CREACIÓ D'UN CMS AMB METEORJS I ANGULARJS

Jordi Campeny Puig

Grau de Multimèdia

Enginyeria web

Ignasi Lorente Puchades

Carlos Casado Martinez

25 de Gener del 2017

FITXA DEL TREBALL FINAL

Títol del treball	Creació d'un CMS amb MeteorJS i AngularJS
Nom de l'autor	Jordi Campeny i Puig
Nom del consultor/a	Ignasi Lorente Puchades
Nom del PRA	Carlos Casado Martinez
Data de lliurament (mm/aaaa)	01/2017
Titulació o programa	Grau de Multimèdia
Àrea del Treball Final	Enginyeria web
Idioma del treball	Català
Paraules clau	CMS, AngularJS, MeteorJS

RESUM DEL TREBALL

La principal finalitat d'aquest treball és desenvolupar un sistema gestor de continguts a partir de les llibreries MeteorJS i AngularJS, sistema que ha de permetre a qualsevol usuari, amb uns determinats coneixements de maquetació web, crear i personalitzar una pàgina web a través del servidor central, el qual gestiona tots els llocs webs que utilitzin aquest CMS.

Per dur a terme aquest projecte s'ha optat per Scrum, una metodologia àgil que permet, de forma recursiva, dissenyar i desenvolupar funcionalitats de la plataforma tenint sempre un producte en producció.

D'altra banda, en referència al desenvolupament de cada un dels components que conformen el sistema, s'ha aplicat el concepte d'arquitectura modular, i així crear un disseny on cada funcionalitat del sistema es divideix i s'encapsula de forma independent de la resta. La comunicació entre aquests diferents mòduls es duu a terme a través dels seus corresponents serveis i/o esdeveniments gestionats pel framework.

Com a resultat s'ha obtingut un gestor de continguts capaç de centralitzar la gestió de qualsevol pàgina web de la que disposi de la IP i d'un arxiu pujat per FTP. Així, l'usuari pot editar qualsevol element d'una pàgina a través dels estils i de la personalització de la graella, a més a més de poder refrescar i actualitzar la pàgina web externa amb un únic *click*.

Finalment podríem dir que, tot i assolir els objectius mínims exposats en aquesta memòria, i

tenint un producte estable, encara queden moltes funcionalitats pendents per esdevenir-lo en un producte competitiu a nivell laboral. Aquestes mancances es solucionarien amb, entre d'altres, l'adhesió de pluguins, el desenvolupament de la plataforma de suport per desenvolupadors, la descentralització de les dades, etc.

ÍNDEX

1. INTRODUCCIÓ

1.1 CONTEXT I JUSTIFICACIÓ DEL TREBALL.

La idea del sistema neix de l'alta demanda en els darrers temps en quant a creació de pàgines web. Alguns sistemes de gestió de continguts (CMS), com Wordpress[1] o Drupal[2], ofereixen la possibilitat de crear-les en qüestió de segons i amb un procés d'instal·lació força intuïtiu i ràpid, a més de permetre't la possibilitat d'editar tot el seu contingut. Malgrat això, es necessari un procés d'instal·lació i, en funció de les modificacions que es vulguin dur a terme, requerirà d'uns coneixements tècnics.

Així doncs, aquest nou CMS, en les seves versions finals, vindrà a cobrir la necessitat de creació d'una pàgina web sense processos d'instal·lacions ni coneixements tècnics i tot això a un Host personal i totalment independent, fent que totes aquelles persones amb coneixements de maquetació puguin desenvolupar pàgines web, centrant-se exclusivament amb el disseny, ja que el procés de codificació es durà a terme mitjançant una abstracció, representada visualment a través del panell d'administració del CMS.

Per una banda, no requerirà d'un procés d'instal·lació, ja que tota la lògica de les pàgines generades es centrarà en un únic servidor central i una única base de dades, d'aquesta manera el propietari d'una pàgina només haurà de pujar al seu servidor un únic arxiu a través del FTP, aquest arxiu serà l'encarregat de gestionar totes les comunicacions i instal·lacions amb els gestor de continguts central, en altres paraules, obtindrà de forma reactiva tot el contingut necessari, en aquest cas, arxius html, css, js i imatges.

1.2 OBJECTIUS DEL TREBALL

El projecte té com a objectiu el desenvolupament d'un sistema central de gestor de continguts, el qual connectarà, actualitzarà i generarà pàgines web. Aquest sistema facilitar a qualsevol persona la creació, gestió i administració d'una pàgina web, sense la necessitat de

[1] Per a més informació consulti: https://es.wordpress.com/ Data d'última consulta: 01.01.2017
[2] Per a més informació consulti: https://www.drupal.org/ Data d'última consulta: 01.01.2017

coneixements tècnics. A partir del propòsit plantejat anteriorment, s'esmenten seguidament els objectius a aconseguir:

Crear un sistema gestor de continguts a un servidor independent, el qual guardarà y gestionarà els diferents rols d'usuari i continguts de cada una de les webs que s'hi connectin.

Crear un sistema d'actualització de les pàgines webs externes, amb la finalitat d'establir connexions segures entre el servidor gestor dels continguts i les pàgines web externes.

Crear una pàgina web que es connecti al CMS creat per tal d'obtenir el contingut i els usuaris, tot això des d'un servidor extern al gestor de continguts.

Crear una interfície, la qual permetrà connectar-se y gestionar les diferents opcions emmagatzemades y processades al servidor central.

Crear una interfície en la qual l'usuari administrador pugui editar i gestionar el contingut i la seva estructura de forma responsiva, sense coneixements de programació ni de maquetació.

Crear una interfície on l'usuari administrador pugui crear i eliminar les pàgines que contindrà la web.

1.3 ENFOCAMENT I MÈTODE SEGUIT

En aquest projecte, degut a les especificacions i característiques, s'ha optat per la creació d'un sistema totalment nou a través de les metodologies àgils de desenvolupament de programari, Scrum per l'organització i planificació, i Arquitectura modular per l'estructura del sistema.

D'aquesta manera, s'assumeix que el disseny de les classes, dels components, les funcionalitats, els serveis i les especificacions de requisits es un procés iteratiu a través de tot el desenvolupament del projecte. D'aquesta manera, no es necessari esperar a la finalització del disseny de tota la plataforma per començar a programar-la.

L'elecció de la metodologia Scrum es justifica pel fet de que ens permet tenir en tot moment un producte estable i funcionant, implementant funcionalitats de forma recursiva a mesura que les necessitem. En aquest cas resulta molt interessant pel fet de que és molt probable que es vagin afegint i eliminant funcionalitats a mesura que el projecti vagi creixent.

Pel que fa a l'arquitectura del programari, la modularització en el nostre cas, és complementa perfectament amb la metodologia organitzativa escollida, ja que cada un dels casos d'ús que es generin es podran desenvolupar com un nou mòdul de forma independent i aïllat de la resta, afavorint l'extensió del sistema i evitant la modificació del producte actual.

La combinació d'aquestes metodologies ens permet crear un producte sense la necessitat de tenir les especificacions finals a les primeres fases de disseny, així doncs, sempre que es vulgui afegir un nou requisit al sistema s'haurà de seguir el mateix procés iteratiu, passant per totes les fases de desenvolupament de programari conegudes i esmentades anteriorment.

1.4 PLANIFICACIÓ DEL TREBALL

- **PAC1 – Pla de treball. (21/09 – 04/10)**
    - Elaboració del títol i el índex.
    - Context i justificació del Treball.
    - Objectius del Treball.
    - Enfocament i mètode seguit.
    - Planificació del Treball.
- **PAC2 – Disseny i arquitectura. (05/10 – 02/11)**
    - Disseny
        - Anàlisis dels primers requisits bàsics
        - Casos d'ús generats a partir dels requisits.
        - Disseny del model de domini.
        - Creació i disseny dels primers sprint a realitzar a al següent fase del projecte.
        - Desenvolupament dels Test a passar.
    - Arquitectura
        - Tecnologies i estructura del Client.
        - Tecnologies i estructura del Servidor.
        - Tecnologia i estructura de la Base de Dades.
        - APIs / llibreries de tercers utilitzades.
- **PAC3 – Desenvolupament Servidor (03/11 – 04/12)**
    - Desenvolupament CMS
        - Configuració inicial (Seed) del projecte.

        - Creació i gestió dels usuaris i els seus rols.
        - Procés d'iniciació i establiment de connexió.
        - Creació de les col·leccions de la base de dades.
        - Mètodes gestors de la base de dades.
        - Configuració del sistema d'emails.
- **PAC4 – Desenvolupament Client (05/12 – 09/01)**
    - Creació de l'interfície d'administració al servidor central.
        - Configuracions generals de la pàgina web.
        - Llistar, editar i gestionar els continguts de la pàgina web.
        - Eliminar i crear noves seccions a la pàgina web.
    - Creació pàgina web pública
        - Generador de contingut a partir de l'informació rebuda del CMS.
        - Generar les pàgines dinàmicament a partir de l'informació rebuda del CMS.
    - Memòria
    - Presentació virtual

1.4.1 Planificació del treball.

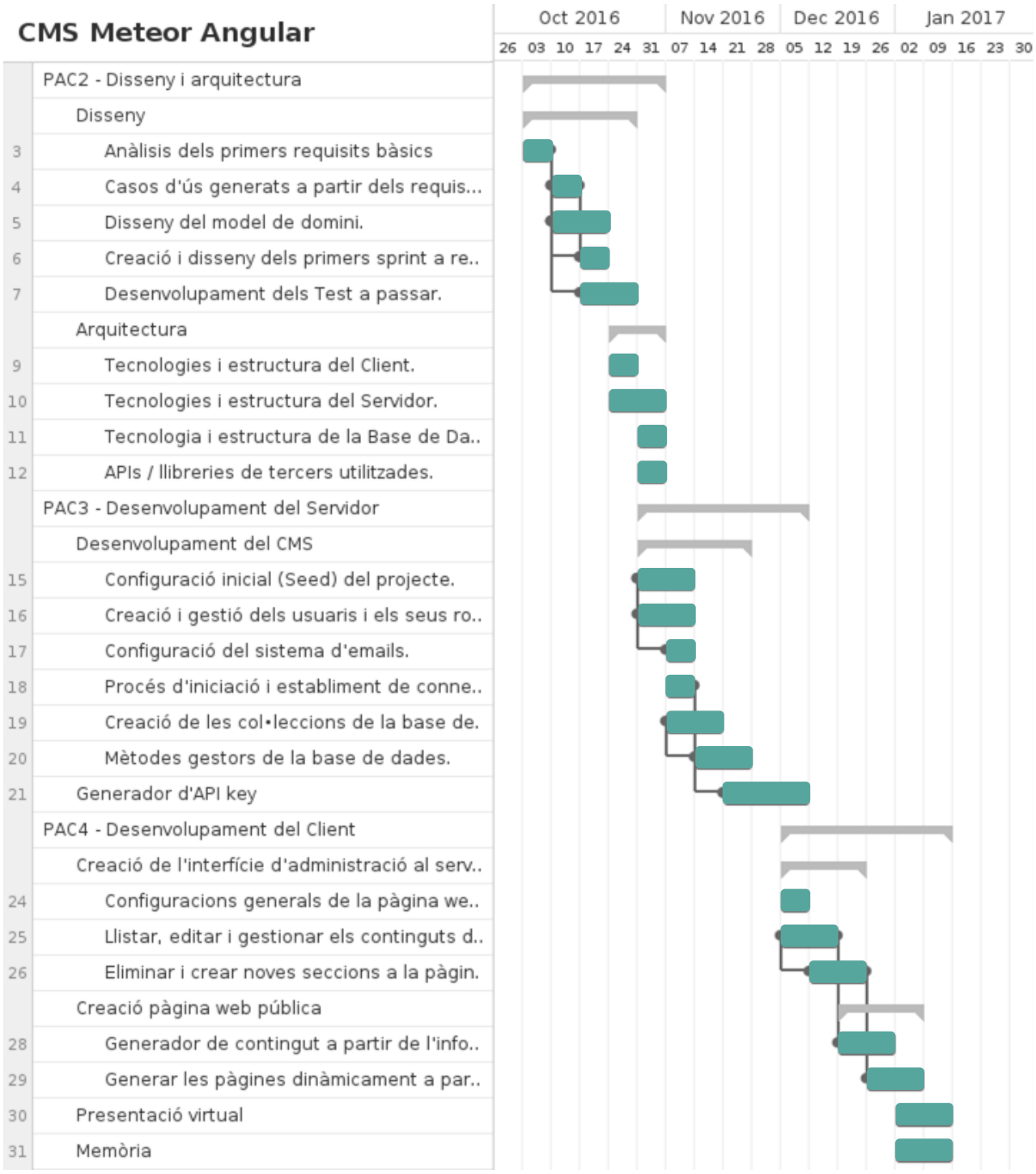


Figura 1: diagrama de Gantt

Font: elaboració pròpia

2. DISSENY

2.1 REQUISITS

A continuació s'exposen els requistis, els quals intenten expressar les necessitats i restriccions que afecten al nostre producte de programari, en aquest cas el gestor de continguts i tota la seva infraestructura. A més de permetre delimitar quines de les possibles solucions al problema són més adequades (les que compleixen els requisits) i quines no ho fan.

Aquestes necessitats i restriccions han estat generades a partir de la creació d'un perfil d'un stakeholder, el qual el seu principal objectiu i interès és el de la creació d'una pàgina web personal a partir d'un gestor de continguts.

Així doncs, l'especificació d'aquests requisits recull totes aquelles especificacions acordades entre el perfil de stakeholder i els desenvolupadors de la plataforma, permetent generar una eina de comunicació per tots dos grups, esdevenint l'element central a partir del qual es generen totes les preses de decisions i d'on emergeixen els dissenys de les posteriors etapes del projecte.

2.1.1 Requisits no funcionals.

Els requisits no funcionals fan referència a les restriccions que afecten al conjunt de possibles solucions, afectant majoritàriament a gran part del sistema i la seva infraestructura.

	CODI	DESCRIPCIÓ
NO FUNCIONALS	**1.1**	La pàgina web ha d'estar allotjada a un servidor diferent del gestor de continguts.
	1.2	L'administrador no haurà de configurar el sistema DNS, SSH o FTP del seu servidor. Només haurà de tenir instal·lat Apache i una versió superior o igual a 5.4 de PHP.
	1.3	Quan l'administrador vulgui accedir a la secció d'edició, el client redirigirà l'usuari al servidor central amb el CMS amb una id (per tal de mostrar informació personalitzada en el login, com logo, nom, etc) on seguidament haurà de fer login per poder editar el contingut.

	1.4	Totes les pagines webs estaran gestionades pel mateix CMS.
	1.5	Els arxius html, css i javascript que composaran la pàgina web externa seran generats quan es vagi a actualitzar, per posteriorment eliminar-los, amb l'objectiu d'estalvi de memòria.
	1.6	Quan es vulgui actualitzar la seva web amb els canvis, el CMS a d'enviar una senyal al servidor extern, fent que aquest descarregui i actualitzi els arxius font.
	1.7	La pàgina web del servidor extern no utilitzarà cap framework MVC, però haurà d'estar preparada per la seva possible inclusió en un futur.
	1.8	Cada un dels components del gestor de continguts estaran comunicats entre ells a través d'una arquitectura modular.

2.1.2 Requisits funcionals.

Els requisits funcionals tenen com a objectiu descriure quines funcionalitats ha de proporcionar el sistema, i com aquest ha de reaccionar als diferents estímuls que rep de l'exterior.

	CODI	DESCRIPCIÓ
FUNCIONALS	**2.1**	L'administrador ha d'identificar-se per accedir a l'interfície d'edició i gestió de continguts.
	2.2	Des de l'interfície d'edició s'ha de poder crear, modificar i eliminar les pàgines, els continguts i les opcions generals, com la IP del servidor extern, a més de l'opció de refrescar la web externa.
	2.3	S'ha de poder estructurar cada una de les pàgines per tal de crear un sitemap personalitzat.
	2.4	Cada una de les seccions d'una pàgina ha de poder ser editada de forma independent, a més de poder guardar-la com a plantilla per utilitzar-la posteriorment.
	2.5	Les seccions d'una pàgina es poden guardar de forma privada o pública, funció de si es desitja que altres usuaris puguin utilitzar-les.
	2.6	Tots els components d'una secció han de tenir la possibilitat d'editar el seu tamany per tal de crear un grid personalitzat.
	2.7	A cada un dels components d'una secció es podrà elegir quin element s'hi vol inserir: Text, imatge, o text d'encapçalament.

	2.8	Haurà d'existir la possibilitat de cada component, secció i pàgina tingui el seu propi css.
	2.9	Els menús de navegació (encapçalaments) i els pues de pàgina podran elegir amb quin format i disseny es mostra
	2.10	Quan es realitzi el procés d'actualització, s'haurà d'informar a l'usuari de cada una de les fases i l'estat actual d'aquest.

2.2 CASOS D'ÚS.

Per tal de mantenir una documentació dels requisits s'utilitzen els casos d'ús, que permeten fer servir diferents graus de detall i formalismes amb la finalitat de descriure diversos escenaris, els quals es donen entre el sistema i els stakeholders en un moment, acció i procés determinat, a més de detallar el comportament observable del sistema.

El stakeholder que realitza la funció d'actor principal és el perfil d'administrador el qual té com a objectiu la creació d'una pàgina web.

2.2.1 Cas d'ús: Accedir a la pàgina d'administració del sistema gestor de continguts.

Un usuari intenta entrar a la pàgina d'administració del sistema de gestor de continguts, i el sistema ha de comprovar si aquest té permisos per accedir-hi. En cas que l'usuari pugui accedir se li atorgarà una identificació i se li mostraran exclusivament els continguts generats per aquest usuari o tot el que sigui públic.

CU001	**ACCÉS A LA PÀGINA D'ADMINISTRACIÓ DEL CMS.**	
Actor principal	Usuari no identificat.	
Requisits associats	• 1.3 Personalització de la pàgina d'accés. • 1.4 Tots els continguts estaran gestions pel mateix CMS. • 2.1 Identificació de l'usuari com a administrador.	
Escenari principal	**Pas**	**Acció**
	1	L'usuari accedeix a l'apartat d'administració en el seu lloc web, en el servidor extern, a través de la url "/admin".
	2	El servidor extern redirecciona l'usuari cap a la pàgina

		d'identificació en el servidor CMS on es mostrarà informació personalitzada per l'usuari, transmesa a la capçalera de la petició.	
	3	L'usuari introdueix les dades d'identificació i clica a iniciar sessió.	
	4	El sistema comprova que l'usuari existeix i la contrasenya és correcte	
		4a	Si la identificació es correcta redirigeix l'usuari a la pàgina d'administració.
		4b	Si la identificació és incorrecte es retorna un missatge d'error informant del problema
Excepcions	**Pas**	**Acció**	
	2	En cas que el servidor no pogués redireccionar l'usuari, se li facilitaria la url perquè hi accedís manualment.	
Freqüència	Desconeguda		
Importància	Vital		
Urgència	Immediatament		

2.2.2 Cas d'ús: Crear una secció d'una pàgina de forma independent.

L'administrador d'una pàgina web vol crear seccions modulars i reutilitzables que puguin ser inserides posteriorment a una pàgina de la web. A més de poder-hi introduir qualsevol tipus de contingut i editar el seu estil i posició.

CU002	**CREACIÓ DE LAYOUT.**	
Actor principal	Administrador d'una pàgina.	
Requisits associats	• 1.8 Arquitectura Modular. • 2.2 Crear, modificar i eliminar qualsevol tipus de contingut. • 2.4 Crear i guardar les seccions de forma independent. • 2.5 Guardar les seccions de forma pública o privada. • 2.6 Tots els components d'una secció han de ser editables. • 2.7 Escollir el tipus de component de cada secció. • 2.8 Tots eles elements han de poder tenir el seu propi CSS.	
Escenari principal	**Pas**	**Acció**
	1	L'usuari accedeix a l'apartat anomenat Layouts.
	2	L'usuari selecciona si desitja crear una nova secció o si vol editar una guardada anteriorment.
		2a Si selecciona l'opció de crear, es mostrarà l'editor de

			Layouts sense cap contingut.
		2b	Si selecciona l'opció d'editar una secció existent, es mostrarà l'editor amb tots els continguts inserits prèviament.
	3	L'usuari introdueix o modifica el nom de la secció.	
	4	L'usuari selecciona quins elements vol afegir a la secció: Text, imatges, encapçalaments o d'altres.	
	5	L'usuari ajusta el tamany de cada un dels components, a més de poder modificar el seu ordre, sense importar quan han sigut inserits.	
	6	L'usuari guarda la secció a la base de dades per tal de poder tenir-la disponible des de l'editor de pàgines (cas d'ús CU003).	
Excepcions	**Pas**	**Acció**	
	6	En cas d'existir un Layout prèviament guardat i creat per aquest usuari amb el mateix nom, retornarà un missatge informatiu i no guardarà l'element a la base de dades.	
Freqüència	Desconeguda		
Importància	Vital		
Urgència	Immediatament		

2.2.3 Cas d'ús: Crear una pàgina

El sistema ha de permetre que l'administrador creí pàgines a partir de les seccions creades posteriorment o bé de noves seccions que no es guardaran a la base de dades com a plantilles.

