
        PhotolabelrlangPeer::doDelete($critDel);

        //Parsejo la llista d'idiomes
        $taulaIdiomes=explode(',',$langs);
        //Per cada idioma
        for($i=0;$i<sizeof($taulaIdiomes);$i++)
        {
                //Obtinc el symbol de l'idioma
                $symbolIdioma=$taulaIdiomes[$i];
            //Obtinc l'identificador de l'idioma
                $critLang = new Criteria();
                $critLang->add(LangPeer::SYMBOL, $symbolIdioma);
                $result=LangPeer::doSelectOne($critLang);
                $langId=$result->getId();
```

```
                        //Obtinc els camps associats a l'idioma
                        $labName= utf8_decode($request->getParameter('name_'.$symbolIdioma));
                        $labDesc= utf8_decode($request->getParameter('desc_'.$symbolIdioma));
                        //Afegeixo els valors a la taula PhotoLabelRLang
                        $phrlang=new Photolabelrlang();
                        $phrlang->setPhotolabelid($id);
                        $phrlang->setLangid($langId);
                        $phrlang->setName($labName);
                        $phrlang->setDescription($labDesc);
                        $phrlang->save();
                    }
                    //Construdeixo la resposta
                    $this->jsonResponse ='{"status":"ok"}';
    } else if($type=="del") {
                    //Esborro l'etiqueta de la base de dades
                    //Esborro de la taula d'eqtiquetes
                    $rot=PhotolabelPeer::RetrieveByPk($id);
                    $rot->delete();

                    $critDel= new Criteria();
                    $critDel->add(PhotolabelrlangPeer::PHOTOLABELID,$id);
                    PhotolabelrlangPeer::doDelete($critDel);

                    //Construdeixo la resposta
                    $this->jsonResponse ='{"status":"ok", "tag_id":"'.$id.'"}';
          }
         }
         else {
          $arrayResponse = array("status" => "ko");
          $this->jsonResponse=json_encode($arrayResponse);
   }
        } else {
         $arrayResponse = array("status" => "ko");
         $this->jsonResponse=json_encode($arrayResponse);
 }
 }
        else {
         $arrayResponse = array("status" => "ko");
         $this->jsonResponse=json_encode($arrayResponse);
 }
}
```

12.2 Formulari de crear Aula

Acció

```
public function executeAdd($request)
{
  $services = ServicePeer::getCurrentServicesStack();
  $serviceType = ServiceTypePeer::getCurrentServiceTypeStack();
  $this->forward404Unless($service = $services[0]);
  $this->s = $service;
  $newServiceName='Community';
  $newservice = new Service($this->getUser()->getGuardUser());
  $newservice->setServiceTypeName($newServiceName);
  $this->ss = ServiceTypePeer::retrieveByName($newServiceName);

  $this->forward404Unless($this->ss && $this->ss->getParentName() == $serviceType[0]->getName()
);
  $newservice->setParent($service->getId());
  $this->form = new CommunityCreateForm($newservice);
  if($request->isMethod('post')){

        $serviceFormReq = $request->getParameter('service');
        $newservice->setParent($services[0]->getId());
      $serviceFormReq['captcha'] = $captcha = array(
          'recaptcha_challenge_field' => $request->getParameter('recaptcha_challenge_field'),
          'recaptcha_response_field'  => $request->getParameter('recaptcha_response_field'),
          );
        $this->form->bind($serviceFormReq);

        if ($this->form->isValid())
        {
              $newservice = $this->form->save();
              $this->redirect('community/index?'.ServicePeer::getParametersByService($newservice));
        }
  }
}
```

Template

```
<?php $service = $form->getObject() ?>
<?php use_helper('Javascript') ?>
<?php use_helper('I18N') ?>
<?php use_stylesheet('community/formstyle') ?>
<?php use_javascript('roman/jquery/ui/jquery.ui.button.min.js') ?>
<?php use_javascript('roman/genomedu-screen/submitbtn.js') ?>

<div id="form" class="center">
<h1><?php echo __("New classroom") ?></h1>

<form action="<?php echo url_for('entity/add?'.ServicePeer::getCurrentServiceParameters()) ?>"
name="community" method="post" <?php $form->isMultipart() and print 'enctype="multipart/form-data" ' ?
>>
  <table class="form_in">
    <thead><tr><td colspan="2"></td></tr></thead>
    <tfoot>
      <tr>
        <td colspan="2">
          <input type="submit" value="<?php echo __('Create classroom') ?>" />
        </td>
      </tr>
    </tfoot>
    <tbody>
      <?php echo $form['title']->renderRow() ?>
      <?php echo $form['subtitle']->renderRow() ?>
      <?php echo $form['password']->renderRow() ?>
      <?php echo $form['password_again']->renderRow() ?>
      <tr><td colspan="2"><?php echo __("Please, type the following words:") ?></td></tr>
      <tr>
            <td colspan="2"><?php echo $form['captcha']->render() ?></td>
      </tr>
    </tbody>
  </table>

</form>
</div>
<?php include_partial('global/entitymenu', array('service' => $service)) ?>
```

12.3 Estructura dins l'aplicació de Taules i línies

```
mapexpl:
  tables:
    Human:
      Artery:
        i18n: { en: Artery, ca: Arteria, es: Arteria }
        male_x: 45
        male_y: 50
        female_x: 45
        female_y: 50
        pos_x: 45
        pos_y: 50
        genre:  m
        style:  asexual
        zoomlines: [ Artery ]
        zoomlines_nom: [ Zoom ]
        rotationlines: [ ]
        rotationlines_nom: [ ]
  zoomlines:
    Human:
      Artery:
        lupe: 0
        micro: 11
        electro: 0
        scales: "20,30,40,70,100,150,200,350,500,750,1000"
        lupet: -
        microt: mouse
        electrot: -
        mask:  2
        i18n: { en: Artery, ca: Arteria, es: Arteria }
```