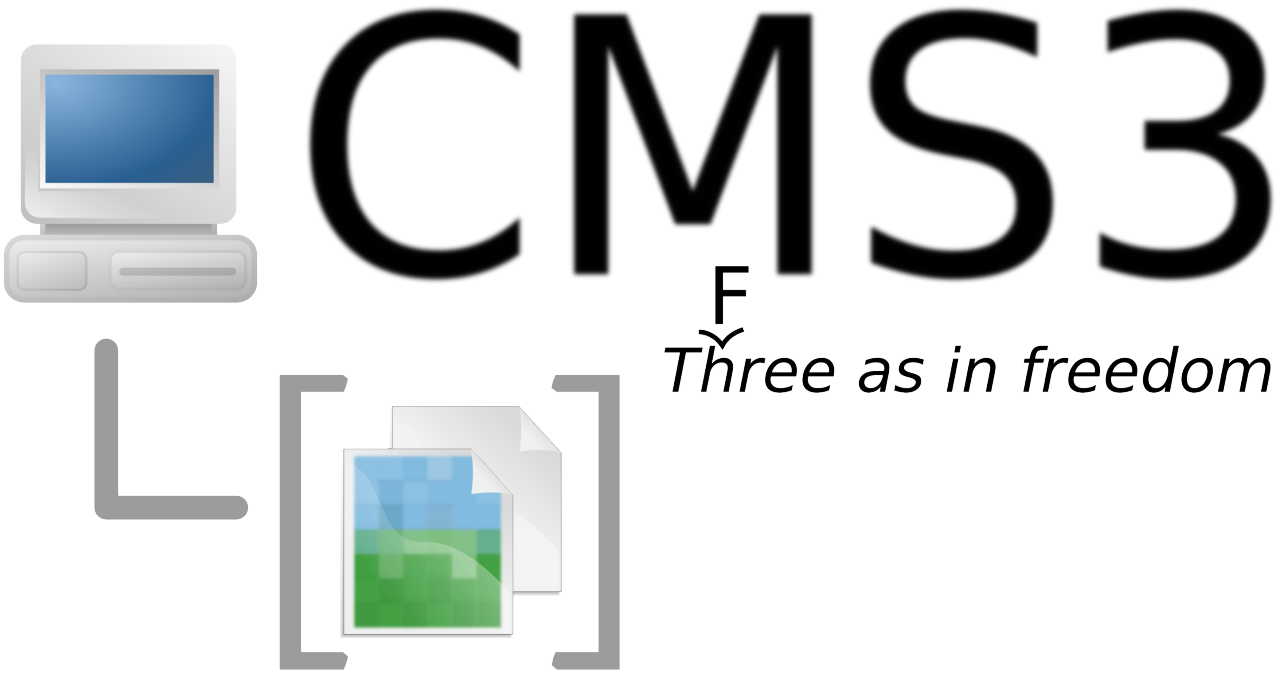
CMS3 Rapport

Af Jonas F. Jensen.



Indholdsfortegnelse

Forord

Denne rapport afslutter CMS3 projektet, og forklare hvordan systemet, samt nogle af mine overvejelser med hensyn til udviklingen af dette system. Rapporten er ikke en komplet dokumentation af systemet, kildekode dokumentation kan findes som vedlagt digital kopi, og anbefales som opslagsværk under læsningen af denne rapport. Specifikationer og konventioner for CMS3 kan findes i Appendiks A, B, C og D. Disse anbefales som opslagsværker under læsningen af denne rapport. Denne rapport er vedlagt et lille udsnit af mine kildekoder i printet form, hvis læseren ønsker at undersøge noget af kildekoden, anbefales dette udsnit. Igen anbefales det kraftigt at den bliver gennemgået med vedlagt digital kopi af kildekode dokumentationen som opslagsværk. Hele kildekoden og en virtuel maskine med opsat og installeret server er vedlagt som digital kopi. Brugsvejledning til dette kan findes i Appendiks F: CMS3 Server opsætnings vejledning. Ud gennem rapporten benytter jeg ofte engelsk termer, når jeg har været opmærksom på det har jeg skrevet fodnoter med deres danske oversættelse, og/eller forklaring.

Min primære desktop kører Ubuntu 6.10 og CMS3 er udviklet med relative primitive værktøjer, min primære tekst editor hedder gedit, og laver ikke andet end at understøtte syntaks highlighting. Til UML design og skelet generation har jeg benyttet Umbrello. Derudover har jeg benyttet phpDocumentor til generering af dokumentation. Et lille kommandolinje program kaldet sitecopy til synkronisering af test serveren. Jeg har også benyttet diverse kommando linje applikationer som ssh, gpg, tar og man. Nogle af disse bliver også kort beskrevet i rapporten. Sidst men ikke mindst har jeg benyttet vmplayer fra VMware til opsætning af virtuel test server, på test serveren har jeg benyttet Ubuntu 6.10 Server Edition og diverse server applikationer heriblandt apache, vsftpd osv.