CU003	**CREACIÓ DE PÀGINA.**		
Actor principal	Administrador d'una pàgina.		
Requisits associats	• 1.7 La pàgina web externa no utilitzarà cap framework. • 1.8 Arquitectura Modular. • 2.2 Crear, modificar i eliminar qualsevol tipus de contingut. • 2.8 Tots eles elements han de poder tenir el seu propi CSS.		
Escenari principal	**Pas**	**Acció**	
	1	L'usuari accedeix a l'apartat anomenat Pages.	
	2	L'usuari selecciona si desitja crear una nova pàgina o si vol editar una guardada anteriorment.	
		2a	Si selecciona l'opció de crear, es mostrarà l'editor de

			pàgines sense cap contingut.
		2b	Si selecciona l'opció d'editar una secció existent, es mostrarà l'editor amb totes les seccions i continguts inserits prèviament.
	3	L'usuari introdueix o modifica el nom de la pàgina.	
	4	L'usuari decideix introduir una nova secció a la pàgina.	
		4a	Si vol inserir una nova secció, el sistema afegirà un nou editor de seccions sense cap contingut
		4b	Si vol inserir una secció creada i guardada posteriorment com a plantilla, se li obrirà un finestra i podrà seleccionar de forma múltiple quines vol afegir-hi.
	5	L'usuari pot ajustar, modificar i editar qualsevol element d'una secció tal i com es feia en el cas d'ús CU002.	
	6	L'usuari guarda la pàgina a la base de dades per tal de poder tenir-la disponible des de l'editor del sitemap (CU004).	
Excepcions	**Pas**	**Acció**	
	6	En cas d'existir una pàgina prèviament guardada i creada per aquest usuari amb el mateix nom, retornarà un missatge informatiu i no guardarà l'element a la base de dades.	
Freqüència	Desconeguda		
Importància	Vital		
Urgència	Immediatament		

2.2.4 Cas d'ús: Edició de l'estructura jeràrquica de les pàgines

El sistema ha de permetre que l'administrador modifiquí l'estructura jeràrquica de les pàgines, en altres paraules les pàgines han de poder niar d'altres.

CU004	**EDICIÓ DEL SITEMAP**	
Actor principal	Edició de l'estructura jeràrquica de les pàgines.	
Requisits associats	• 1.7 La pàgina web externa no utilitzarà cap framework. • 1.8 Arquitectura Modular. • 2.2 S'ha de poder generar un sitemap personalitzat. • 2.9 Seleccionar el format de l'encapçalament i/ peu de pàgina.	
Escenari principal	**Pas**	**Acció**
	1	L'usuari accedeix a l'apartat anomenat Sitemap.

	2	L'usuari ordena (agafant i arrastrant) les pàgines.	
		2a	Pot modificar el seu ordre dins d'un mateix nivell de jerarquia.
		2b	Pot niar pàgines dins d'una altre de forma infinita i sense cap tipus de restricció.
	3	L'usuari pot seleccionar quin encapçalament i quin peu de pàgina tindrà cada una de les seccions.	
		3a	Si vol tenir un encapçalament i un peu de pàgina igual per totes les pàgines ho ha de poder fer assignant-ne un de forma predeterminada.
		3b	Si vol que a una pàgina en concret tingui un encapçalament i peu de pàgina diferent a la resta ho farà a través de l'icona d'edició, on li apareixerà una finestra amb les possibilitats.
	4	L'usuari guarda la distribució de les pàgines a la base de dades per tal de que el compilador i creador de la pàgina web l'utilitzi com a referencia (CU006).	
Freqüència	Desconeguda		
Importància	Vital		
Urgència	Immediatament		

2.2.5 Cas d'ús: Pujar una imatge

El sistema ha de permetre que l'administrador pugi imatges per poder ser posteriorment inserides a qualsevol component.

CU005	**PUJAR UNA IMATGE**	
Actor principal	Administrador d'una pàgina i sistema gestor de continguts.	
Requisits associats	• 1.8 Arquitectura Modular. • 2.2 Crear, modificar i eliminar qualsevol tipus de contingut. • 2.7 Escollir el tipus de component de cada secció.	
Escenari principal	**Pas**	**Acció**
	1	L'usuari accedeix a l'apartat anomenat Media.
	2	L'usuari selecciona quina imatge desitja pujar al servidor i li assigna un nom, una descripció i si és pública o privada.

	3	L'usuari decideix afegir un component de tipus imatge a una secció de pàgina on s'obrirà una nova finestra.	
		3a	Si vol pujar una nova imatge, el sistema li oferirà l'opció de afegir una imatge que nova i no existent a al sistema. Mateix procés que CU0005 pas 2.
		3b	Si l'usuari vol afegir una imatge prèviament pujada o compartida per algun altre usuari, se li obrirà una finestra amb un llistat amb totes les imatges.
	4	L'usuari selecciona i afegeix la imatge al component creat prèviament.	
Excepcions	**Pas**	**Acció**	
	2, 3b	En cas de que l'arxiu no sigui del format corresponent retornarà un missatge informatiu i eliminarà l'arxiu temporal.	
	2, 3b	En cas de que l'arxiu sobrepassi el límit de pes permès retornarà un missatge informatiu i eliminarà l'arxiu temporal.	
	2, 3b	En cas de que l'arxiu se li assigni un nom ja existent retornarà un missatge informatiu.	
Freqüència	Desconeguda		
Importància	Important		
Urgència	Hi ha pressió		

2.2.6 Cas d'ús: Actualitzar pàgina web externa

El sistema haurà de permetre a l'usuari actualitzar en qualsevol moment la seva pàgina web externa amb el nou contingut generat des de l'interfície d'edició de continguts.

CU006	**Actualitzar pàgina web**
Actor principal	Administrador d'una pàgina i sistema gestor de continguts.
Requisits associats	• La pàgina web i el CMS s'allotjaran a servidors diferents • 1.2 L'usuari no haurà de configurar DNS, SSH ni FTP. • 1.5 Els arxius font seran eliminats del servidor central al finalitzar el procés. • 1.6 El servidor central iniciarà la comunicació i la pàgina web s'encarregarà de baixar els arxius. • 1.7 La pàgina web externa no utilitzarà cap framework.

<table>
<tr><td></td><td colspan="2">1.8 Arquitectura Modular.
2.10 Oferir informació del procés d’actualització a l’usuari.</td></tr>
<tr><td rowspan="8">Escenari principal</td><td>Pas</td><td>Acció</td></tr>
<tr><td>1</td><td>L’usuari clicarà al símbol d’actualització de la pàgina web des de qualsevol secció de l’interfície d’edició de continguts.</td></tr>
<tr><td>2</td><td>El sistema compilarà cada una de les pàgines que l’usuari hagi creat, i les col·locarà dins de carpetes, en funció de l’estructura jeràrquica. Tot el contingut generat es col·locarà a una carpeta amb un nom generat aleatòriament.</td></tr>
<tr><td>3</td><td>El sistema enviarà un missatge a la pàgina web externa informant-la de que té nous arxius per descarregar-se, juntament amb la url per fer-ho.</td></tr>
<tr><td>4</td><td>La pàgina externa iniciarà la descarrega a la url facilitada pel sistema gestor dels continguts.</td></tr>
<tr><td>5</td><td>La pàgina realitzarà una copia de seguretat de tots els arxius anteriors i instal·larà els nous arxius descarregats.</td></tr>
<tr><td>6</td><td>La pàgina web enviarà un missatge de confirmació al servidor centrar.</td></tr>
<tr><td>7</td><td>El servidor central eliminarà la carpeta amb el projecte generat per motius de seguretat i rendiment.</td></tr>
<tr><td rowspan="3">Excepcions</td><td>Pas</td><td>Acció</td></tr>
<tr><td>3, 6</td><td>En cas de no rebre cap missatge de confirmació ni d’iniciació de descarrega, el servidor central eliminarà els arxius generats passat un determinat temps i retornarà un missatge d’error a l’usuari.</td></tr>
<tr><td>4, 5</td><td>En cas d’error en algun dels processos, la pàgina web externa enviarà un missatge al servidor central informant de l’error i retornat al pas 1.</td></tr>
<tr><td>Freqüència</td><td colspan="2">Desconeguda</td></tr>
<tr><td>Importància</td><td colspan="2">Vital</td></tr>
<tr><td>Urgència</td><td colspan="2">Inmediatament</td></tr>
</table>

2.3 MATRIU DE TRAÇABILITAT. REQUISITS CONTRA CASOS D'ÚS.

La matriu de traçabilitat de requisits és un requadre que vincula cada un dels requisits presents en el projecte des del seu origen fins a la finalització completa, amb cada un dels casos d'ús.

Aquesta matriu ajuda a assegurar que cada un dels requeriments afegeix valor al producte que s'està desenvolupant, mostrant el vincle entre els requisits, les necessitats i els objectius del projecte. D'aquesta manera es poden realitzar seguiments durant el cicle de vida de la creació del producte, assegurant que es satisfan cada una de les especificacions plantejades pels stakeholders.

En l'eix de coordenades horitzontal trobem la referència d'identificació de cada un dels casos d'ús plantejats i descrits a l'apartat anterior, per l'altra banda, a l'eix de coordenades verticals trobem la referència a tots els requisits que s'han documentat a partir de les necessitats i objectius dels stakeholders.

	C001	C002	C003	C004	C005	C006
1.1						X
1.2						X
1.3	X					
1.4	X					
1.5						X
1.6						X
1.7			X	X		X
1.8		X	X	X	X	X
2.1	X					
2.2		X	X	X	X	

2.3				X		
2.4		X				
2.5		X				
2.6		X				
2.7		X			X	
2.8		X	X			
2.9				X		
2.10						X

2.4 MODELS DEL DOMINI

A continuació es mostren diferents models de domini que modelitzen la interfície del sistema per cada un dels casos d'ús descrits anteriorment. Aquest diagrama de classes UML que representa classes conceptuals del món real en un domini d'interès utilitzant l'orientació a objectes s'ha hagut de adaptar al framework MVC utilitzat per aquest projecte, on cada una de les classes representa un mòdul amb les seves respectives vistes i controladors, i en el cas de comunicar-se amb mòduls externs ho realitzarà a través de serveis.

Totes aquelles classes que fan la funció de façana, en aquest cas, són mòduls els quals disposen d'un servei amb uns determinats mètodes que poden ser invocats des de l'exterior per altres mòduls, aquests mètodes són els encarregats posteriorment de saber quins mòduls fills utilitzar. Així doncs, cada un dels casos d'ús s'han encapsulat com a components totalment independents, que per comunicar-se uns amb els altres, és necessari fer-ho a través dels mòduls façana.

2.4.1 LayoutManager UML

Aquest primer mòdul tindrà la principal funció d'editar, crear i eliminar qualsevol secció d'una pàgina, tal i com es descriu el cas d'ús *CU002 Creació de layout*.

Tal i com es descriu en el cas d'ús amb el que està relacionat aquest mòdul, les seves principals funcions i responsabilitats seran les de crear diferents components, gestionar els estils, crear una instancia seva, editar el seu contingut i exposar totes les instàncies Layout generades i guardades per l'usuari.

Així doncs, els components que pot crear aquest mòdul es representen en el sistema amb el nom de childrenLayout, el qual és un mòdul abstracte el qual han d'implementar tots aquells components que es vulguin afegir a una secció d'una pàgina. En aquesta cas ho podem observar a través de totes les classes que hereten de ChildrenLayout, fent que aquest mòdul només s'hagi de preocupar de quin tipus de component vol l'usuari, sense la necessitat de saber com està creat o quines són les seves dependències.

Per altra banda, tenim el mòdul que gestiona els estils dels components i de les seccions, aquest està creat a partir d'un servei el qual s'hi poden comunicar tots els altres mòduls, de manera que s'abstrau tota la lògica de les propietats CSS i es desacobla de tots els altres elements. Aquest servei pot ser utilitzat des de l'exterior a través del mòdul façana de les seccions d'una pàgina, amb la finalitat d'atorgar la funcionalitat d'estil a altres elements, com les pàgines.

Finalment tenim les funcionalitats de creació, edició i llistat, les quals s'han encapsulat en mòduls diferents per tal d'optimitzar el rendiment de l'aplicació, a diferencia del diagrama, que mostra el Layout com una única classe. Aquesta optimització està basada en el fet de que el mòdul de llistat necessita tenir contínuament accés a la base de dades, fent que si l'usuari vulgues crear o editar una secció d'una pàgina hagués d'estar consultant a la base de dades informació de la qual no necessita. Per tant, tenint en compte el principi de segregació d'interfícies, s'ha decantat per l'opció de tenir diferents interfícies específiques en comptes d'una interfície de propòsit general.

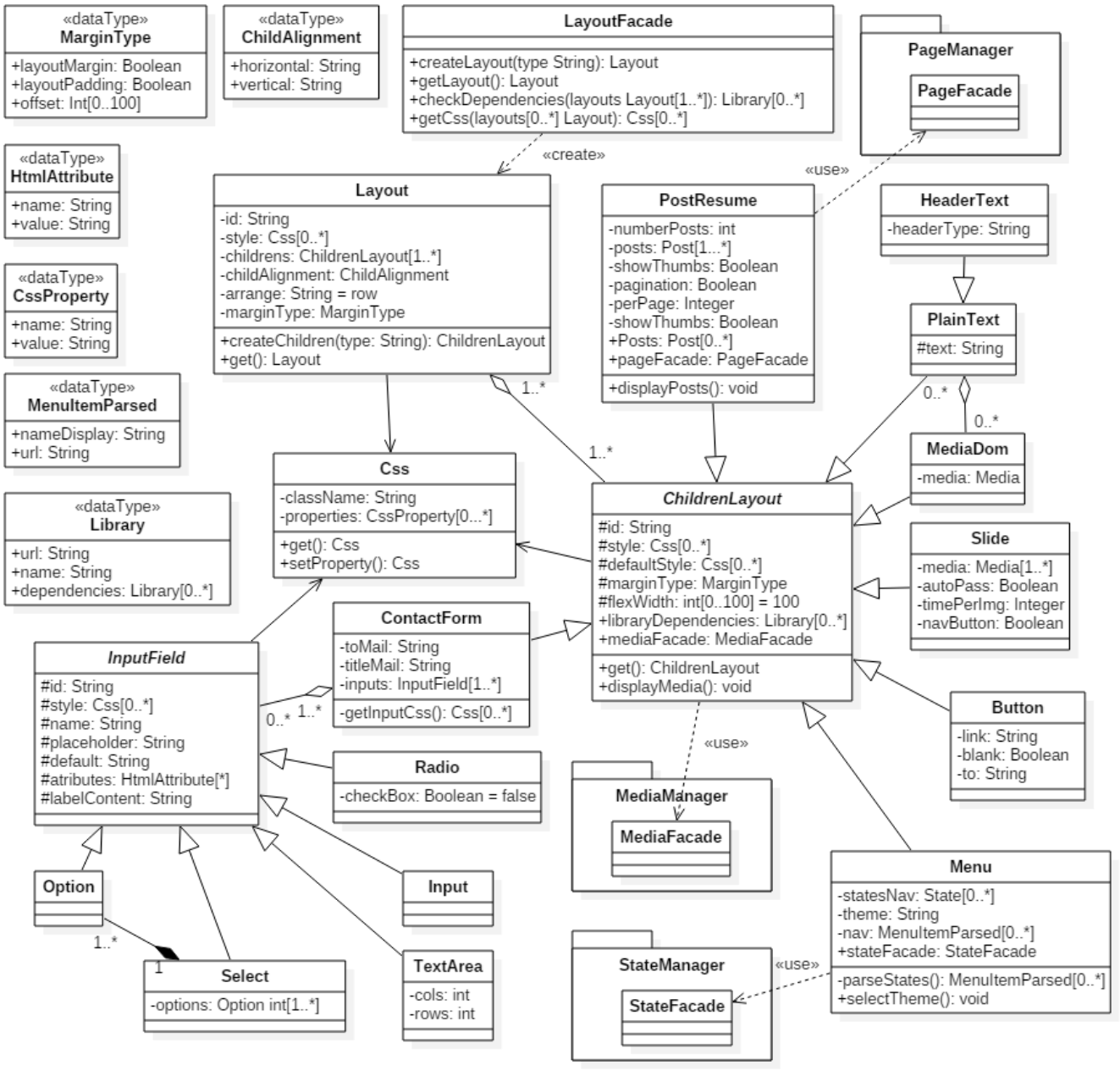


Figura 2: Diagrama de domini LayoutManager

Font: elaboració pròpia

2.4.2 PageManager uml

El mòdul de continuació està centrat en el cas d'ús *CU003 Creació d'una pàgina*, el qual té com a funció principal la creació de pàgines a partir dels mòduls de seccions (Layouts) anomenats anteriorment. Com es pot observar aquest servei té una dependència clara sobre el servei anterior, tal i com s'argumenta a continuació.

Per una banda, tenim que el mòdul de Creació de pàgines ha de poder llistar totes aquelles seccions que han estat creades, així doncs, necessita comunicar-se amb el servei de seccions per tal de poder llistar els diferents Layouts guardats a la base de dades.

Per altra banda, tal i com s'esmentava en el cas d'ús l'usuari ha de tenir la possibilitat de crear seccions d'una pàgina sense la necessitat de guardar-les com a plantilles, així doncs, aquest mòdul requerirà de la funcionalitat de creació de seccions present en el servei de Layouts, el qual instanciarà un objecte de secció i el retornarà a la pàgina, aquest objecte mantindrà totes les seves funcionalitats, com la de creació de components fills i la d'inserció d'estils.

A més a més, per tal de facilitar l'edició de l'estructura de la pàgina (sitemap), serà necessari que l'usuari disposi d'una llistat de les diferents pàgines que ha creat, per tant, mantenint el principi de segregació d'interfícies descrit anteriorment, la classe Page representada com a una únic model s'ha separat en funció de les responsabilitats de les que disposava, creació, edició i llistat.

Finalment, com es pot observar s'ha tingut en compte el disseny d'aquest mòdul per tal de poder integrar en un futur la generació d'entrades de blogs a partir de l'herència de totes les funcionalitats d'una pàgina, d'aquesta manera, tots els mòduls que vulguin crear una pàgina o una entrada nova només s'hauran de preocupar per quin tipus volen rebre, sense tenir en compte com ho han de fer, o quines són les seves funcionalitats internes.

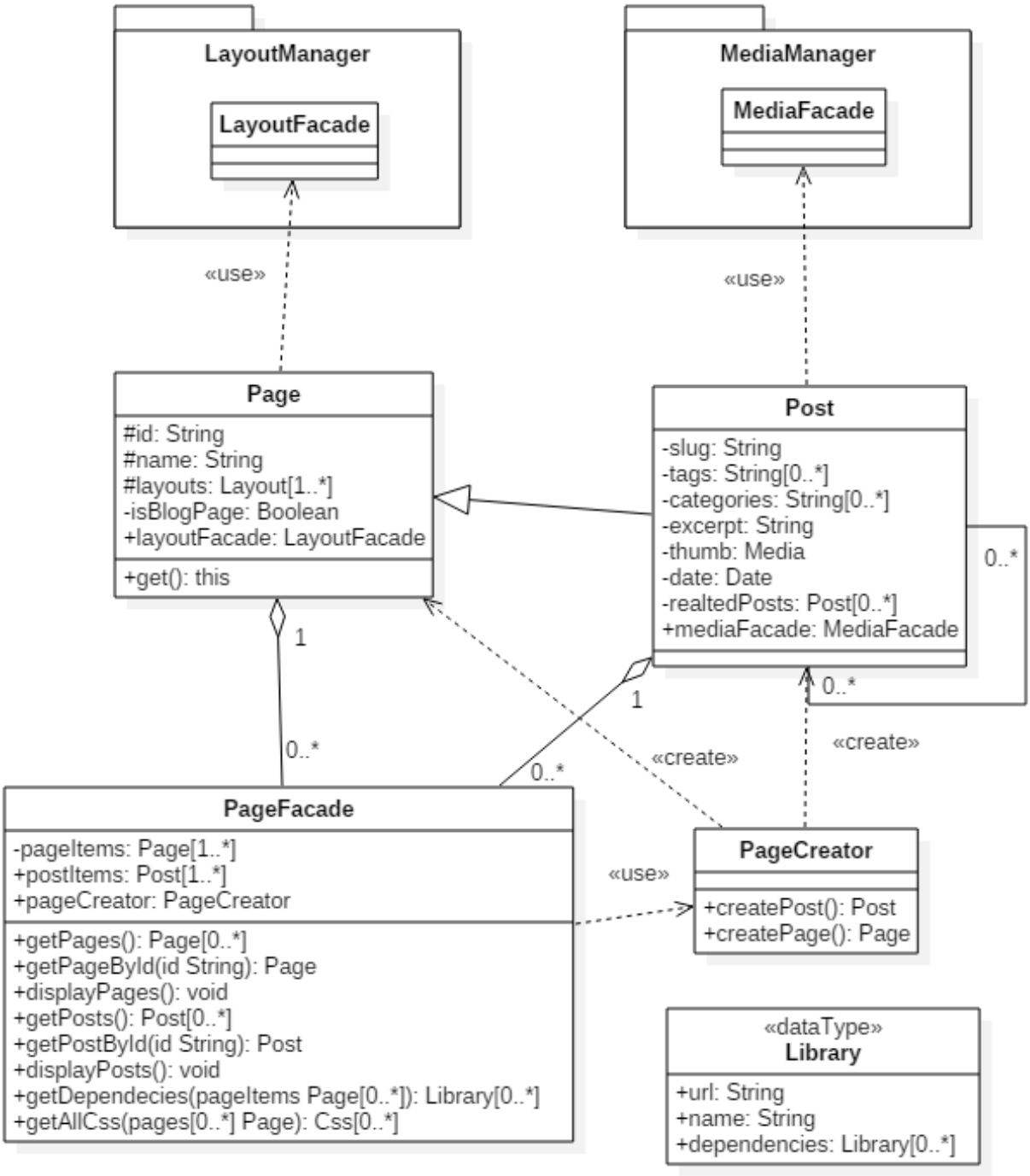


Figura 3: Diagrama de domini PageManager

Font: elaboració pròpia

2.4.3 StateManager UML

Aquest mòdul està relacionat amb els requisits presents en el cas d'ús *CU004 Edició del Sitemap* el qual té com a funció principal la modificació i personalització dels les diferents pàgines que conformen el lloc web de l'usuari. Les seves principals funcions són les de permetre obtenir l'informació de les diferents pàgines de l'usuari, per posteriorment modificar i emmagatzemar el seu ordre o com unes pàgines nien d'altres.

Com es pot observar en el diagrama, aquest mòdul té diferents dependències, principalment requereix del servei del mòdul de pàgines per tal d'obtenir totes aquelles instàncies generades i guardades per l'usuari.

Per altra banda, tal i com s'esmenta en el cas d'us *CU004 Edició del Sitemap* ha de disposa de la possibilitat d'inclusió i selecció d'encapçalaments i de peus de pàgina, aquests han de poder ser generats a partir de l'extensió i herència del mòdul de creació i edició de seccions d'una pàgina. Aquestes classes filles hauran de disposar de la possibilitat de reescriure alguns dels seus mètodes per ajustar-se a les necessitats, a més de poder instanciar components especials en funció del tipus, en altres paraules, la classe Layout ha de ser capaç de instanciar i especialitzar la classe abstracte ChildrenLayout en funció del seu tipus (Header, Footer, Other).

Finalment, seguint el patró d'objecte compost, qualsevol mòdul que utilitzi el servei de Layout, no haurà de saber ni conèixer de quin tipus de secció està utilitzant.

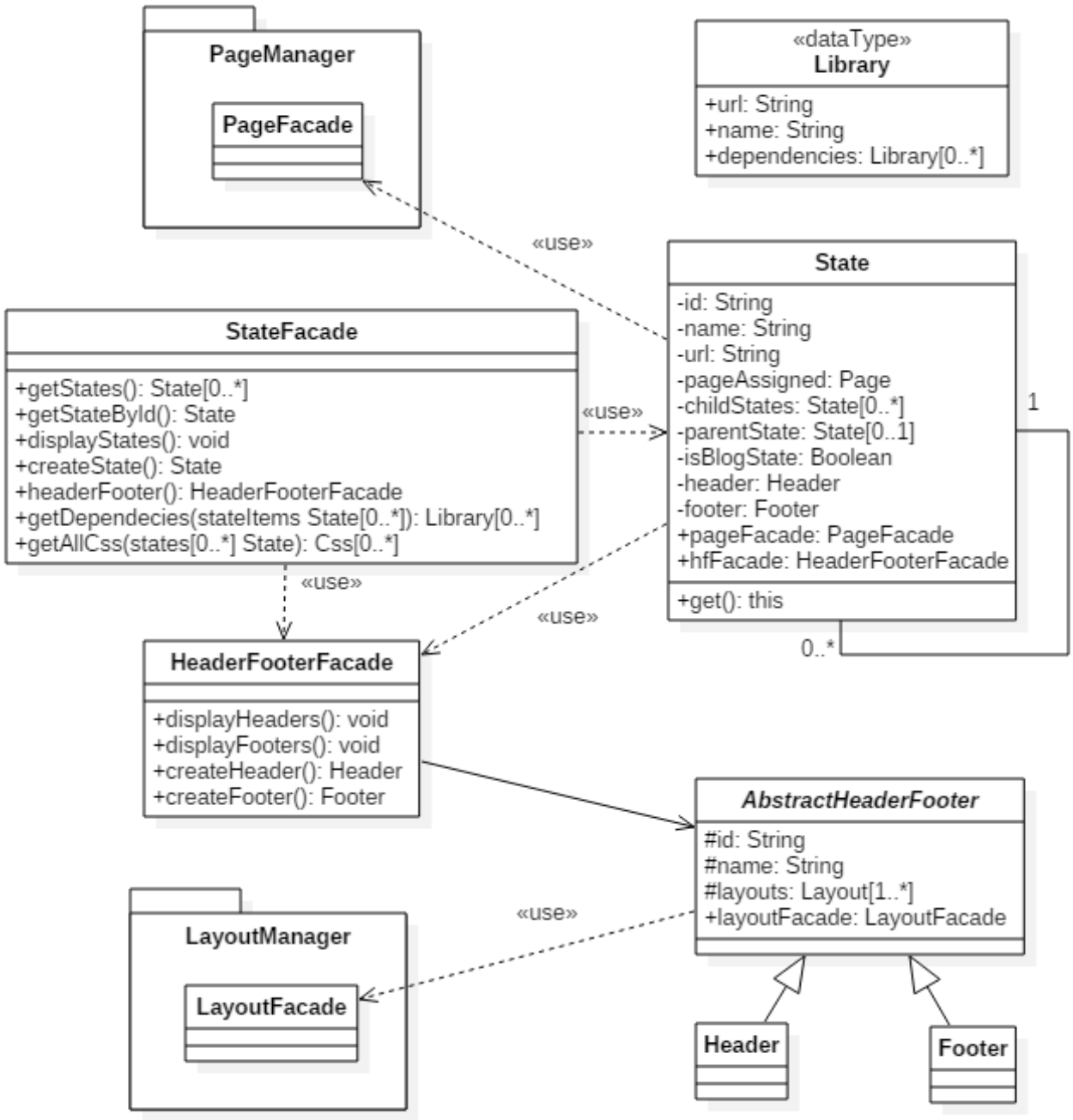


Figura 4: Diagrama de domini StateManager

Font: elaboració pròpia

2.4.4 MediaManager UML

Tal i com s'esmenta en el cas d'ús *CU005 Pujar imatges*, el següent mòduls té la responsabilitat d'atorgar al sistema de la possibilitat de pujar i gestionar les imatges que l'usuari desitgi mostrar en el seu lloc web.

Així doncs, com es pot observar en el model del domini, hi ha dos mòduls principals, encarregats de crear l'entitat en el sistema i la de gestionar l'arxiu físic. La classe Media és l'encarregada de oferir les diferents possibilitats de les que disposa l'usuari, com pujar una imatge, guardar-la i mostrar les que tingui guardades.

Per altra banda, serà necessari un servei que sigui capaç de rebre una imatge i gestionar-la per si sol, emmagatzemant-la al disc físic i crear una entitat a la base de dades fent-hi referencia. Aquest servei seria l'encarregat de crear les imatges en miniatures en un futur.

Aquest mòduls reben instruccions i peticions d'altres serveis a partir de la classe façana, a la qual utilitzen tots els altres mòduls per obtenir i gestionar les imatges del sistema.

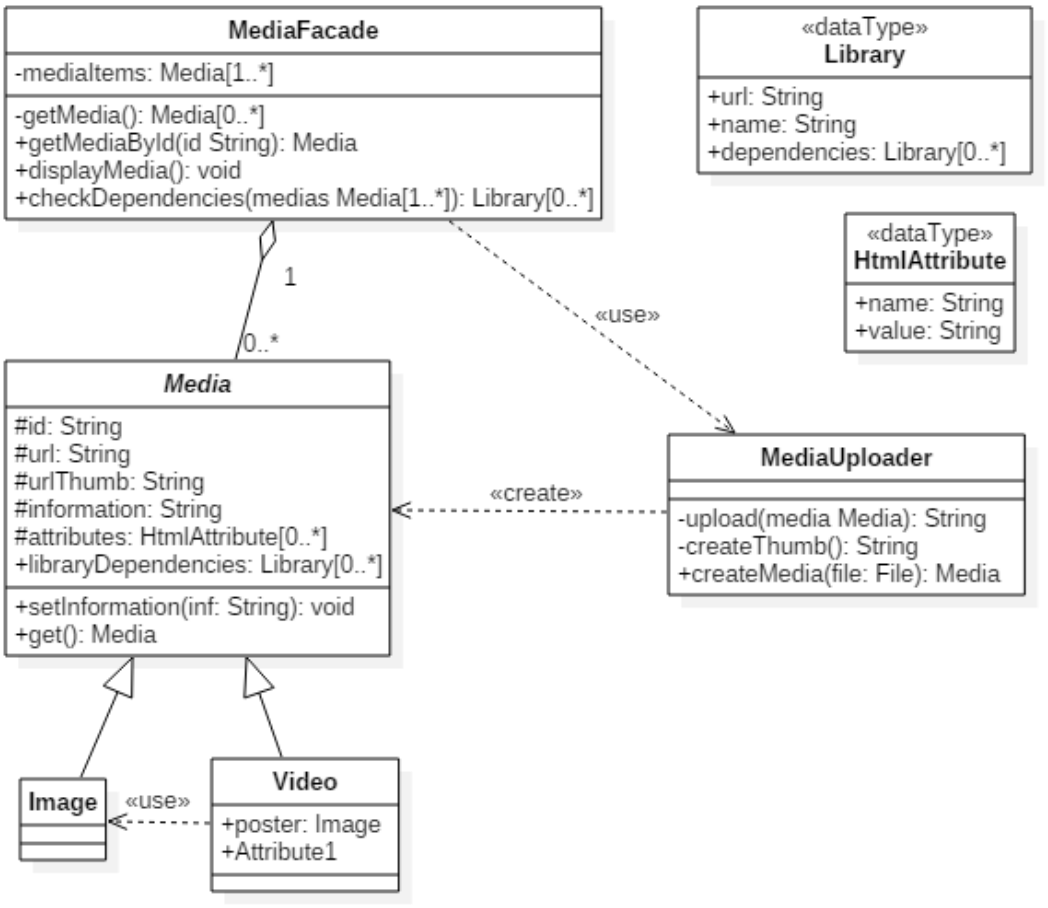


Figura 5: Diagrama de domini MediaManager

Font: elaboració pròpia

2.4.5 WebCompilatorManager UML

Aquest mòdul té la responsabilitat de crear i compilar el lloc web de l'usuari, a més de notificar a la pàgina web externa de que existeixen noves actualitzacions que hauria de descarregar, tal i com s'explica al cas d'ús *CU006 Actualitzar pàgina web.*

Aquest servei tal i com està representat en el diagrama de seqüencies d'aquest mateix apartat té la responsabilitat de captar l'esdeveniment que inicia l'usuari quan desitja actualitzar la seva pàgina web externa. El procés s'inicia enviant una petició al servei que gestiona els l'estructura jeràrquica de la pàgina, el qual retorna totes les pàgines en un objecte on es pot comprovar el seu ordre i com estan niades unes dins d'altres.

Seguidament, el compilador genera els arxius físics a partir de l'objecte retornat pel servei anterior, i els col·loca a una carpeta temporal situada a una url generada de forma aleatòria prèviament, a l'hora que crea un objecte el qual conté diferents tipus de metadades de la carpeta, el qual serà enviat a la pàgina web externa com a notificació de que s'ha d'iniciar el procés d'actualització.

La web externa rep l'objecte i inicia la descarrega a la direcció que contingui aquest objecte passat. Un cop descarregats els arxius, realitzarà una copia de seguretat i els emmagatzemarà a una carpeta de backups.

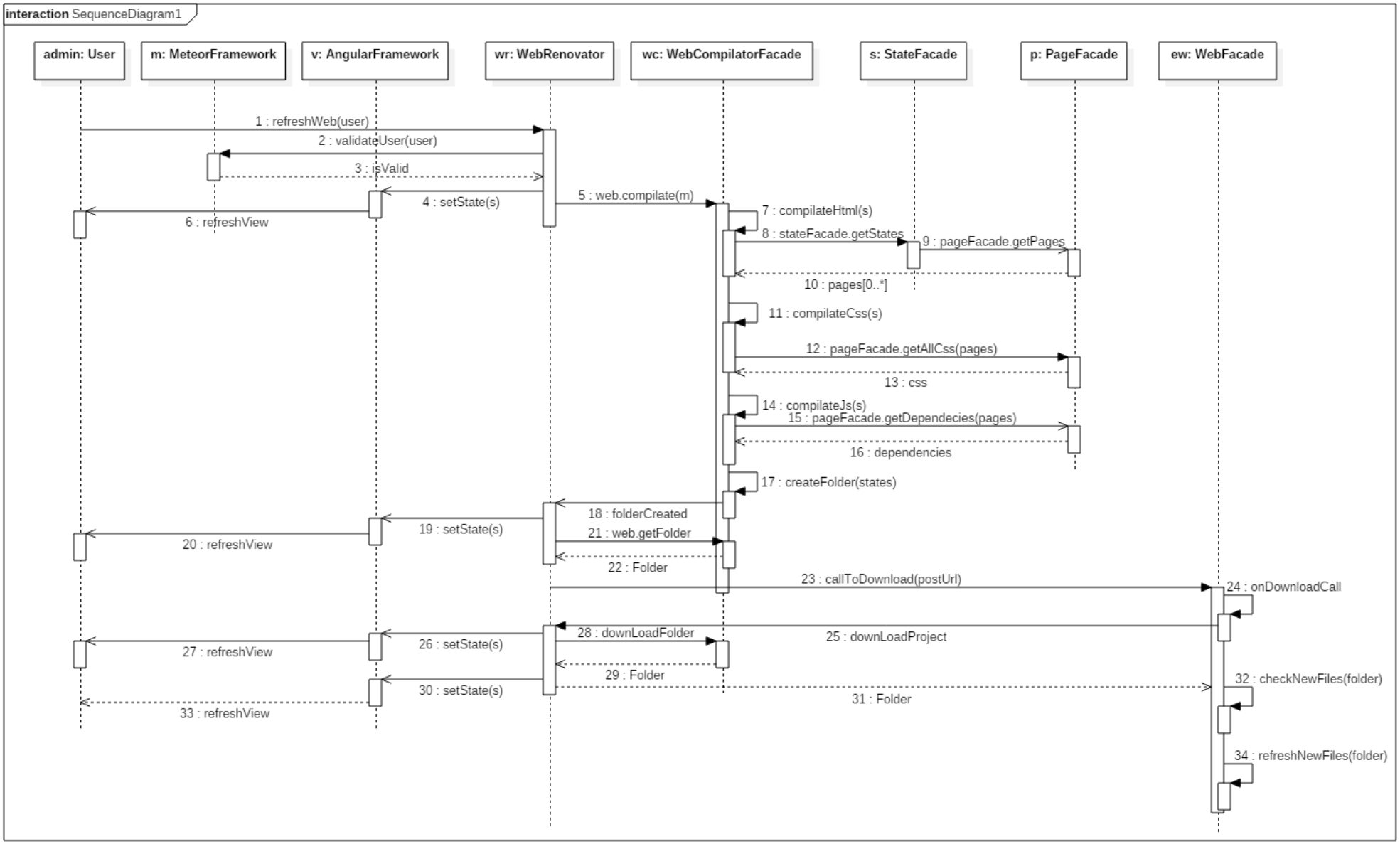


Figura 6: Diagrama de seqüències WebCompilatorManager

Font: elaboració pròpia

Finalment, la web externa enviarà una missatge de confirmació al servei façana del compilador, el qual eliminarà la carpeta temporal per mesures de seguretat i rendiment, a l'hora que enviï una notificació al controlador de les vistes de l'interfície d'edició de continguts, perquè aquest mostri un missatge de confirmació a l'usuari.

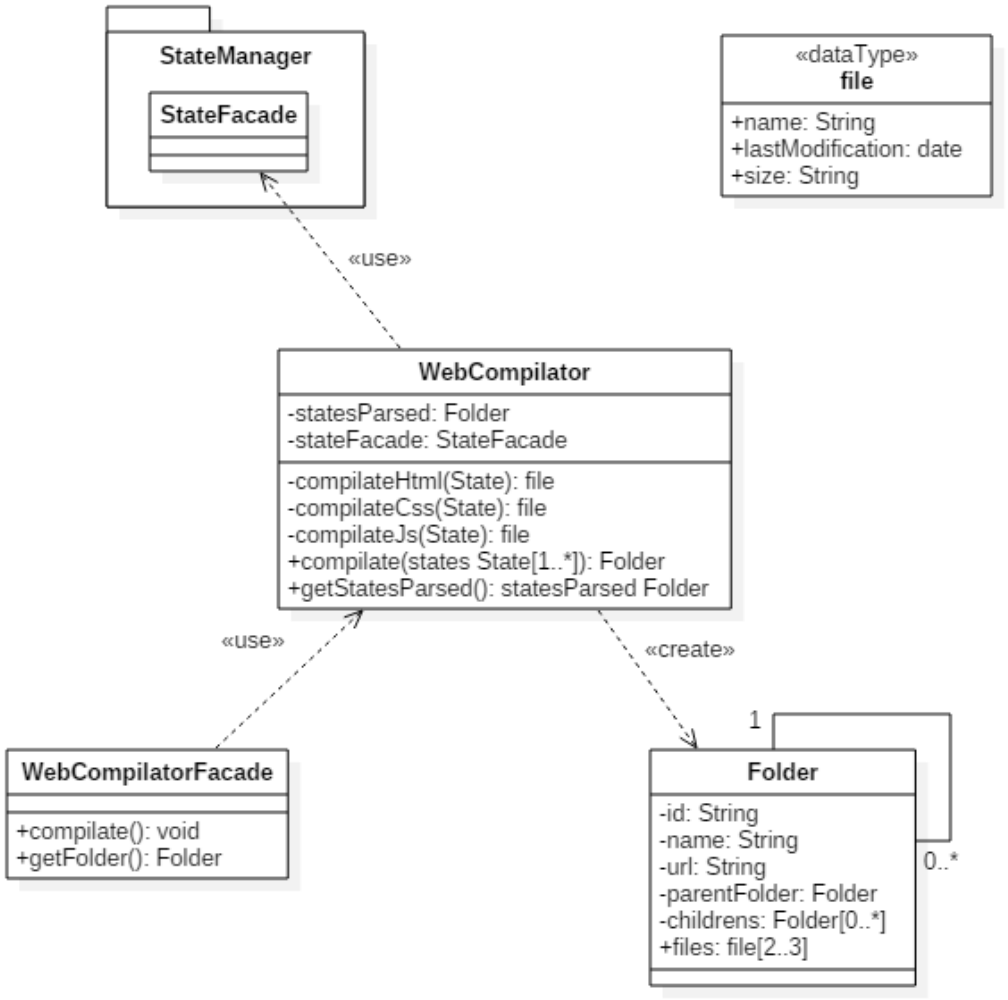


Figura 7: Diagrama de domini WebCompilatorManager

Font: elaboració pròpia

2.4.6 Model del Domini integral

En aquest model de domini es representa com cada un dels serveis es connecten entre ells, a més de la direcció de la dependència. Cada un d'aquests serveis façana té els seus mòduls i les seves vistes, les quals són gestionades per un mateix controlador, que s'encarrega de captar tots aquells estímuls de l'usuari per tal de mostrar i oferir les eines de les que requereixi i hagi demanat.