Forudsætninger

Jeg vil i denne rapport forklareen lille smule teori, derfor er det en forudsætning at læseren har grundlæggende forståelse for php5 og OOP (objekt orienteret programmering) i blandt andet php5. Jeg vil ikke forklare OOP i denne rapport, under teori afsnittet vil jeg diskuttere opbygningen af en event model med interfaces. Men interfaces, abstrakte klasser osv. vil ikke blive gennemgået til denne rapport. Hvis læseren skulle ønske at læse op på noget af dette kan dokumentationen af php fra php.net anbefales.

Ydermere er en grundlæggende forståelse af *nix[1] systemer anbefalet. Her snakker vi om simpel shell programmering, benyttelse af kommandolinje applikation. Men dette er ikke strengt nødvendigt. Hvis læseren skulle ønske at læse op på dette kan linuxbog.dk om unix anbefales, men man kan også læse op på dette når eller hvis det bliver nødvendigt. Jeg vil benytte enkelte kommandoer i denne rapport, hvoraf nogle vil bliver forklaret. Mens dele af kildekoden vil kræve et grundlæggende kendskab til *nix systemer.

Sidst men ikke mindst er det forventes det selvfølgelig at læseren har grundlæggende forståelse af HTTP, (X)HTML, CSS, XML, XSLT og EMCAScript[2]. Forståelse af dette bliver i højgrad nødvendigt man har tænkt til at arbejde med kildekoden (produktet) vedlagt denne rapport.

1 *nix, er en forkortelse for Unix-like, altså unix ligende systemer. Hvorved der tales om LSB (Linux Standart Base) og/eller (u)certificerede implementeringer af the Single UNIX Specification.

2 EMCAScript, defineret i EMCA-262, er standarten som Javascript, Jscript og andre implementeringen implementere.

Projekt beskrivelse

I dette hoved afsnit finder du en kopi af den projekt beskrivelse som dette projekt er udarbejdet efter. Det vil sige at denne projekt beskrivelse blev skrevet før implementering af produktet og tilblivelsen af denne rapport.

Problem beskrivelse

Et problem jeg ofte er løbet ind i når jeg f.eks skulle hjælpe en af mine venner med at sætte hjemmeside op, er at man mangler et fornuftigt lille simpelt sikkert og brugervenligt CMS system. Samme problem har jeg med min egen hjemmeside, jeg vil nemlig gerne have et system der ikke har nogen høj indlærings kurve. Hvis man f.eks. kigger på nogle af de store opensource systemer vil man se f.eks. Joomla, Typo3, MediaWiki, WordPress osv. Man kan godt lave en mindre hjemmeside i disse systemer, problmet er bare at det kræver en mindre uddannelse at benytte dem. Dette skyldes at systemerne kan bruges til at lave en hel portal med.

Et andet problem med mange af de nuværende systemer er at man altid skal installere udvidelser manuelt. Opdateringer eller opgraderinger er ofte endnu værre her skal man sikre sig at man har taget backup manuelt, også skal man igennem en 2 siders guide før man har installeret opdateringen. Dette resultere i at folk ikke gider opdater deres systemer og de går hen og bliver sårbare overfor angreb. Jeg kender det alt for godt, jeg har nemlig både en WordPress blog og en MediaWiki installation som begge er af ældre dato.

Projekt beskrivelse

Min løsning på mit problem er må jo logisk nok være at bygge et CMS system, der er enkelte at benytte, ikke har formange uoverskuelige features og kan opdateres op en enkel måde. Helt konkret er ideen at jeg vil lave et system, som kan gemme data i en database og hente dem igen og præsentere dem for brugeren. Derudover skal der selvfølgelig være en administrations del hvor man kan redigere databasen på en enkel og brugervenlig måde. Det er det vi kender som et CMS system, men udover det skal mit system være opbygget i php5 med objekter, benytte mod_rewrite til at omskrive URL'erne og have en defineret plugin arkitektur.