Com es pot observar, existeix un petit servei a la web externa, el qual ha de ser pujat per l'usuari a través de la connexió FTP en un inici, aquest servei no requereix de cap tipus de instal·lació per part de l'usuari, a més d'estar escrit amb llenguat PHP, facilitant a l'usuari l'instal·lació de dependències del seu servidor, ja que només serà necessari instal·lar Apache i PHP.

Finalment, dir que la comunicació entre el servidor extern i el gestor de continguts es realitzarà a través de peticions POST on l'informació anirà emmagatzemada a estructures de dades JSON.

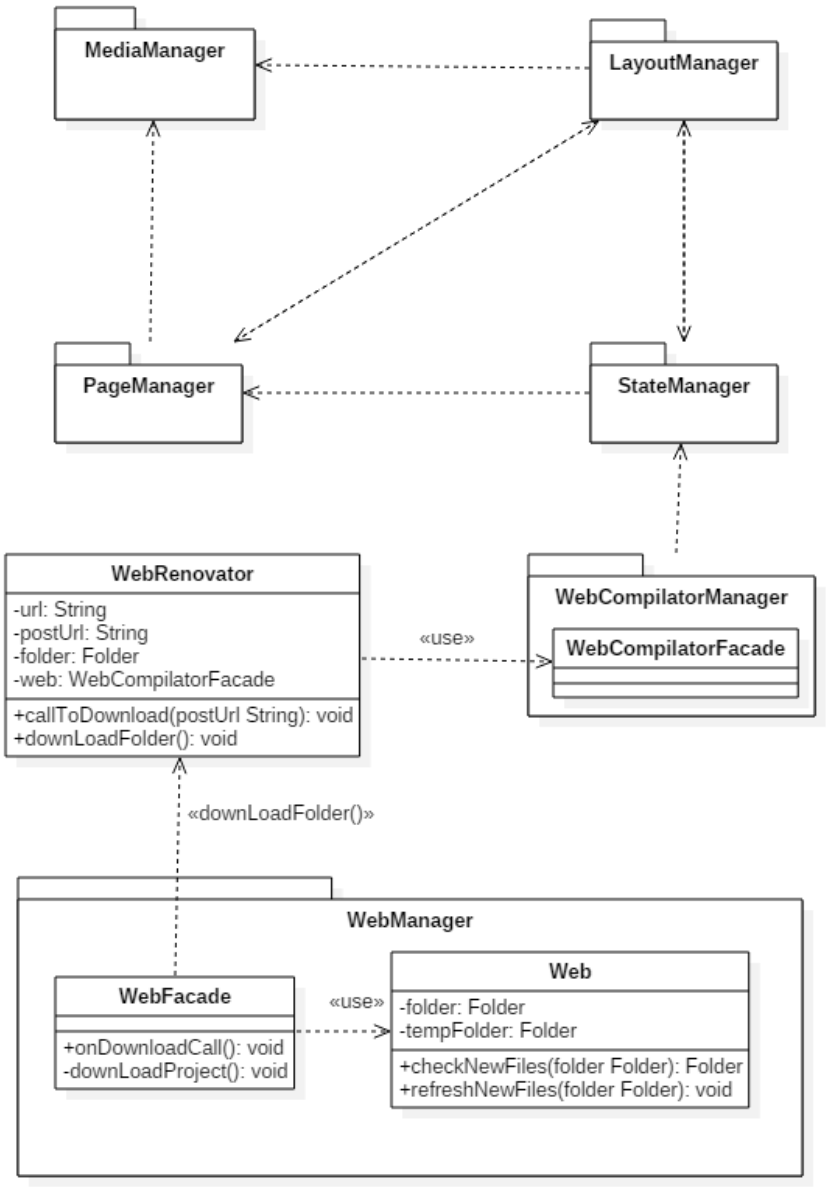


Figura 8: Diagrama de domini WebCompilatorManager

Font: elaboració pròpia

2.5 TASQUES (SPRINTS)

Per cada un dels casos d'ús esmentats a l'apartat 2.2 s'han desenvolupat uns sprint per tal de facilitar la gestió i planificació del projecte, amb l'objectiu de tenir un producte sòlid a la finalització de cada una de les etapes.

Cada un d'aquests blocs representa un cas d'ús en concret, el qual es desglossen en tasques superior a 4 hores però inferiors a 16, que s'han de realitzar per la seva total implementació en el projecte. Cada una d'aquestes noves implementacions ha de generar un increment en el producte, esdevenint-lo potencialment lliurable.

2.5.1 Accedir a la pàgina d'administració del CMS.

Referència Cas d'ús: CU001.

TASCA	REQUISITS	HORES
Sistema d'identificació d'usuaris	2.1	4
Maquetació pantalla d'identificació	1.3	4
Maquetació del Dashboard del CMS	1.4, 2.2	12
Crear el controlador principal del Dashboard	1.4	4

2.5.2 Nom: Editar l'estructura d'una pàgina.

Referència Cas d'ús: CU002.

TASCA	REQUISITS	HORES
Maquetació edició de continguts	2.2	16
Crear el component Layout	1.8, 2.5, 2.7	12
Crear el component Css	1.8, 2.8	8
Crear el component abstracte ChildrenLayout	1.8, 2.2, 2.4, 2.6	8
Crear PlainText (extend de ChildrenLayout)	2.2, 2.6, 2.7	4
Crear HeaderText (extend de PlainText)	2.2, 2.6, 2.7	4
Crear Button (extend de ChildrenLayout)	2.2, 2.6, 2.7	4
Crear MediaDom (extend de ChildrenLayout)	2.2, 2.6, 2.7	8
Crear Slide (extend de ChildrenLayout)	2.2, 2.6, 2.7	12
Crear el servei LayoutFacade	1.8, 2.2, 2.4	8

2.5.3 Nom: Crear una pàgina.

Referència Cas d'ús: CU003.

TASCA	REQUISITS	HORES
Maquetació secció de creació de pàgina	2.2	8
Crear el component Page	1.8, 2.8	16

Crear col·lecció d'objecte Page	1.7	4
Crear pageCreator	1.7, 2.2	4
Crear el servei PageFacade	1.8	12

2.5.4 Nom: Crear i editar secció d'una pàgina web.

Referència Cas d'ús: CU004.

TASCA	REQUISITS	HORES
Maquetació secció crear estats de la web	2.2, 2.3	12
Crear el component State	1.8, 2.2, 2.3	8
Crear col·lecció d'objecte State	1.7, 2.3	4
Crear el servei StateFacade	1.8	16

2.5.5 Nom: Crear i editar encapçalament i/o peu de pàgina.

Referència Cas d'ús: CU004.

TASCA	REQUISITS	HORES
Maquetació secció crear encapçalament i/o peu de pàgina	2.2	12
Crear els components Header i Footer	1.8, 2.2, 2.9	8
Crear col·leccions d'objectes Header i Footer	1.7, 2.9	4
Crear el servei HeaderFooterFacade	1.8	16

2.5.6 Nom: Pujar i seleccionar imatges.

Referència Cas d'ús: CU005.

TASCA	REQUISITS	HORES
Maquetació secció media	2.2, 2.7	4
Crear el component MediaUploader	1.8, 2.2, 2.7	16
Crear el component Media	2.7	8
Crear el servei MediaFacade	1.8, 2.7	16

2.5.7 Nom: Crear el compilador Web.

Referència Cas d'ús: CU006.

TASCA	REQUISITS	HORES
Maquetació secció compilació i descàrrega	2.10	4
Crear el component WebCompilator	1.1	16
Crear el mètode WebCompilator::compilateHtml	1.5	12
Crear el mètode WebCompilator::compilateCss	1.5	12
Crear el mètode WebCompilator::compilateJs	1.5	8
Crear el component Folder	1.1, 1.2, 1.5, 1.6	8

Crear col·lecció d'objecte Folder	1.1, 1.2	4
Crear el servei WebCompilatorFacade	1.1, 1.6, 1.8, 2.10	8

2.5.8 Nom: Actualitzar la pàgina web del servidor extern.

Referència Cas d'ús: CU006.

TASCA	REQUISITS	HORES
Crear el component WebRenovator	1.1, 1.2, 1.6, 1.8, 2.10	16
Crear el servei WebFacade al servidor extern	1.1, 1.2, 1.6	16
Crear els mètodes checkNewFiles i refreshNewFiles del servei WebFacade	1.6, 1.7	16
Crear el component de la web externa	1.1, 1.6, 1.8	8

2.6 WIREFRAMES

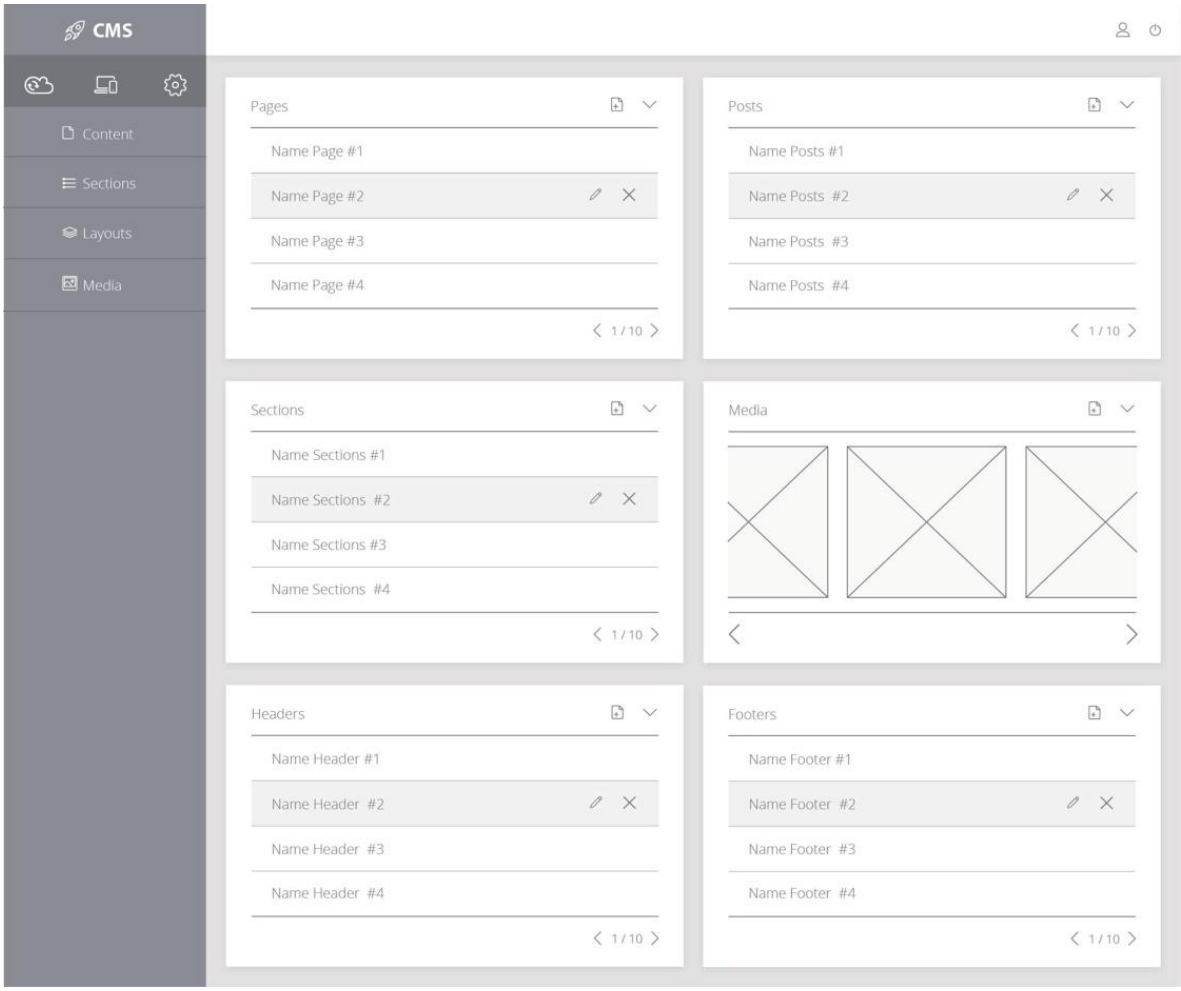


Figura 9: wireframes secció principal

Font: elaboració pròpia

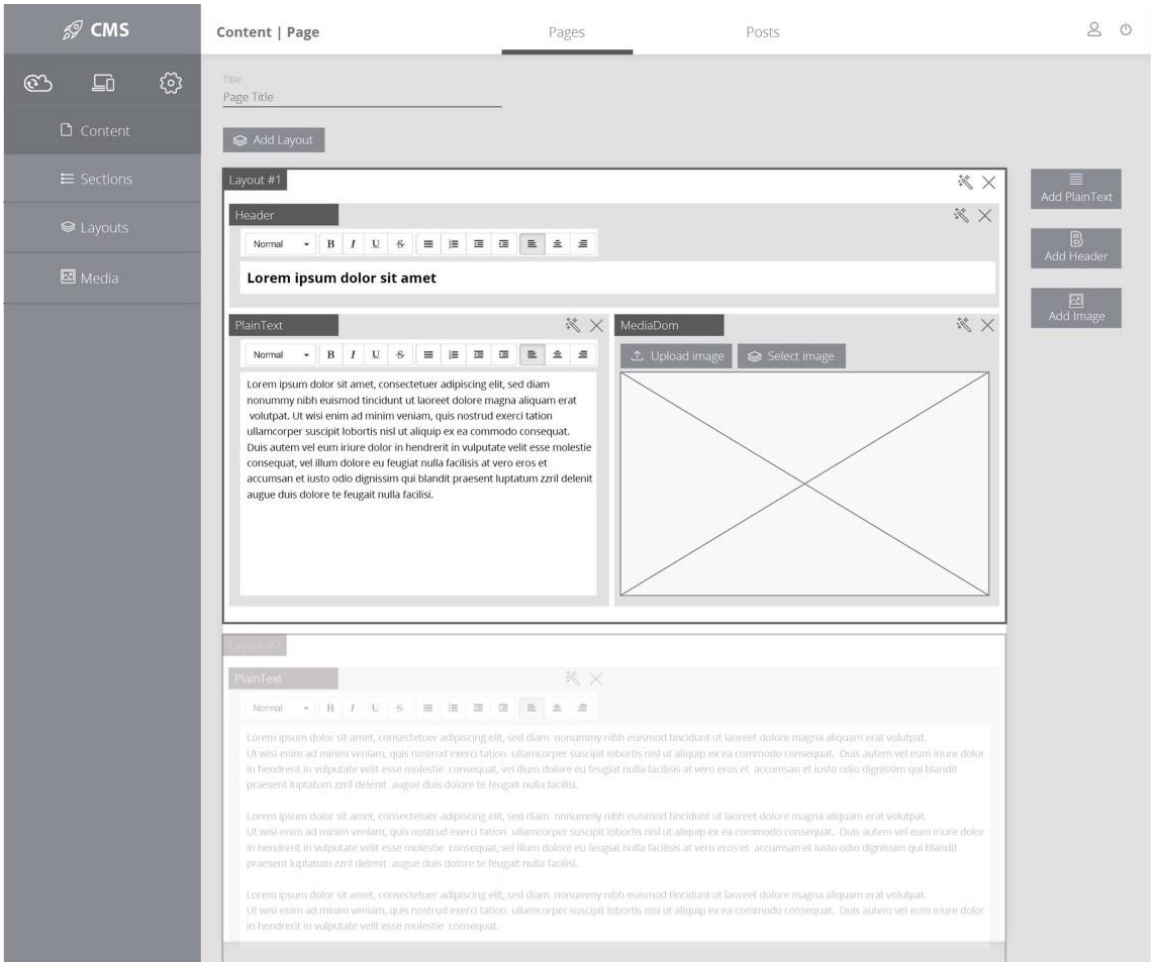


Figura 10: wireframes secció d'edició de continguts

Font: elaboració pròpia

2.7 DISSENYS D'ALTA FIDELITAT

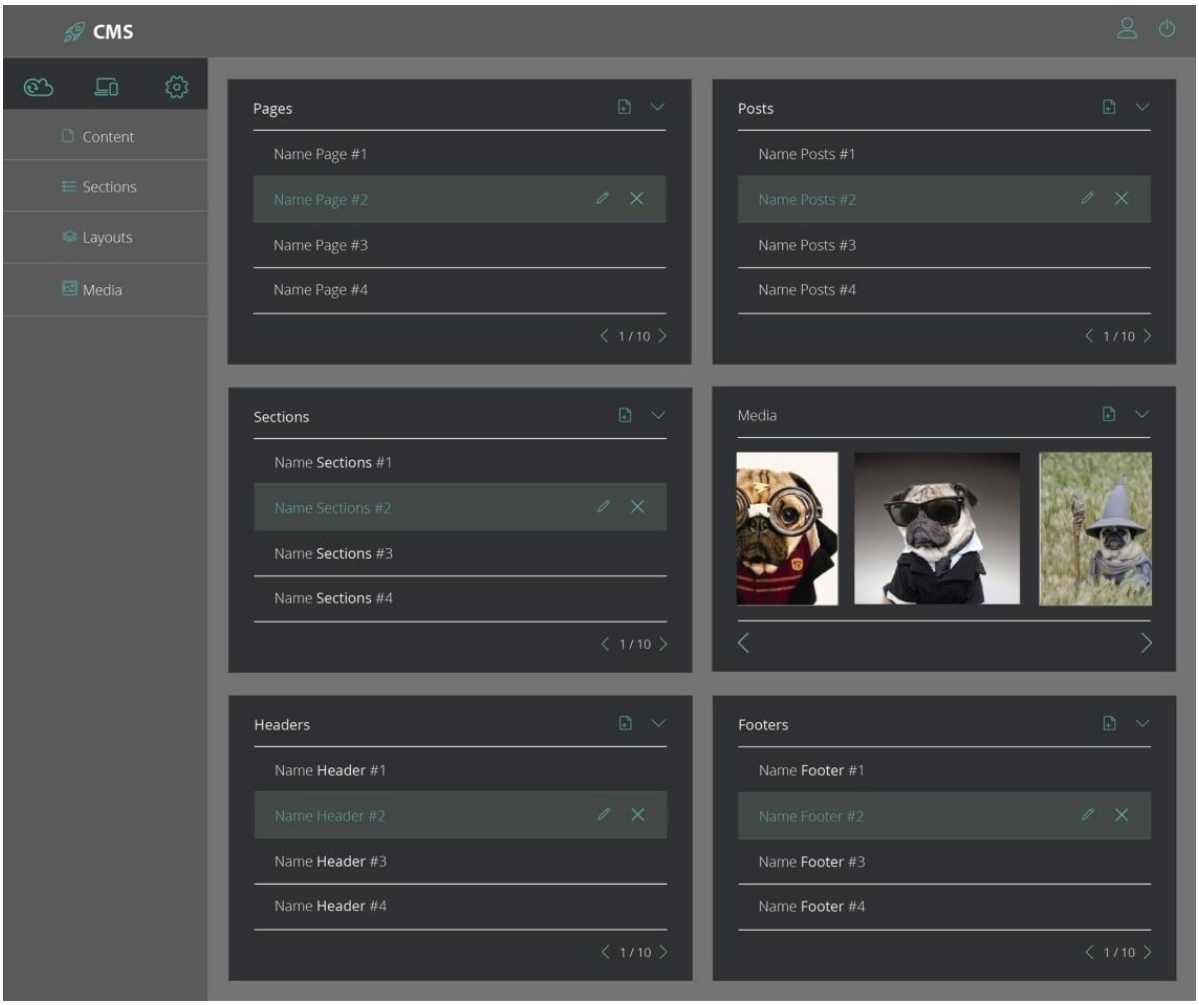


Figura 11: disseny d'alta fidelitat secció principal

Font: elaboració pròpia

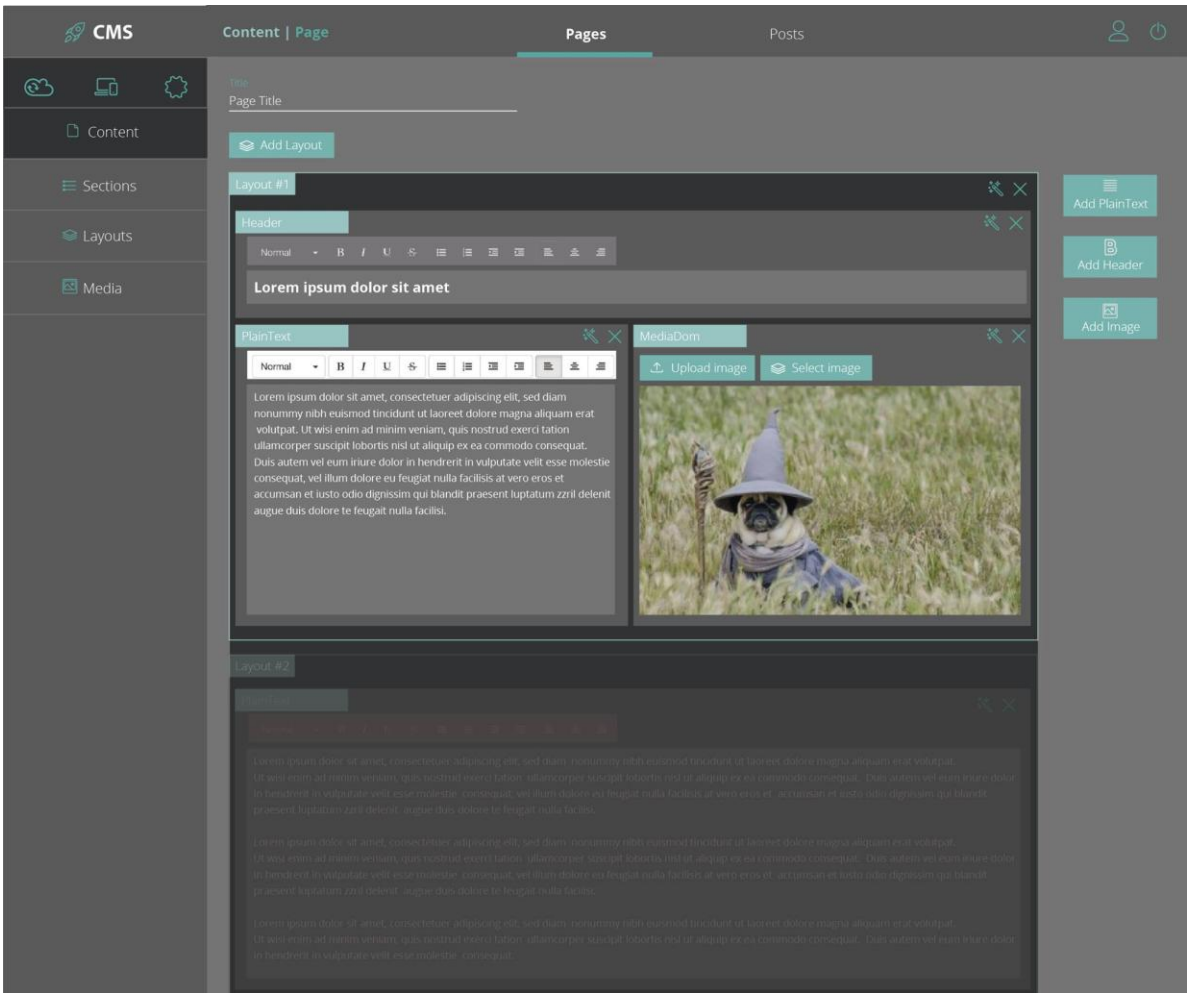


Figura 12:disseny d'alta fidelitat secció d'edició de continguts

Font: elaboració pròpia

3. ARQUITECTURA

3.1 MÒDULS

Es podria definir l'arquitectura modular com aquell procés d'organització de funcionalitats que es centra principalment en desenvolupar una aplicació o servei de software a partir d'una sèrie de petits components, cada un d'ells ejectant-se de forma autònoma i comunicant-se entre ells, per exemple a través de serveis o d'esdeveniments, en el nostre els serveis són anomenats façanes i els esdeveniments estan gestionats pels frameworks.

Cada un d'aquests components té la responsabilitat de gestionar una part del model i la lògica del sistema total, com es pot observar en el diagrama, cada una de les funcions principals afegides a partir de l'anàlisi de requeriments i casos d'ús, és desenvolupada com un mòdul aïllat de la resta, el qual es comunica amb altres mòduls a partir del seu servei façana, que té la funció de rebre peticions de l'exterior.

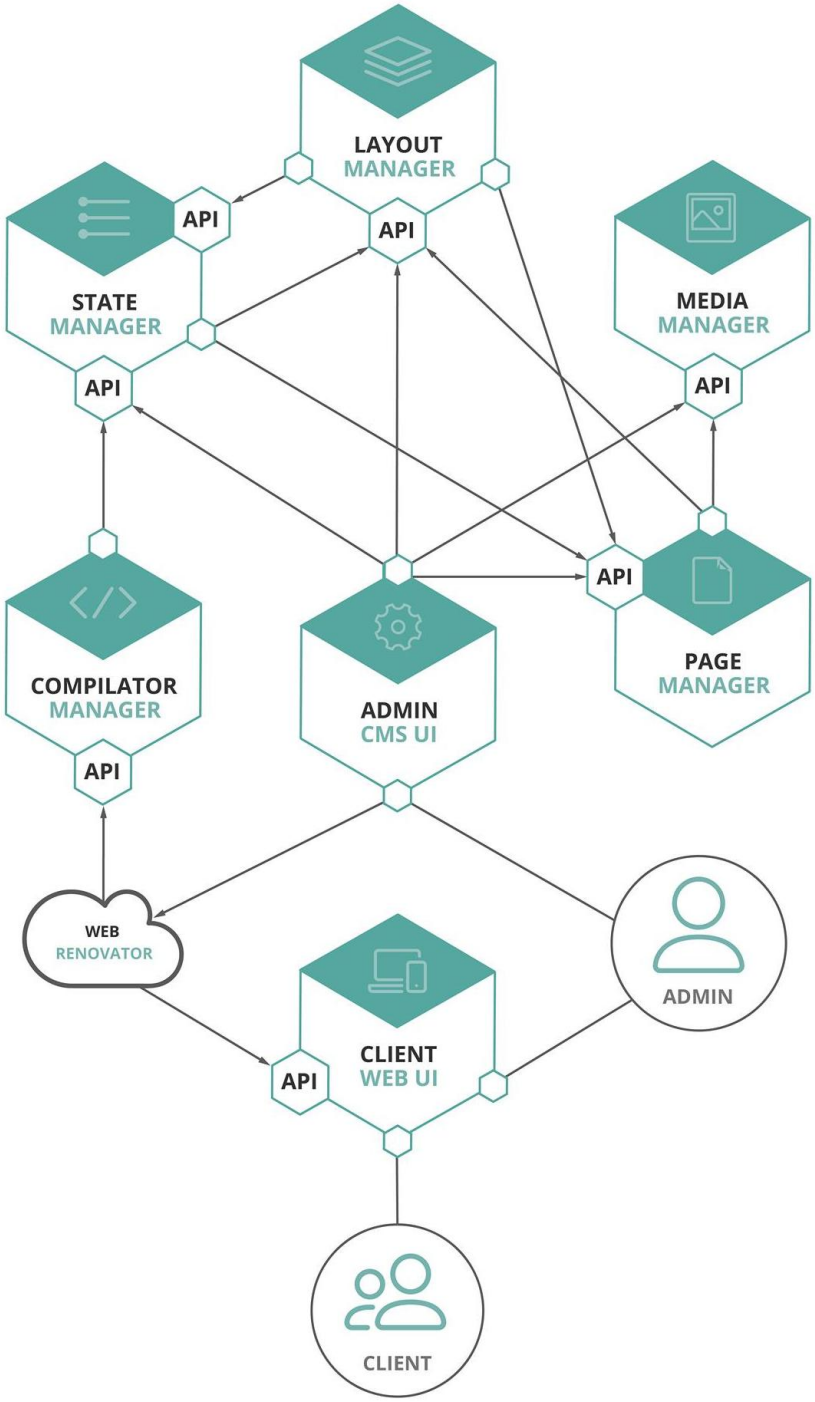
Així doncs, cada un dels mòduls té un codi totalment independent de la resta, promovent la reutilització de codi i el desacoblament entre funcionalitats, ja que podem modificar un mòdul o servei sense afectar els altres, només tenint en compte que mantinguí tota aquella informació que exposa a l'exterior. Aquesta divisió del sistema ens permet que cada un dels mòduls pugui estar escrit amb diferent codi, d'aquesta manera, no és necessari replicar l'entorn del servidor central que gestiona els continguts a cada un dels servidors externs.

En el diagrama podem veure que cada un dels blocs representa un mòdul el qual exposa exclusivament la seva API, la qual, com s'ha esmentat anteriorment, serà utilitzada per altres mòduls. Les connexions entre mòduls no sempre resultaran bidireccionals, ni tampoc es duran a terme enviant informació a través de les serveis, ja que pot existir la possibilitat que un mòdul o component utilitzi directament un servei extern, és per això que es representen els components o mòduls interns com a petits blocs dels quals surten unes fletxes, les quals descriuen la direcció de la petició.

Els usuaris que s'hi representen, tindran limitacions de control i accés sobre aquests mòduls en funció de si tenen els permisos per entrar i gestionar l'interfície de gestió de continguts. Cal dir

que, quan l'usuari administrador vulgui realitzar qualsevol modificació o actualització a la seva pàgina web des del servidor central, només tindrà contacte directe amb el servei Admin CMS UI, el qual representa el mòdul controlador de les vistes de l'interfície i és l'encarregat de gestionar tots aquells estímuls interns que realitzi l'usuari.

Finalment, el web renovator és un servei encarregat de gestionar totes aquelles comunicacions entre el servidor central i la pàgina web externa, de tal manera, que tot el seu contingut físic es troba repartir entre ambdós llocs, ja que ell es l'encarregat d'enviar-se i rebre les peticions d'un servidor a l'altre, fent que cap altre mòduls depengui de l'exterior.



Font: elaboració pròpia

Figura 13: disseny de l'arquitectura modular

Per exemplificar de forma pràctica l'aplicació d'aquesta arquitectura modular a continuació mostro una de les aplicacions i utilitzacions d'un mòdul.

```
1  class PageFacade {
2      constructor($reactive){
3          // ... skipped lines ... //
4      }
5
6      // ... skipped methods ... //
7
8      save(page, callback){
9          this.call('insertPage', page,
10             (error, response) => {
11                 if(typeof callback == 'function')
12                     callback(error, response);
13             }
14         );
15     }
16 };
```

```
1  class FooModule {
2      constructor($reactive, pageFacade, $rootScope){
3          // ... skipped lines ... //
4          this.pageFacade = pageFacade;
5          this.root = $rootScope;
6      }
7
8      // ... skipped methods ... //
9
10     savePage(page){
11         this.pageFacade.save(page,
12             (error, response) => {
13                 if(!error){
14                     this.root.throwMessage('Page saved successfully');
15                 } else {
16                     this.root.throwMessage(error.reason);
17                 }
18             }
19         );
20     }
21 };
```

En aquest cas podem veure com una classe externa (FooModule) injecta com a dependència el component pageFacade, aquest component no és més que el servei del mòdul que gestiona les pàgines. Així doncs, la classe externa pot utilitzar qualsevol de les funcions que exposa el mòdul que gestiona les pàgines (pageManager), sense la necessitat de conèixer com aquesta guardarà la pàgina. A més a més, podem veure a les línies 14 i 16 de la classe FooModule, com es llença un esdeveniment a través del framework, el qual serà recollit pel mòdul que estigui escoltant aquest bus d'esdeveniments.

Així doncs, cada un dels mòduls representats en el diagrama anterior es representa com una o diverses classes les quals centralitzen les seves funcions que poden exposar a l'exterior a

través d'una classe servei, anomenada en aquesta projecte, façana. D'aquesta manera, per utilitzar un servei façana de qualsevol altre mòdul, només s'haurà de d'injectar com a dependència en el seu constructor (línia 2 de FooModule), amb la finalitat de poder instanciar-la en qualsevol moment (línies 11, 14, 16 de FooModule).

3.2 BASE DE DADES

S'ha representat el disseny de la base de dades mitjançant el paradigma de l'orientació a objectes, degut a que s'ajusta millor al concepte de base de dades no relacional, a diferencia dels diagrames ER.

Cada una de les classes o entitats, representa una col·lecció de la base de dades, amb els seus atributs, els quals poden ser totalment dinàmics, sense necessitat d'un tipus ni necessitat de que existeixin tots.

Els tipus de dades són els formats que prendran els objectes que s'emmagatzemin dins de l'array de l'atribut.

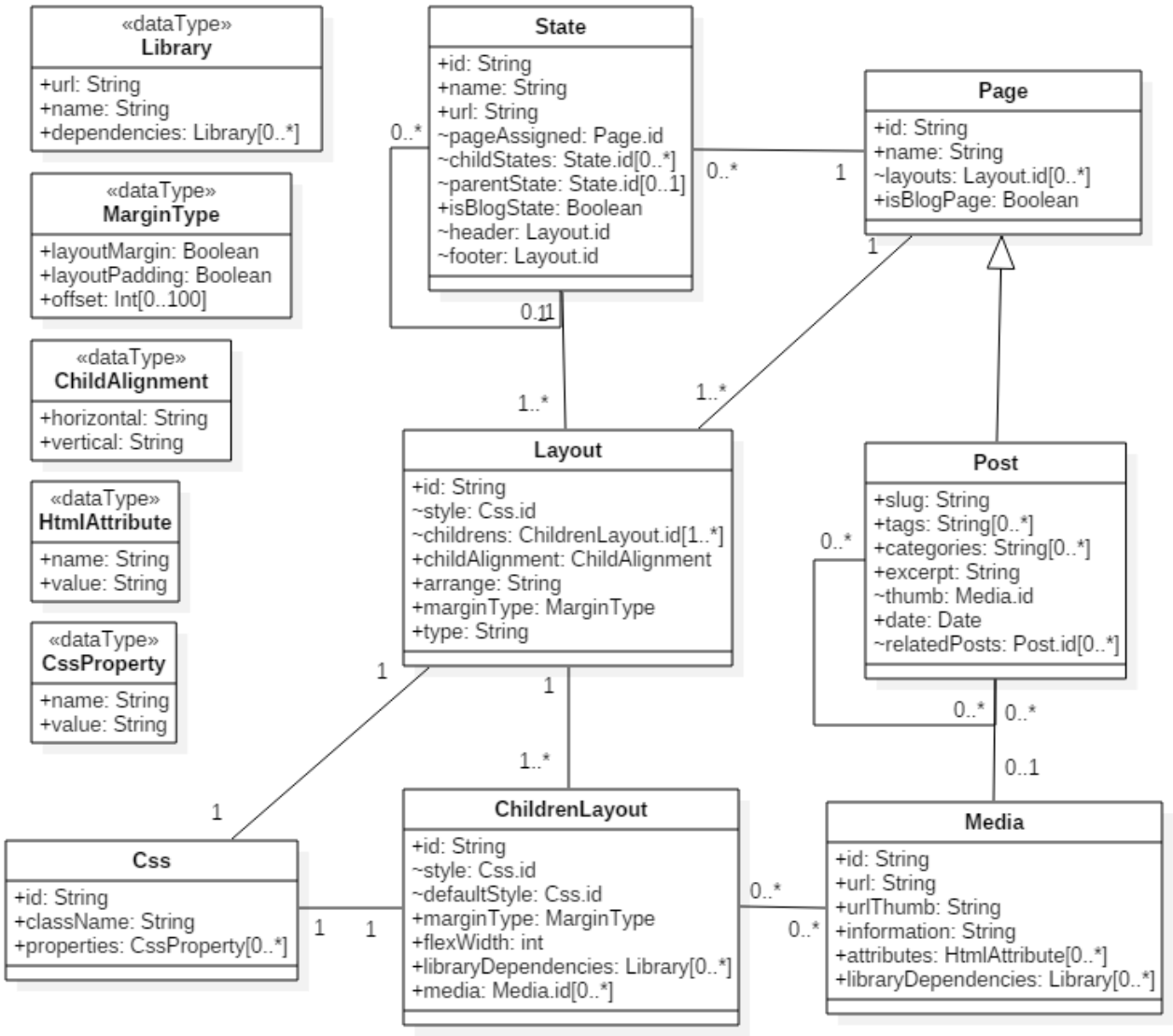


Figura 14: diagrama base de dades no relacional

Font: elaboració pròpia

4. DESENVOLUPAMENT

4.1. APIs / LLIBRERIES DE TERCERS.

4.1.1 Meteor (https://www.meteor.com/)

Meteor.js es defineix com una plataforma de desenvolupament , construïda a partir de diferents col·leccions de biblioteques i paquets que s'uneixen de forma ordenada per facilita el desenvolupament. Està basat en idees de frameworks anterior i biblioteques, amb l'objectiu d'oferir d'una des de la iniciació d'un prototip d'aplicació fins a aplicacions complexes en entorns de producció exigents. A continuació podem veure la seva aplicació pràctica juntament amb AngularJS:

```
import angular from 'angular';
import angularMeteor from 'angular-meteor';
import { Meteor } from 'meteor/meteor';

class PageFacade {
    constructor($scope, $reactive){
        'ngInject';

        //attach to $scope to digest after Meteor call
        $reactive(this).attach($scope);

        // ... skipped lines ... //

        this.subscribe('pages');

        this.helpers({
            pages(){
                return Pages.find({});
            }
        })
    }

    // ... skipped methods ... //

    save(page, callback){
        this.call('insertPage', page,
            (error, response) => {
                // ... skipped lines ... //
            }
        );

    }
};

const name = 'pageFacade';

export default angular.module(name, [
    angularMeteor,
]).component(name, {
    controllerAs : name,
    controller : PageFacade
});
```

A les primeres tres línies podem observar com importem diferents llibreries de tercers, en aquest cas cal destacar el mòdul angularMeteor (https://github.com/Urigo/angular-meteor) importat a la segona línia, el qual permet integrar meteor amb angular.

Seguidament podem observar la creació de la classe, on en el constructor li introduïm les dependències que utilitzarà, ja siguin mòduls, serveis, APIs de tercers o components propis del framework. En aquest projecte podríem destacar dues dependències pràcticament comunes entre tots els mòduls, una és la de $scope[3], la qual permet vincular l'informació dinàmica del nostre model amb la vista. Per altra banda tenim la dependència $reactive[4], la qual és un servei importat a través del mòdul comentat anteriorment, angularMeteor, el qual permet crear un vincle reactiu entre la $scope i l'informació provinent del backend, fent que qualsevol operació a la base de dades o al servidor, refresqui automàticament la vista del client (ja que prèviament estaria modificant la $scope i seria aquesta la que modificaria la vista).

En relació amb l'exposat fins ara, les funcions subscribe[5] i helpers[6], permeten vincular la $scope amb una col·lecció a la base de dades, a l'exemple, estaríem obtenint tots els elements pàgina de la col·lecció Pages, fent que si es produís un canvi a la base de dades, el servidor actualitzaria el miniMongo[7] (una copia de la base de dades que permet obtenir l'informació directament des del client de forma més òptima), al actualitzar-lo, la vista és refrescada amb els nous elements ja que està observant aquesta simulació de base de dades.

Finalment, podem executar operacions al servidor a través de mètodes de Meteor. Aquests mètodes és poden cridar mitjançant l'operació call, a la qual li passem el nom del mètode que volem executar, els arguments i un callback per gestionar la resposta. Per crear un mètode de Meteor ho haurem de realitzar dins dels arxius que s'executen al costat del servidor i tindran una forma semblant a:

[3] Per a més informació consulti: https://docs.angularjs.org/guide/scope Data d'última consulta: 01.01.2017
[4] Per a més informació consulti: https://angular-meteor.com/api/angular-meteor/1.3.11/reactive Data d'última consulta: 01.01.2017
[5] Per a més informació consulti: https://angular-meteor.com/api/angular-meteor/1.3.11/subscribe Data d'última consulta: 01.01.2017
[6] Per a més informació consulti: https://angular-meteor.com/api/angular-meteor/1.3.11/helpers Data d'última consulta: 01.01.2017
[7] Per a més informació consulti: https://guide.meteor.com/collections.html#client-collections Data d'última consulta: 01.01.2017

```
import { Meteor } from 'meteor/meteor';
import { Pages } from './collection';

export function insertPage(newPage){
    if (!this.userId)
        throw new Meteor.Error(400, 'You must be logged in');

    // ... skipped check lines ... //

    //insert into DB and get _id
    newPage._id = Pages.insert(newPage);

    return { page: newPage };
}

Meteor.methods({
    insertPage
});
```

En aquest exemple, el mètode tindria la finalitat d'inserir una nova pàgina a la base de dades, per fer-ho és necessari importar la col·lecció Pages, la qual s'explica el seu funcionament al següent apartat, i agregar la funció que desitgem executar a la operació methods[8] de l'API de Meteor. Cal destecar que la gestió d'errors es realitza a través de Meteor, el qual ens permet llençar esdeveniments d'error, els quals seran retornats a través del callback a la funció call del client.

4.1.2 MongoDB (https://www.mongodb.com/es)

MongoDB és un gestor de base de dades de codi obert que utilitza models de dades orientades a objectes.

MongoDB és una de les moltes base de dades que han anat apareixent als últims anys sota el concepte *NoSQL*. Aquest nou paradigma d'emmagatzament de dades es diferencia de les antigues bases de dades, les quals utilitzaven taules i files per emmagatzemar informació, a diferencia que MongoDB ho fa a través de col·leccions i documents, aquests documents estan compostos per parells d'atributs clau-valor.

En aquest projecte la seva implementació i utilització es realitza integrament a través de Meteor, i en alguns casos a través de AngularJS, el qual instancia un servei extern per fer-ho. Podem veure la seva principal utilització en els exemples següents:

[8] Per a més informació consulti: https://guide.meteor.com/methods.html Data d'última consulta: 01.01.2017

```
import { Mongo } from 'meteor/mongo';

export const Pages = new Mongo.Collection('pages');

Pages.allow({
    insert (userId, page) {
        return userId && page.owner === userId;
    },
    update (userId, page, fields, modifier) {
        return userId && page.owner === userId;
    },
    remove (userId, page) {
        return userId && page.owner === userId;
    }
});
```

En aquest exemple podem veure com primerament importem el mòdul de Mongo proporcionat per Meteor amb la finalitat de crear una nova col·lecció (podríem dir que si estiguéssim treballant sobre un paradigma relacional esdevindria una taula). Per fer-ho instanciem la classe Collection[9] de l'API de Mongo, passant-li exclusivament el nom de la col·lecció. En aquest punt ja seriem capaços de crear nous documents (files en el paradigma relacional) a aquesta nova col·lecció. Malgrat això, per tal de limitar l'accés i la gestió d'aquestes col·leccions, en definitiva, augmentar la seva seguretat, podem utilitzar el mètode allow[10], que ens permet restringir i limitar la gestió de la col·lecció. En l'exemple anterior estaríem l'imitant les funcions d'inserció, modificació i eliminació a usuaris registrats i propietaris del document.

Ara seriem capaços de mostrar i crear nous documents a la base de dades des del nostre servidor, però per poder fer-ho des del client es necessari un altre pas:

```
import { Meteor } from 'meteor/meteor';
import { Pages } from './collection';

Meteor.publish('pages', function(){
    return Pages.find({'metaData.owner' : this.userId});
});
```

Aquest pas, com bé s'anomena el mètode de l'API de Meteor, té la funció de publicar el que nosaltres desitgem, en aquest cas estaríem publicant totes les pàgines que el seu propietari coincidís amb l'usuari que estigués identificat, en altres paraules, només mostrarem als usuaris

[9] Per a més informació consulti: https://docs.meteor.com/api/collections.html Data d'última consulta: 01.01.2017

[10] Per a més informació consulti: https://docs.meteor.com/api/collections.html#Mongo-Collection-allow Data d'última consulta: 01.01.2017

les seves pàgines. Per poder accedir a aquesta publicació és necessari subscriure's des del client, tal i com hem vist a l'apartat anterior.

Com es pot observar, el que estem retornant en el cas anterior és el resultat d'una consulta a la base de dades, aquestes consultes es poden realitzar tant des del client (a través dels helpers anomenats a l'apartat anterior) com es del servidor. A continuació es mostra un exemple de consulta:

```
Pages.find({
    $or : [{
        'metaData.owner' : userId
    },{
        $and :[{
            'metaData.public' : {
                $exists : true
            }
        },{
            'metaData.public' : true
        }]
    }]
});
```

En aquesta consulta estem demanant a la base de dades que ens retorni totes les pàgines que hagi creat l'usuari, o bé aquelles que siguin públiques.

Finalment, cal tenir en compte, que si des del client realitzem aquesta darrera consulta, i des del servidor estem publicant la col·lecció amb les restriccions anteriors, no obtindríem les pàgines públiques, ja que el servidor no les dipositaria al miniMongo (simulació de la base de dades, que té com a finalitat optimitzar les consultes provinents del client).

4.1.3 AngularJS (https://angularjs.org/)

AngularJS és un *framework MVC* de codi obert escrit amb Javascript, que té com a principal funció la de simplificar el desenvolupament web a través de l'automatització i sincronització de les vistes amb els models. A més a més, té la capacitat de gestionar el two-way binding[11] de forma molt lleugera, alhora que és suportat a la majoria dels navegador i construït per poder crear codi *testable*.

A més a més, AngularJS ens permet desenvolupar el nostre codi de forma modular, ja que està pensat per poder crear una aplicació a través de diferents mòduls[12], ja siguin de tercers o propis. Aquests mòduls, estan composts bàsicament per components (en el paradigma MVC

[11] Per a més informació consulti: https://docs.angularjs.org/guide/databinding
[12] Per a més informació consulti: https://docs.angularjs.org/guide/module

seria el model), serveis i vistes. Cal destecar, que el vincle i el dinamisme entre la capa de la vista i del model és tan alta, que el controlador està íntegrament gestionat pel propi framework. A continuació es mostra un exemple d'un mòdul:

```
1  import angular from 'angular';
2  import angularMeteor from 'angular-meteor';
3
4  import { Meteor } from 'meteor/meteor';
5
6  import { name as PageResume } from './pageResume/pageResume';
7  import { name as PageCreation } from './pageCreation/pageCreation';
8  import { name as PageEditor } from './pageEditor/pageEditor';
9
10 class PageFacade {
11     constructor($scope, $reactive, $rootScope, pageEditor){
12         'ngInject';
13
14         // ... skipped lines ... //
15     }
16
17     // ... skipped methods ... //
18 };
19
20 class PageFacadeService {
21     // ... skipped lines ... //
22 }
23
24 const name = 'pageFacade';
25
26 export default angular.module(name, [
27     angularMeteor,
28     PageResume,
29     PageCreation,
30     PageEditor
31 ]).component(name, {
32     controllerAs : name,
33     controller : PageFacade
34 }).service(name, PageFacadeService);
```

Pel que fa a les importacions de l'inici del document, primerament necessitem importar el mòdul angular, bàsicament és el propi framework. Seguidament tenim angularMeteor, el qual és un mòdul extern d'AngularJS que ens permet connectar el client amb el servidor, concretament amb Meteor.

Importem el Meteor en el cas que necessitem cridar o executar algun dels mètodes creats al costat del servidor, en cas contrari no seria necessària la seva implementació. També podria ser necessari importar la col·lecció en el cas de utilitzar i realitzar una consulta a través de un helper.

Per finalitzar les importacions, hi ha dos elements més vinculats al propi framework i al seu funcionament, un és la d'importar els mòduls propis per tal d'injectar-los com a dependència

al mòdul que estem creant, i per altra banda tindríem el template, que hauríem d'importar l'arxiu html i agregar-lo al component amb l'atribut template.

Els mòduls que es vulguin injectar com a dependències del mòdul que estiguem creant s'hauran d'afegir a través d'un Array, just després del nom del mòdul, tal i com es mostra a l'exemple anterior a la línia 26.

Després de la creació del mòdul, haurem d'afegir tots els elements dels quals volem que estigui compost aquest mòdul, com els components[13] i els serveis[14] (tot i que pot tenir molts altres elements com filtres, directives, controladors, etc.).

Aquest servei estarà disponible posteriorment per tots els components del mòdul que injecti com a dependència aquest mòdul. Per tenir-hi accés s'han d'afegir en el constructor el nom del servei, tal i com es mostra a la línia 11 amb el servei PageEditor, tot i que podria utilitzar el seu propi servei, en aquest cas el PageFacade, present a la línia 20 i afegit al mòdul a la línia 34.

4.1.4 Node-json2html (https://github.com/moappi/node-json2html)

Json2html és sistema generador de contingut HTML a través de Javascript, el qual permet convertir objectes JSON a HTML a través de la seva API. Té la possibilitat de ser implementat com un mòdul al costat del servidor o com una llibreria al costat del client.

La seva utilització és molt senzilla, malgrat això s'ha hagut d'adaptar l'estructura de cada un dels objectes que s'emmagatzemaven a la base de dades, ja que per poder crear el contingut html era necessari una determinada composició i organització de l'informació en un objecte de tipus JSON.

[13] Per a més informació consulti: https://docs.angularjs.org/guide/component Data d'última consulta: 01.01.2017

[14] Per a més informació consulti: https://docs.angularjs.org/guide/services Data d'última consulta: 01.01.2017

```
1  import { Meteor } from 'meteor/meteor';
2  import { Pages } from './collection';
3  import json2html from 'node-json2html';
4
5  export function pageParser(id){
6      const pageObject = Pages.findOne({ _id : id });
7
8      var html = json2html.transform([{}], pageObject);
9
10     return html;
11 }
12
13 /* Example page Object
14  *  pageObject = {
15  *      '<>'          : 'main',
16  *      class         : 'qwertyu',
17  *      layout        : 'row',
18  *      'layout-wrap' : '',
19  *      html          : [] || 'Lorem Ipsum'
20  *  };
21  */
```

Per transformar l'objecte JSON a un String amb el contingut html, només hem d'utilitzar l'operació transform que ens ofereix la llibreria json2html, on el primer argument és un objecte amb diferents configuracions, i el segon l'objecte JSON a ser transformat.

Aquests objectes JSON han de tenir l'estructura que es mostra a la línia 13 de l'exemple, on l'atribut '<>' determina el tipus d'etiqueta, i la resta d'atributs correspondran a atributs del propi element, excepte l'atribut html, el qual contindrà l'element o elements fills de l'objecte, d'aquesta manera, un element pot contenir tants fills com sigui necessari a través de l'array de l'atribut html, i a l'hora els fills poden tenir de forma recursiva més fills.

4.1.5 Angular Material (https://material.angularjs.org/latest/)

Angular Material es un component que s'utilitza juntament amb AngularJS per atorgar estil a diferents elements, a més de diverses funcionalitats a partir de directives. En aquest projecte s'utilitza principalment el sistema de graelles que ofereix, per tal de generar el contingut de la pàgina web externa.

Aquesta llibreria s'ha utilitzat tant per la realització i maquetació de panell d'administració del gestor de continguts com a les webs externes, tal i com s'ha comentat anteriorment. La principal funció a les webs externes a través de les directives (atributs dels elements html) anomenats layout i flex, els quals ens permeten editar l'amplada i la distribució de tots els fills d'un element.

```
<div layout="row" layout-wrap>
  <div flex="50"> 50% Width</div>
  <div flex="25"> 25% Width</div>
  <div flex="25"> 25% Width</div>
</div>
```

Com es pot observar a l'exemple anterior, l'atribut layout, determina la posició en la qual s'aniran col·locant els elements fills, i l'atribut layout-wrap té la funció de que no es generi un salt de línia després de cada element, fent que si la suma de diferents elements és inferior al 100%, aquests es mostraran en una mateixa fila. Finalment, l'atribut flex present a cada un dels fills determina en percentatge del total de l'amplada de l'element pare, en cas de no inserir cap valor, s'adaptarà en funció de l'últim element.

4.1.6 Ui.Sortable (http://angular-ui.github.io/ui-sortable/)

Aquest component ofereix la possibilitat d'obtenir les funcionalitats de *drag&drop* de característiques de jQueryUI[15] a través d'una directiva d'AngularJS, fent possible la redistribució i reordenament dels elements dinàmics generats al navegador a través d'una cadena d'objectes.

```
<section ui-sortable ng-model="pageEditor.pageContainer.html">
  <div ng-repeat="layout in pageEditor.pageContainer.html">
    <layout-editor layout-id="layout._id"></layout-editor>
  </div>
</section>
```

Per implementar aquesta llibreria s'ha de primer importar al mòdul d'angular que la vulguem utilitzar. Per poder aprofitar les seves caracteristiques hem d'incloure l'atribut ui-sortable a l'element pare que contingui l'iteració d'elements fills, aquets atribut utilitza una directiva a la qual li passem l'array d'elements a través de l'atribut ng-model[16] (atribut del propi AngularJS, que permet vincular el contingut de l'element amb un objecte o variable del component).

Quan l'usuari modifiqui a través del drag&drop la posició i ordre dels layouts, aquests es reordenaran dins l'objecte que li passem per ng-model, fent que quan es vulgui guardar a la base de dades no faci falta consultar i modificar les noves posicions.

[15] Per a més informació consulti: http://api.jqueryui.com/ Data d'última consulta: 01.01.2017
[16] Per a més informació consulti: https://docs.angularjs.org/api/ng/directive/ngModel Data d'última consulta: 01.01.2017

4.1.7 Angular UI Tree (https://github.com/angular-ui-tree/angular-ui-tree)

Angular Tree és un component de AngularJS UI que permet ordenar i niar diferents elements d'una llista mitjançant la funció de *drag&drop*, sense dependre de jQuery[17]. Aquesta llibreria s'ha utilitzat per resoldre el problema a l'hora d'ordenar i niar els diferents estats uns dins dels altres, a continuació es mostra un exemple:

```
1  <!-- Nested node template -->
2  <script type="text/ng-template" id="state-template.html">
3    <div ui-tree-handle>
4      {{state.title}}
5    </div>
6    <ol ui-tree-nodes="" ng-model="state.states">
7      <li ng-repeat="state in state.states" ui-tree-node ng-include="'state-template.html'">
8      </li>
9    </ol>
10 </script>
11
12 <!-- Template -->
13 <div ui-tree>
14   <ol ui-tree-nodes="" ng-model="states" id="tree-root">
15     <li ng-repeat="state in states" ui-tree-node ng-include="'state-template.html'"></li>
16   </ol>
17 </div>
```

Per implementar aquesta llibreria ha sigut necessari crear un template pels estats, aquests template s'inclou a través de ng-include[18] de forma recursiva a cada un dels elements iteradors, d'aquesta manera es crea un element nou per cada un dels fills dels que disposi.

Per poder crear aquest template dinàmic i recursiu és necessari que el template importat s'importi a ell mateix en cas de tenir més estats, tal i com es mostra a la línia 7 de l'anterior exemple.

Finalment, a l'igual que passava amb Ui.sortable, els elements que són redistribuïts també són modificats en el component respectiu, fent que quan l'usuari guardi la nova distribució, aquesta s'emmagatzemi a la base de dades sense la necessitat de comprovar i modificar el nou ordre.

4.1.8 Text Angular (https://github.com/fraywing/textAngular)

Aquesta llibreria ens permet crear i personalitzar un gestor de text a través d'un component d'AngularJS, captant fàcilment el contingut editat i introduït per l'usuari al model de la nostre aplicació. Per poder ser utilitzat, només hem d'injectar-lo com a dependència a l'hora de crear

[17] Per a més informació consulti: http://api.jquery.com/ Data d'última consulta: 01.01.2017
[18] Per a més informació consulti: https://docs.angularjs.org/api/ng/directive/ngInclude Data d'última consulta: 01.01.2017

el nostre mòdul i seguidament afegir l'atribut text-angular a l'element on volem que es mostri l'editor de text, juntament amb l'objecte del model que volem que es vinculi.

```
<div text-angular ng-model="plainText.metaData.html"></div>
```

Això permetrà a l'usuari modificar el contingut a través de l'editor de text, a l'hora que es manté vinculat amb el model, fent que quan l'usuari guardi, el seu contingut s'emmagatzemi automàticament a la base de dades sense la necessitat de modificar res.

A més a més ens permet configurar com es mostrarà i les funcions que tindrà aquest editor de text. En el nostre cas la configuració ha estat la següent:

```
function config($provide){//textAngular configuration
  'ngInject';
  $provide.decorator('taOptions', ['taRegisterTool', '$delegate',
    (taRegisterTool, taOptions) => { // $delegate is the taOptions we are decorating
          taOptions.toolbar = [
              ['h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'p', 'pre', 'quote'],
              ['bold', 'italics', 'underline', 'strikeThrough', 'ul', 'ol', 'redo', 'undo'],
              ['html', 'insertLink', 'insertVideo', 'wordcount', 'charcount']
          ];
          return taOptions;
      }
    ]);
}
```

En aquesta funció de configuració s'ha modificat bàsicament els elements d'edició que s'han de mostrar, a l'hora que el seu ordre i la seva distribució, ja que cada un dels arrays representa una fila.

4.2. DOCUMENTACIÓ DEL CODI

A continuació es documenten tots aquells elements del sistema, amb la finalitat de ajudar a comprendre'l, simplificar la seva extensió i facilitar la incorporació de tercers en el projecte.

Aquesta documentació es troba organitzada en un primer nivell jeràrquic per tots aquells **mòduls** creats i dissenyats anteriorment, els quals tenen un conjunt de funcions i elements que poden ser utilitzats per altres mòduls.

D'aquesta manera, totes les funcions del mòdul que es vulguin exposar a altres mòduls s'hauran d'incoporar a la seva façana, també anomenada en aquest cas API. Cada un dels elements de l'**API** que es mostren a continuació són mètodes de la classe façana que actuen

com a endpoint. Aquestes classes façanes tenen el rol de **serveis**, ja que els seus mètodes han de ser utilitzats per diferents classes, mòduls i components.

Cada un d'aquesta mòduls tenen funcions que requereixen d'un model i d'una vista per poder ser utilitzats, en altres paraules, totes aquelles operacions, requisits i casos d'ús que requereixin d'una interfície gràfica que interactuí amb l'usuari, s'han encapsulat en **components**, els quals tenen una vista i un model, controlats per un controlador integrat en el propi framework.

Així doncs, la documentació present en aquest apartat fa referencia als endpoints (API) dels serveis i els components de cada un dels mòduls principals dissenyats a les primeres fases del projecte.

4.2.1 LayoutManager (CMS Angular Meteor/imports/ui/components/layoutManager/)

Aquest mòdul té com a principal funció la d'editar, crear i eliminar qualsevol secció d'una pàgina a través de la seva API (LayoutFacade), a més d'oferir la possibilitat d'instanciar components per la edició dels continguts.

LayoutFacade.deleteLayout (layout, [callback]) **API Endpoint** (CMS Angular Meteor/imports/ui/components/layoutManager/layoutFacade.js, line 38)	
Descripció	Elimina de la base de dades el layout. Envia un missatge de confirmació.
Arguments	**layout** *Layout* Objecte layout que s'ha d'eliminar, és necessari que contingui la id de l'element. **callback** *Function* Funció que retorna l'error i la resposta del procediment. En cas d'èxit l'error equivaldrà a null.

LayoutFacade.save (layout, [callback]) **API Endpoint** (CMS Angular Meteor/imports/ui/components/layoutManager/layoutFacade.js, line 59)	
Descripció	Guarda un nou layout a la base de dades, en el cas d'existir, serà sobreescrit.
Arguments	**layout** *Layout* Objecte layout que s'ha de guardar. En cas de provenir de la base de dades ha de contenir l'atribut _id.

	callback *Function* Funció que retorna l'error i el layout guardat. En cas d'èxit l'error equivaldrà a null.

LayoutFacade.update(layout, [callback]) **API Endpoint** (CMS_Angular_Meteor/imports/ui/components/layoutManager/layoutFacade.js, line 81)	
Descripció	Sobreescriu el layout a la base de dades, en cas de no existir retornarà un error.
Arguments	**layout** *Layout* Objecte layout que s'ha de modificar. És necessari que contingui l'atribut _id assignat per Mongo quan va ser creat. **callback** *Function* Funció que retorna l'error i el layout actualitzat. En cas d'èxit l'error equivaldrà a null.

LayoutFacade.getLayoutById (layoutId, callback) **API Endpoint** (CMS_Angular_Meteor/imports/ui/components/layoutManager/layoutFacade.js, line 92)	
Descripció	Retorna el layout corresponent, en funció del paràmetre id que se li passa al mètode, a través de la resposta del callback.
Arguments	**layoutId** *String* Cadena de text amb el valor de la id del layout que es vol cercar. **callback** *Function* Funció que retorna l'error i la resposta del procediment, en aquest cas el layout. En cas d'èxit l'error equivaldrà a null.

LayoutFacade.createEmptyLayout( ) **API Endpoint** (CMS_Angular_Meteor/imports/ui/components/layoutManager/layoutFacade.js, line 101)	
Descripció	Retorna un objecte Layout predeterminat, preparat per començar a ser editat a través de l'editor de Layouts.

LayoutFacade.displayLayouts(callback) **API Endpoint** (CMS_Angular_Meteor/imports/ui/components/layoutManager/layoutFacade.js, line 123)	
Descripció	Mostra una finestra flotant amb tots els layouts de l'usuari i públics, i retorna aquells que l'usuari hagi seleccionat un cop accepti.

Arguments	**callback** *Function* Funció que retorna un array amb tots els layouts seleccionats per l'usuari. El segon paràmetre és un booleà que determina si s'ha cancel·lat la selecció.

LayoutFacade.throwMessage(message, [options], [callback]) **API Endpoint** (CMS_Angular_Meteor/imports/ui/components/layoutManager/layoutFacade.js, line134)	
Descripció	Mostra una finestra flotant amb el missatge i les opcions que se li passin en els paràmetres.
Arguments	**message** *String* Cadena de text amb el valor que es vol mostrar en el missatge. **options** *Object* Objecte amb les possibles respostes al missatge de la finestra flotant. L'opció predetermina és "Okay" **callback** *Function* Funció que retorna l'opció seleccionada per l'usuari, en cas de cancel·lació retornarà null.

LayoutResume **Component** (CMS_Angular_Meteor/imports/ui/components/layoutManager/layoutResume/layoutResume.js, line1)	
Descripció	Mostra el resum de tots els layouts de l'usuari i els públics.
Mètodes	**setFilter** ([*String* filter]) (CMS_Angular_Meteor/imports/ui/components/layoutManager/layoutResume/layoutResume.js, line44) Modifica el paràmetre utilitzat per filtrar els layouts de l'usuari **movePage** (*String* direction) (CMS_Angular_Meteor/imports/ui/components/layoutManager/layoutResume/layoutResume.js, line48) Determina cap a quina direcció es vol desplaçar l'usuari a través de la paginació. Les opcions de l'argument d'entrada són next o prev. **delete** (*Layout* layout) (CMS_Angular_Meteor/imports/ui/components/layoutManager/layoutResume/layoutResume.js, line56) Esborra un layout de la base de dades. Utilitza l'API del mòdul LayoutFacade.

LayoutEditor **Component** (CMS_Angular_Meteor/imports/ui/components/layoutManager/layoutEditor/layoutEditor.js, line1)	
Descripció	Mostra l'editor de layouts, el qual permet crear i/o modificar objectes de tipus layout.
Atributs	**name** String (Bidireccional) Cadena de text amb el valor del nom del layout. **layoutId** String (Unidireccional –Opcional) Cadena de text amb el valor de la id del layout que es vol editar. **options** Object (Unidireccional – Opcional) Delimita les característiques i els atributs que pot editar l'usuari. Els atributs de l'objecte tenen com a valor un booleà i poden ser: *save, delete* i *changeName*. **layoutToPrint** Layout (Bidireccional – Opcional) En cas d'haver una id a la url, aquest paràmetre sobreescriurà la id, i li donarà preferència a per ser editat a aquest layout.
Mètodes	**save** ([*Layout* layout]) (CMS_Angular_Meteor/imports/ui/components/layoutManager/layoutEditor/layoutEditor.js, line31) Guarda o actualitza el layout de la base de dades que està essent editat. Utilitza l'API del mòdul LayoutFacade. **delete** ([*Layout* layout]) (CMS_Angular_Meteor/imports/ui/components/layoutManager/layoutEditor/layoutEditor.js, line46) Elimina el layout de la base de dades que està essent editat. Utilitza l'API del mòdul LayoutFacade. Redirecciona l'usuari si s'ha finalitzat correctament. **createLayout** ( ) (CMS_Angular_Meteor/imports/ui/components/layoutManager/layoutEditor/layoutEditor.js, line63) Mètode que s'executa en el constructor, i te com a finalitat actualitzar tota la vista amb el layout que rep el component a través de la url, o bé de l'atribut layoutToPrint. **removeChildren** (*Integer* index) (CMS_Angular_Meteor/imports/ui/components/layoutManager/layoutEditor

	/layoutEditor.js, line88) Elimina un layout fill del layout que està essent modificat. El paràmetre d'entrada determina al posició a la que es trobava. **addChildren** (*String* type) (CMS_Angular_Meteor/imports/ui/components/layoutManager/layoutEditor/layoutEditor.js, line92) Crea i inserta un nou fill pel layout que esta essent editat a partir de l'API ChildrenLayout. El paràmetre d'entrada és una cadena de text que determina quin tipus de fill es vol afegir.

ChildrenLayout.createChildren(type, [callback]) **API Endpoint** (CMS_Angular_Meteor/imports/ui/components/childrenLayout/childrenLayout.js, line22)	
Descripció	Instancia un layout fill en funció del tipus que se li passi per argument. És una classe mirall que gestiona la creació de tots els objectes layouts fills.
Arguments	**type** String Cadena de text que determina quin tipus de fill serà creat. **callback** *Function* Funció que retorna un únic valor amb l'objecte del layout fill instanciat.

cssManager.generateClassId( ) **API Endpoint** (CMS_Angular_Meteor/imports/ui/components/cssManager/cssManagerService.js, line7)	
Descripció	Genera noms de classes aleatoris. Utilitzat a la creació de layouts i fills de layouts.

cssManager.openCssEditor(event, container) **API Endpoint** (CMS_Angular_Meteor/imports/ui/components/cssManager/cssManagerService.js, line18)	
Descripció	Obra la finestra flotant amb l'editor d'estils. Necessita un element d'estil per la seu correcte funcionament, en cas contrari no serà obert.
Arguments	**event** DOMEvent Esdeveniment del click generat per obrir la finestra flotant. **container** Style Object Objecte del tipus style amb el contingut dels estils de l'element a editar. L'objecte ha de constar d'un nom de classe juntament amb un array d'objectes de clau valor similar al CSS (propietat : valor).

CssManager **Component** (CMS_Angular_Meteor/imports/ui/components/layoutManager/cssManager/cssManager.js, line1)	
Descripció	Mostra l'editor d'estils, el qual permet modificar objectes de tipus style, eliminant, afegint o editant cada una de les propietats de l'array d'estils.
Mètodes	**addProperty** ([*Object* property]) (CMS_Angular_Meteor/imports/ui/components/layoutManager/cssManager/cssManager.js, line27) Afegeix una nova propietat d'estil a la llista d'estils de l'objecte style que està essent editat. El valor per defecte es pren del input de la finestra flotant. **removeProperty** (Object property) (CMS_Angular_Meteor/imports/ui/components/layoutManager/cssManager/cssManager.js, line42) Elimina una propietat d'estil de la llista d'estils de l'objecte style que està essent editat. **openCssEditor** ( ) (CMS_Angular_Meteor/imports/ui/components/layoutManager/cssManager/cssManager.js, line50) Obre la finestra flotant amb l'editor d'estils. És un mètode privat gestionat per l'API (CssManager.openCssEditor()). **closeCssEditor**([*Boolean* save]) (CMS_Angular_Meteor/imports/ui/components/layoutManager/cssManager/cssManager.js, line54) Tanca la finestra flotant amb l'editor d'estils. En cas que el paràmetre d'entrada sigui cert, aquest actualitzarà el valor de l'objecte d'estils que ha estat editat, en cas contrari es cancel·larà totes les edicions no guardades. No hi ha cap consulta ni ordre a la base de dades.

4.2.2 PageManager (CMS Angular Meteor/imports/ui/components/pageManager/)

Aquest mòdul té com a principal funció la d'editar, crear i eliminar qualsevol pàgina a través de la seva API (PageFacade).

PageFacade.deletePage(page, [callback]) **API Endpoint** (CMS_Angular_Meteor/imports/ui/components/pageManager/pageFacade.js, line 35)	
Descripció	Elimina de la base de dades la pàgina i tot el seu contingut. Envia un missatge de confirmació.
Arguments	**page** *Page* Objecte de tipus page que s'ha d'eliminar, és necessari que contingui la id de l'element. **callback** *Function* Funció que retorna l'error i la resposta del procediment. En cas d'èxit l'error equivaldrà a null.

PageFacade.save(page, [callback]) **API Endpoint** (CMS_Angular_Meteor/imports/ui/components/pageManager/pageFacade.js, line 56)	
Descripció	Guarda una nova pàgina a la base de dades, en el cas d'existir, serà sobreescrita.
Arguments	**page** *Page* Objecte de tipus page que s'ha de guardar. En cas de provenir de la base de dades ha de contenir l'atribut _id. **callback** *Function* Funció que retorna l'error i la pàgina guardada. En cas d'èxit l'error equivaldrà a null.

PageFacade.update(page, [callback]) **API Endpoint** (CMS_Angular_Meteor/imports/ui/components/pageManager/pageFacade.js, line 79)	
Descripció	Sobreescriu la pàgina a la base de dades, en cas de no existir retornarà un error.
Arguments	**page** *Page* Objecte page que s'ha de modificar. És necessari que contingui l'atribut _id assignat per Mongo quan va ser creat.

	callback *Function* Funció que retorna l'error i la pàgina actualitzada. En cas d'èxit l'error equivaldrà a null.

pageFacade.getPageById(pageId, callback) **API Endpoint** (CMS_Angular_Meteor/imports/ui/components/pageManager/pageFacade.js, line 91)	
Descripció	Retorna la pàgina corresponent, en funció del paràmetre id que se li passa al mètode, a través de la resposta del callback.
Arguments	**pageId** *String* Cadena de text amb el valor de la id de la pàgina que es vol cercar. **callback** *Function* Funció que retorna l'error i la resposta del procediment, en aquest cas la pàgina. En cas d'èxit l'error equivaldrà a null.

PageFacade.createEmptyPage( ) **API Endpoint** (CMS_Angular_Meteor/imports/ui/components/pageManager/pageFacade.js, line 100)	
Descripció	Retorna un objecte Page predeterminat, preparat per començar a ser editat a través de l'editor de pàgines.

PageResume **Component** (CMS_Angular_Meteor/imports/ui/components/pageManager/pageResume/pageResume.js, line 1)	
Descripció	Mostra el resum de totes les pàgines creades per l'usuari.
Mètodes	**setFilter** ([*String* filter]) (CMS_Angular_Meteor/imports/ui/components/pageManager/pageResume/pageResume.js, line 44) Modifica el paràmetre utilitzat per filtrar les pàgines de l'usuari **movePage** (*String* direction) (CMS_Angular_Meteor/imports/ui/components/pageManager/pageResume/pageResume.js, line 48) Determina cap a quina direcció es vol desplaçar l'usuari a través de la paginació. Les opcions de l'argument d'entrada són next o prev. **delete** (*Page* page) (CMS_Angular_Meteor/imports/ui/components/pageManager/pageResume/

	pageResume.js, line 56) Esborra una pàgina de la base de dades. Utilitza l'API del mòdul PageFacade.

PageEditor **Component** (CMS_Angular_Meteor/imports/ui/components/pageManager/pageEditor/pageEditor.js, line1)	
Descripció	Mostra l'editor de pàgines, el qual permet crear i/o modificar objectes de tipus page.
Atributs	**name** String (Bidireccional) Cadena de text amb el valor del nom de la pàgina. **layoutId** String (Estàtic –Opcional) Cadena de text amb el valor de la id de la pàgina que es vol editar.
Mètodes	**save** ([*Layout* layout]) (CMS_Angular_Meteor/imports/ui/components/pageManager/pageEditor/pageEditor.js, line 27) Guarda o actualitza la pàgina de la base de dades que està essent editat. Utilitza l'API del mòdul PageFacade. **delete** ([*Layout* layout]) (CMS_Angular_Meteor/imports/ui/components/pageManager/pageEditor/pageEditor.js, line 43) Elimina la pàfgina de la base de dades que està essent editat. Utilitza l'API del mòdul PageFacade. Redirecciona l'usuari si s'ha finalitzat correctament. **createPage** ( ) (CMS_Angular_Meteor/imports/ui/components/pageManager/pageEditor/pageEditor.js, line 67) Mètode que s'executa en el constructor, i te com a finalitat actualitzar tota la vista amb la pàgina que rep el component a través de la url. **addLayout** ( ) (CMS_Angular_Meteor/imports/ui/components/pageManager/pageEditor/pageEditor.js, line 43) Inserta un nou element Layout que estava guardat a la base de dades al contingut de la pàgina. Per fer-ho utilitza l'api LayoutManager.displayLayouts, la qual retorna tots els layouts que hagi seleccionat l'usuari. **removeLayout** (*Integer* index)

	(CMS_Angular_Meteor/imports/ui/components/pageManager/pageEditor/pageEditor.js, line 91) Elimina un layout de la pàgina que està essent modificat. El paràmetre d'entrada determina al posició a la que es trobava. **addNewLayout** ( ) (CMS_Angular_Meteor/imports/ui/components/pageManager/pageEditor/pageEditor.js, line 87) Crea i inserta un nou layout a la pàgina que esta essent editat a partir de l'API LayoutFacade.createEmptyLayout.

4.2.3 StateManager (CMS_Angular_Meteor/imports/ui/components/stateManager/)

Aquest mòdul té la principal funció d'editar i gestionar la jerarquia de les diferents pàgines que ha creat l'usuari. Només consta d'un component que actualitza la base de dades amb la nova jerarquia, ja que tots els usuaris tindran un objecte estat creat automàticament.

A més a més, no requereix d'endpoints públics, degut a que les consultes que siguin necessaris realitzar sobre aquest mòdul es faran a la base de dades directament, ja que s'han de fer sempre des del costat del servidor per part del compilador.

StateEdit **Component** (CMS_Angular_Meteor/imports/ui/components/stateManager/stateEdit/stateEdit.js, line 1)	
Descripció	Mostra el resum de totes les pàgines creades i guardades per l'usuari organitzades de forma jeràrquica. Permet editar quines pàgines nien d'altres i l'ordre d'aquestes.
Mètodes	**saveState** ( ) (CMS_Angular_Meteor/imports/ui/components/stateManager/stateEdit/stateEdit.js, line 1) Guarda l'ordre i la jerarquia de les pàgines. Envia un missatge de confirmació abans d'executar-se.

4.2.4 CompilatorManager (CMS_Angular_Meteor/imports/ui/components/compilatorManager/)

Aquest mòdul té com a principal funció la de compilar, comprimir i enviar al servidor extern de l'usuari tot el codi font creat i editat per l'usuari a través del panell d'edició del gestor de continguts.

CompilatorFacade.compile( ) **API Endpoint** (CMS_Angular_Meteor/imports/ui/components/compilatorManager/compilatorFacade.js, line 79)	
Descripció	Inicia tot el procés en cadena de compilació, comprimir i enviament dels arxius fonts.

CompilatorFacade.sendToBuild(state, pages) **API Endpoint** (CMS_Angular_Meteor/imports/ui/components/compilatorManager/compilatorFacade.js, line 84)	
Descripció	Inicia la primera etapa del procés de compilació i enviament dels arxius font, corresponent a la creació dels arxius a partir de l'informació emmagatzemada a la base de dades. En cas d'èxit s'executarà la següent etapa CompilatorFacade.CompileToZip.
Arguments	**state** *State* Objecte de tipus State que s'utilitzarà per generar la jerarquia de les pàgines de la web externa. **pages** Array <Page> Conjunt d'objectes Page associats a cada un dels estats i creats per l'usuari.

CompilatorFacade.compileToZip( ) **API Endpoint** (CMS_Angular_Meteor/imports/ui/components/compilatorManager/compilatorFacade.js, line 97)	
Descripció	Executa la segona etapa del procés de compilació i enviament dels arxius font, és concretament la fase de comprimir totes les carpetes i arxius en un únic arxiu d'extensió .zip. Una vegada finalitzada aquesta etapa sense cap error s'executarà la tercera i última fase CompilatorFacade.refreshWeb.

CompilatorFacade.refreshWeb( ) **API Endpoint** (CMS_Angular_Meteor/imports/ui/components/compilatorManager/compilatorFacade.js, line 107)	
Descripció	Duu a terme l'última etapa del procés de compilació i enviament dels arxius font, és concretament la fase d'enviament de l'arxiu comprimit al servidor extern.

4.3. FUNCIONAMENT DEL COMPILADOR

El compilador és un mòdul que té la finalitat d'actualitzar la pàgina web externa de l'usuari a partir de l'informació generada i emmagatzemada per l'usuari a la base de dades a través del panell d'edició i gestor de continguts, per fer-ho executa tres etapes de forma síncrona.

Primerament, tenim la primera fase del procés, la qual és l'encarregada de **crear els arxius font** de la pàgina web externa a partir de meta informació de la base de dades.

En segon lloc, tenim l'etapa de **compilació**, on el servidor crea un arxiu comprimit a partir de totes les carpetes i arxius generats en el pas previ.

I en darrer lloc, el servidor **actualitza** la pàgina web externa a través de diferents subetapes que es detallen a continuació, en el respectiu apartat.

4.3.1 Construcció

```
// ... skipped lines ... //

class CompilatorObject {
    constructor (state, pages) {
        return {
            _id    : state._id,
            owner  : Meteor.userId(),
            states : this.createObject(state, pages)
        };
    }

    createObject (state, pages) {
        // ... skipped lines ... //
    }
}

class Compilator {

    // ... skipped methods ... //

    sendToBuild (state, pages) {
        const compilatorObj = new CompilatorObject(state, pages);

        /** Creando archivos **/

        this.call('fileCreator', compilatorObj,
            (err,res) => {
                if(!err) this.compileToZip();
                /** Archivos creats correctament **/
            }
        );
    }
}
```

Aquesta primera etapa del procés de compilació i actualització del lloc web crida al mètode de Meteor anomenat *fileCreator*, aquest mètode no és més que una funció mirall cap a la funció *compileState*[19].

Els paràmetres d'entrada a la crida d'aquests mètode és un objecte anomenat CompilatorObj, tal i com es pot veure a la línia 22 de l'exemple. Aquest objecte fa referencia a la classe Folder present al UML del webCompilatorManager, i té el principal objectiu de instanciar un objecte amb els mateixos atributs que en el disseny, amb l'objectiu de mantenir una coherència en tot el procés de compilació.

Així doncs, el compilateState, la funció principal d'aquesta etapa, té l'objectiu d'iterar per cada un dels objectes del tipus pàgina presents a l'objecte CompilatorObj (Folder en el disseny), per tal de crear els arxius Html i CSS. Per fer-ho utilitza la llibreria Json2Html, comentada anteriorment, on només se li ha de passar l'objecte al mètode transform per tal d'obtenir una cadena de text amb format Html. Pel que fa el CSS, ha de transformar els objectes d'estil a una cadena de text, a més d'eliminar les cometes i afegir el nom de la classe amb els seus respectius claudàtors, el funcionament intern d'aquesta transformació el la realitza el parseJsonToCssSyntax[20].

Per cada una de les pàgines es creen dos arxius a través de la llibreria 'fs'[21] de node, en els quals a un se li introdueix el contingut de la cadena d'Html i a l'altra el del CSS, i es guarden amb les corresponents extensions a la carpeta que els hi pertoqui en funció de la jerarquia present als estats.

[19] Per a més informació consulti: https://github.com/jcampeny/CMS_Angular_Meteor/blob/master/imports/api/compilator/compileState.js#L4 Data d'última consulta: 01.01.2017

[20] Per a més informació consulti: https://github.com/jcampeny/CMS_Angular_Meteor/blob/master/imports/api/utils/functions.js#L3 Data d'última consulta: 01.01.2017

[21] Per a més informació consulti: https://nodejs.org/dist/latest-v6.x/docs/api/fs.html Data d'última consulta: 01.01.2017

4.3.2 Compilació

```
// ... skipped lines ... //

class Compilator {

    // ... skipped methods ... //

    compileToZip () {
        /** Inicia compilació **/
        this.call('compileToZip',
            (err, res) => {
                if(!err) this.refresWeb();
                /** Compilació realitzada correctament **/
            }
        );
    }
}
```

La següent etapa del procés de compilació té la finalitat de crear un arxiu comprimit d'extensió .zip amb tots els arxius font generats prèviament a través del procediment de construcció. La funció que podem observar a l'exemple anterior crida a un mètode de Meteor que, com passava en el cas anterior, esdevé una funció mirall cap a una factoria, la qual constà d'una operació de compilació, aquesta operació és la següent:

```
import { __DIR__  } from '../utils/functions';
import file_system from 'fs';
import archiver from 'archiver';

function compileToZip(userId, callback){
    const output = file_system.createWriteStream(__DIR__ + '.compiled_webs/' + userId +'.zip');
    const archive = archiver('zip', {
        store: true // Sets the compression method to STORE.
    });

    output.on('close', function () {
        const bytes = archive.pointer();
        callback(bytes);
    });

    archive.on('error', function(err){
        throw err;
    });

    archive.pipe(output);
    archive.directory(__DIR__ + '/.compiled_webs/' + userId + '/', '/');
    archive.finalize();
}
```

En aquest exemple podem veure en primer lloc les importacions, la variable __DIR__ conté el valor del *path* cap fins a l'arrel del nostre projecte, *file_system* és la llibreria de nodeJS destinada a la gestió d'arxius i finalment *archiver*[22], una llibreria de tercers que ens permet crear un arxiu d'extensió .zip.

[22] Per a més informació consulti: https://archiverjs.com/docs/ Data d'última consulta: 01.01.2017

Seguidament s'inicia la funció, a la línia 6 podem observar que creem el destí i el nom on volem guardar l'arxiu comprimit que crearem a través de la llibreria *archiver*, per seguidament seleccionat l'extensió de l'arxiu que volem crear juntament amb un objecte per la configuració de la llibreria.

A les línies 11 i 16 estem assignant funcions que escolten els esdeveniments que llençarà la llibreria quan executi el procés de compilació. Quan es generi un error, aquest serà retornat al client i s'aturarà tot el procés, en canvi a través de l'esdeveniment *close* podem saber quan l'arxiu ha estat creat correctament, amb la finalitat de retornar una resposta al client.

Finalment, les últimes tres línies serveixen per iniciar tot el procés de compilació, on cal destacar la funcionalitat de *directory*, la qual ens permet determinar com i quin directori es de guardar dins l'arxiu comprimit creat a la línia 6.

4.3.3 Enviament

```
1  // ... skipped lines ... //
2
3  class Compilator {
4
5      // ... skipped methods ... //
6
7      refresWeb() {
8          /** Iniciant comunicació amb el servidor extern **/
9          this.call('refreshWeb',
10             (err, res) => {
11                 if(res === 200){
12                     /** Arxius enviats correctament **/
13                 }
14             }
15         );
16     }
17 }
```

Un cop s'ha generat l'arxiu comprimit, s'inicia la tercera i última etapa del procés d'actualització de la web externa. Aquesta fase, a l'igual que les anteriors crida un mètode de Meteor, però en aquest cas, el mètode conté tota la lògica de gestió dels esdeveniments, i no ho delega, tal i com es mostra a continuació.

```
1  export function refreshWeb(){
2      const port = generatePort();
3
4      //Create and bring up the server to download files
5      const downloadServer = new DownloadServer({port, userId : this.userId});
6
7      const onFileDownload = (err, res)=>{
8          if(err) throw new Meteor.Error(400, err);
9          downloadServer.close();
10     };
11     downloadServer.up(onFileDownload);
12
13     //Send request to the external web
14     const promise =  new Promise((resolve, reject) => {
15         refreshExternalWeb((response) => {
16             resolve(response);
17         });
18     });
19
20     return promise;
21 }
```

Aquest mètode inicia generant un port a l'atzar, aquesta funció de la línia 2, no és més que una operació que retorna un valor aleatori enter amb un rang de valors vàlids pel que fa a ports lliures en el servidor. Així doncs, la variable port conté tant sols un valor enter, el qual serà el port on el servidor extern haurà de descarregar l'arxiu comprimit. Aquesta generació aleatòria de port és duu a terme per qüestions de seguretat, fent que si un usuari extern vol atacar al sistema descarregant-se arxius generats per altres usuaris, no li serà possible, ja que no coneixerà quin dels ports està obert.

Malgrat això, si un usuari anés provant cada un dels ports tampoc li seria possible, ja que com podem veure a la línia 5, quan el mètode s'inicia, és crea un servidor de descarrega, el qual està escoltant una sola petició http. Això significa que quan l'usuari actualitzi la seva pàgina web, hi haurà un espai de temps molt curt on es generarà un servidor escoltant un port aleatori, el qual es tancarà quan el servidor extern hagi realitzat la descarrega de l'arxiu comprimit.

D'aquesta manera es crea un sistema d'actualització i descarrega segur, sense la necessitat de creacions de API keys gracies a la classe downloadServer[23].

Aquest objecte DownloadServer, bàsicament aixeca un servidor en un port determinat a través de la llibreria http de nodeJS, el qual retorna un arxiu zip que tingui la id de l'usuari que està realitzant la descarrega. Un cop es finalitza qualsevol petició http al servidor, aquest executa

[23] Per a més informació consulti: https://github.com/jcampeny/CMS_Angular_Meteor/blob/master/imports/api/compilator/downloadServer.js Data d'última consulta: 01.01.2017

una funció, la qual la podem controlar des del mètode de Meteor, tal i com podem veure a la línia 7 de l'exemple anterior. En el nostre cas, quan la resposta de la transacció http sigui satisfactòria tancarem el servidor, per qüestions de seguretat, tal i com hem comentat anteriorment.

Un cop aixecat el servidor de descarrega en el port aleatori, és el moment d'enviar una notificació al servidor extern, aquesta notificació és un POST on s'adjunta el port on el servidor temporal de descarregar estarà esperant la petició. Aquesta funció l'execute el mètode refreshExternalWeb[24] de la factoria compileUtils.

En darrer lloc, quan el servidor extern rep la notificació a través d'una petició http de tipus POST, inicia la descarrega dels arxius al port del servidor que li hagi arribat a la petició. Un cop descarregat descomprimeix els arxius i els col·loca a la mateixa alçada jeràrquica on es troba l'arxiu que rep la notificació del servidor extern. A continuació es mostra el contingut de l'arxiu del servidor extern encarregat de descarregar el codi font comprimit:

```
1  <?php
2  if ( $_SERVER['REQUEST_METHOD'] == 'POST' ) {
3      $CMS_port = $_POST['port'];
4      $CMS_host = $_SERVER['REMOTE_ADDR'];
5      $url = $CMS_host . ':' . $CMS_port;
6
7      $zipCMS = file_get_contents('http://' . $CMS_host . ':' . $CMS_port .'/');
8
9      file_put_contents("downloaded_file.zip", $zipCMS);
10
11     $zip = new ZipArchive;
12     $res = $zip->open('downloaded_file.zip', ZipArchive::CREATE);
13
14     for( $i = 0; $i < $zip->numFiles; $i++ ){
15         $stat = $zip->statIndex( $i );
16         $zip->extractTo(__DIR__ , $stat['name']);
17     }
18 }
```

Com es pot observar és un arxiu molt senzill on a les primeres tres línies gestiona la recepció de l'informació a través de la capçalera de la petició per seguidament crear la url d'on ha de descarregar els arxius. La petició al servidor central es genera a la línia 7, on guarda tot el contingut descarregat en una variable, que posteriorment s'emmagatzema a un arxiu d'extensió zip, tal i com es mostra a la línia 9. Cal destacar que el servidor central es tanca quan la petició de la línia 7 finalitza, sense esperar la resposta de descomprimir els arxius.

[24] Per a més informació consulti: https://github.com/jcampeny/CMS_Angular_Meteor/blob/master/imports/api/compilator/webRefresher.js#L9 Data d'última consulta: 01.01.2017

Finalment, a partir de la línia 11, i a través de la llibreria ZipArchive[25] que ofereix el propi PHP a partir de les versions 5, descomprimim cada un dels arxius i els dipositem a la mateixa alçada jeràrquica que l'arxiu que gestiona tot aquest procediment.

[25] Per a més informació consulti: http://php.net/manual/es/class.ziparchive.php Data d'última consulta: 01.01.2017

5. CONCLUSIONS

La realització d'aquest treball va estar motivada per l'objectiu de brindar la possibilitat a maquetadors i dissenyadors a desenvolupar pàgines webs sense la necessitat d'haver de disposar de determinats coneixements tècnics, d'aquesta manera podrien realitzar projectes web estàtics de forma íntegra i individual.

Així doncs, al ser un projecte amb un abast gran i uns objectius ambiciosos, s'ha desenvolupat amb la finalitat de presentar un producte mínimament viable (MVP), tal i com es va marcar en els objectius inicials. D'aquesta manera, la plataforma no finalitza amb aquesta primera versió, sinó que seguirà creixent constantment en funció de les millores que se li vulguin incorporar. Les pròximes millores que s'incorporaran en un futur pròxim seran la de millorar la gestió del contingut multimèdia, permetre la gestió d'idiomes, optimitzar l'escalabilitat de la base de dades i de l'emmagatzematge d'arxius, possibilitat d'adhesió de plugins, millorar la llibertat en el desenvolupament del contingut i gestionar hosts propis.

Degut a que l'abast del projecte era molt gran i que des d'un principi es volia entregar un producte mínimament viable, es va planificar i dissenyar a través de la metodologia àgil anomenada Scrum, juntament amb una arquitectura modular, la qual permetia afegir noves funcionalitats a la plataforma mantenint un producte estable i potencialment lliurable en qualsevol moment.

Així mateix, gracies a aquesta planificació, ha permès obtenir un producte estable amb les funcionalitats inicials proposades, sense perjudicar en totes aquelles millores i adhesions que s'hi volen incorporar en un futur.

La realització d'aquesta projecte ha estat possible gràcies a l'experiència obtinguda en aquests darrers anys en el Grau Multimèdia, juntament amb coneixements obtinguts en el món laboral, els quals m'han permès planificar, dissenyar i desenvolupar amb èxit aquesta plataforma.

En primer lloc, la planificació d'aquests projecte s'han pogut aplicar coneixements d'organització adquirits a assignatures com gestió de projecte, juntament amb coneixements de planificació tècnica a nivell de software, com és el cas de disseny de base de dades, enginyeria web, disseny i programació orientada a objectes i anàlisi i disseny amb patrons.

En segon lloc, el disseny de l'interfície ha sorgit a partir de diferents etapes de recerca i investigació a nivell d'usabilitat, apresses a assignatures com arquitectura web i disseny d'interfícies multimèdia.

En tercer lloc, el desenvolupament de la plataforma ha pogut ser possible gràcies a tots aquells coneixements en programació obtinguts a través d'assignatures com programació web, programació orientada a objectes i ús de base de dades, entre d'altres.

A més a més, la realització d'aquest projecte m'ha permès, no només a aplicar els coneixements adquirits durant el Grau Multimèdia, sinó que me n'ha aportat de nous. Aquests nous coneixements han estat sobre el SEO, el qual he hagut d'investigar per tal d'optimitzar el funcionament del compilador; MeteorJS, bases de dades no relacionals, utilitzant com a gestor de les dades Mongo i diferents llibreries de NodeJS, com podria ser FS (FileSystem).

6. GLOSSARI

CMS -**C**ontent **M**anagement **S**ystem-

L'acrònim CMS, que es tradueix com a sistema de gestió de continguts, és un programa informàtic que permet crear i administrar continguts d'una pàgines web.

FTP -**F**ile **T**ransfer **P**rotocol-

Les sigles FTP responen a la traducció "Protocol de transferència d'arxius", i és doncs un protocol de xarxa per la transferència d'arxius entre diferents sistemes connectats a través d'una xarxa TCP.

HTML -**H**yper**T**ext **M**arkup **L**anguage-

El llenguatge de marques d'hipertext –traducció literal de les sigles en anglès- consisteix en un llenguatge estandarditzat d'etiquetes que s'utilitza per la creació de pàgines web.

CSS -**C**ascading **S**tyle**s**heets-

És un llenguatge destinat a la definició i creació de les presentacions de les pàgines webs i interfícies gràfiques desenvolupades amb HTML, i la seva denominació es tradueix com a "fulles d'estils en cascada".

JS

És l'abreviatura per descriure arxius i scripts escrits amb el llenguatge Javascript, els quals poden ser modificats per qualsevol editor de text. Aquests arxius són generalment executats pels navegadors web.

UML -**U**nified **M**odeling **L**anguage-

És un llenguatge unificat de modelatge destinat a modelar sistemes de software.

MVC -Model Vista Controlador-

Aquests acrònim respon a la descripció d'un patró d'arquitectura de software, el qual permet separar les dades i la lògica de negoci d'una aplicació de la interfície amb la qual interactua l'usuari.

PHP

Es tracta d'un llenguatge de programació del costat del servidor dissenyat pel desenvolupament web de contingut dinàmic.

JSON -Java**S**cript **O**bject **N**otation-

Es tracta d'un subconjunt de notacions literals d'objectes amb llenguatge Javascript.

API

És una interfície de programació que engloba diferents conjunts de subrutines, funcions i procediments que ofereixen determinades aplicacions amb la finalitat de ser utilitzades per qualsevol altre software independentment del llenguatge i la plataforma desenvolupada.

HTTP - Hyper**t**ext **T**ransfer **P**rotocol-

Aquest acrònim respon a un protocol que permet la transferència de dades i informació, generalment, a través de internet.

DNS -Domain **N**ame **S**ystem-

És un sistema de nomenclatura jeràrquica descentralitzat per dispositius connectats a xarxes IP, com per exemple Internet o xarxes privades.

SSH -Secure **Sh**ell-

L'acronim SSH és el nom del protocol i el programa que ho implementa, i s'utilitza per accedir a màquines externes a través d'una xarxa.

Host

Fa referència a totes aquelles màquines connectades a una xarxa que proveeixen i utilitzen els serveis d'aquesta, ja que aquesta xarxa ofereix serveis de transferència d'arxius, connexions remotes, servidors de base de dades, servidors web, etc.

Scrum

És una metodologia àgil destinada a gestionar el desenvolupament del software amb l'objectiu principal de maximitzar el retorn de la inversió per l'empresa.

Stakeholder

El seu significat és el de interessat o part interessada, i es refereix a totes aquelles persones o organitzacions als quals els hi afecten les activitats i decisions d'una empresa o producte.

Apache

És un servidor web HTTP de codi obert per plataformes Unix, Microsoft i Macintosh, entre d'altres.

Framework

És un entorn de treball per desenvolupar software, el qual depèn d'un llenguatge. Normalment integra components que faciliten el desenvolupament d'aplicacions com són el suport de programes, biblioteques o plantilles, entre d'altres.

7. BIBLIOGRAFIA

7.1 CONTINGUT.

BUILTVISIBLE. “The Basic JavaScript Framework SEO in AngularJS. Data última consulta: 25.11.2016. Link d’accés:

https://builtvisible.com/javascript-framework-seo/

JSON2HTML. “HTML Templating Engine”. Data última consulta: 16.12.2016. Link d’accés:

http://json2html.com/

MOZILLA DEVELOPER NETWORK. “Conocimiento compartido para la web abierta”. Data última consulta: 16.12.2016. Link d’accés:

https://developer.mozilla.org/es/

TSONEV, KRASIMIR. “A JavaScript library woth superpowers”. Data última consulta: 02.01.2017. Link d’accés:

http://absurdjs.com/

STACKEXCHANGE. “Stackoverflow”. Data última consulta: 28.12.2016. Link d’accés:

https://stackoverflow.com/

NODE.JS. “Documentation. Table of Contents”. Data última consulta: 27.11.2016. Link d’accés:

https://nodejs.org/dist/latest-v6.x/docs/api/

MONGODB.”MongoDB Documentatation”. Data última consulta: 07.01.2017. Link d’accés:

https://docs.mongodb.com/

ANGULARJS. “AngularJS API Docs”. Data última consulta: 05.07.2017. Link d’accés:

https://docs.angularjs.org/api

ANGULAR METEOR. “API Reference”. Data última consulta: 27.12.2016. Link d’accés:

https://angular-meteor.com/api/angular-meteor/1.3.11/

METEOR DEVELOPERS. “Introducing Meteor API Docs”. Data última consulta: 30.12.2016. Link d’accés:

http://docs.meteor.com/#/full/

ANGULAR. “Angular Material”. Data última consulta: 01.01.2017. Link d’accés:

https://material.angularjs.org/latest/

LINEARICONS. Data última consults: 02.01.2017. Link d’accés:

https://linearicons.com/

PHP. Vàries consultes. Data última consulta: 07.01.2017. Link d’accés:

http://php.net/

7.2 IMATGES.

FREEPIK.COM. “Flat people avatars inside circles”. Data última consulta: 07.01.2017. Link d’accés:

http://www.freepik.com/free-vector/flat-people-avatars-inside-circles_844579.htm#term=flat%20avatar&page=1&position=2

PREEPIK.COM. “Responsive web design”. Data última consulta: 07.01.2017. Link d’accés:

http://www.freepik.com/free-vector/responsive-web-design_851904.htm

8. ANNEX 1.

8.1. INSTAL·LACIÓ DEL PROJECTE.

1.1 Descarregar el repositori del projecte des de GitHub.

- **1.1.1** Instal·lar Git: https://git-scm.com/downloads
- **1.1.2** Accedir a través de la terminal a la carpeta on es vulgui descarregar el projecte.
- **1.1.3** Executar: $ git clone https://github.com/jcampeny/CMS_Angular_Meteor.git

1.2 Instal·lació de dependències pel servidor gestor de continguts.

- **1.2.1** Instal·lar NodeJS: https://docs.npmjs.com/getting-started/installing-node
- **1.2.2** Instal·lar MeteorJS: https://www.meteor.com/install
- **1.2.3** Instal·lar dependències del projecte
  - 1.2.3.1 Accedir a la carpeta on s'ha descarregat el projecte de GitHub.
  - 1.2.3.2 Executar $ meteor npm install

1.3 Iniciar el servidor de gestor de continguts.

- **1.3.1** Accedir a la carpeta on s'ha descarregat el projecte de github
- **1.3.2** Executar $ meteor

1.4 Creació del servidor extern (màquina virtual).

- **1.4.1** Instal·lar Vagrant de forma global https://www.vagrantup.com/downloads.html
- **1.4.2** Instal·lar plugin per evitar un error al muntar les carpetes compartides de la maquina virtualitzada: $ vagrant plugin install vagrant-vbguest
- **1.4.3** Accedir a la carpeta anomenada servidor_extern, present en els arxius font d'aquest projecte. És aconsellable moure-la a una altra localització en el dispositiu.
- **1.4.4** Aixecar el servidor $ vagrant up.

8.2. ACCÉS AL SERVIDOR GESTOR DE CONTINGUTS.

2.1 Accedir a través d'un navegador a: http://localhost:3000/

2.2 Iniciar sessió amb l'usuari i la contrasenya generades al iniciatlitzar el projecte en el punt 1.3.2. En qualsevol cas es pot iniciar sessió amb l'usuari admin@admin.admin i la contrasenya admin1234.

8.3. ACCEDIR A LA WEB GENERADA PEL SERVIDOR CENTRAL.

3.1 Accedir a través d'un navegador a: http://192.168.33.10/webcms/

Nota: Els arxius font del servidor extern es troben dins de la carpeta servidor_extern/shared_folder/web/webcms.

Nota 2: Dins de la carpeta exemple_compilador es troba una pàgina ja compilada per part del servidor central.