Idéen er nemlig at alt funktionalitet i systemet skal komme fra plugins. Hvis man så vil have et forum på sin hjemmeside installer man blot et forum plugin. På den måde kan man undgå unødvendige funktionalitet i systemet, og gøre det mere enkelt og overskueligt. Lignende systemer findes allerede i dag, man kan i hverfald få udvidelser til mange CMS systemer. Men jeg vil gerne gå et ekstra skidt, hvis altså tiden er på min side. Jeg kunne nemlig godt tænke mig at implementere et mindre pakkehåndterings system. Sådan at udvidelser kan installeres, opdateres og opgraders gennem administrations delen på systemet. Hvorvidt jeg kan nå at implementere dette vil tiden vise, om ikke andet vil jeg have defineret et API som udvidelser kan bliver programmeret op imod. Det er idéen at systemet skulle kunne udvides med plugins til at kunne gøre stort set hvad som helst; mens systemet stadig kan fjerne alle udvidelser og derved indeholde stortset ingen funktionalitet. På denne måde kan systemet udvides efter behov, og samtidlig opdateres op en simple måde. Nu snakkede jeg lige før om at implementere et mindre pakkehåndteringsystem. Det skal lige siges at vi ikke snakker om at system på niveau med APT, RPM, Portage, PEAR osv. Systemet skal kunne installere, opdatere og fjerne pakker, men jeg tror jeg vil glemme alt om f.eks. pakke afhængigheder, da dette vil være vil være et projekt i sig selv. Så hvis jeg får tid til at pakkehåndterings system, så bliver det en meget simpel en af slagsen.

Derudover vil jeg gerne have administrations delen skrevet som en Web 2.0 applikation, dvs. med DHTML, Ajax og hvad der nu hører til. Der skal selvfølgelig også være et template system, men jeg tror det er en af de ting jeg vil ligge mindre vægt på. Selvom jeg vil arbejde lidt innovativt med hele output delen, jeg kunne nemlig godt tænke mig at plugins kunne udgive deres indhold semantisk og at systemet så derefter kunne formatere indholdet til det format brugere efterspørger.

Dette var en kort og meget overordenet gennemgang af hvordan jeg har tænkt mig at systemet skal se ud. Selvom jeg allerede nu har designet plugin arkitekturen, og har de første sekvensdiagrammer liggende, så er det ikke alt der er bestemt. Og det kan godt ende med at jeg forenkler implementeringen af nogle features, det kunne f.eks. betyde total udeladelse af pakkehåndteringsystem.

Tidsplan

Jeg regner med at jeg vil have selve grundsystemet implementeret omkring uge 12, hvorefter jeg så gerne vil have implementeret et par plugins der kan vise hvordan systemet virker. Afhængig af hvor lang tid det tager at skrive grundsystemet, vil jeg også gerne have implementeret et pakkehåndteringsystem.

Beskrivelse

CMS3 er blevet til et halv stort projekt, der består af omkring 5247[3] linjer php kode, og omkring 1200 linjer HTML. Det er derfor en smule uoverskueligt at beskrive alle features i detaljer, det anbefales at man prøver vedlagt virtuelle maskine af. Men her kommer i nu alligevel en gennemgang af de mest imponerede features. Holdt op i mod den projekt beskrivelse som ligger til grund for dette projekt.

Valg af projekt navn

I projekt beskrivelsen forud for dette projekt gav jeg ikke nogen forklaring på mit valg af projekt navn. Det er derfor på tide at komme med forklaringen på hvorfor det her hedder CMS3. Idéen var den at jeg ville tilføje en lille smule web 3.0, nu hvor alle sidder og er så vilde med deres web 2.0 applikationer, kunne det jo være lidt sjovt at se fremad. Web 3.0 er semantisk, det vil sige at alle informationer er skrevet i XML, og derfor kan læses og delvist forståes af computere. Faktisk findes der mange forskellige formater som benyttes på web 3.0 de fleste er XML baserede. Web 3.0 handler altså om at få adskilt indhold og præsentation.

Jeg nævnte med vilje ikke dette i min projekt beskrivelse, fordi web 3.0 er lidt svært at implementere og jeg gerne ville sætte fokus på andre steder, hvor php spiller en større rolle. Men navnet forpligter selvfølgelig, derfor skrev jeg at jeg ville arbejde lidt innovativt med template systemet. Hvilket har resulteret i udviklingen af SemanticOutputHandler, den giver dog ikke et helt rigtig web 3.0 output. I stedet printer den indholdet ud i XML og smider et XSL stylesheet oven på, men det er i hvertfald semantisk.

En anden mere personlig forklaring på navnet er at CMS3 det tredje CMS system jeg har udviklet. Det har selvfølgelig været mindre og andeledes simple systemer med fokus på fast indhold og template system. Faktisk er CMS3 måske i virkeligheden mit fjerde system, da det system jeg i øjeblikket kører på min hjemmeside er noget hjemmestrikket og meget hacket php.

3 4939 linjer php kode optalt med wc, dette er inklusiv kommentarer, inline dokumentation og licens header.

Plugin arkitektur

CMS3 har en plugin arkitektur, faktisk er slet ingen funktionalitet tilbage i systemet hvis man fjerner alle plugins, hvilket faktisk er muligt. Det eneste der er tilbage vil være en tom backend uden nogen frontend til pakkemanageren. Plugin arkitekturen er sådan et plugin implementere en række interfaces som så afgøre hvilken funktionalitet pluginet udbyder. Som minimum krav skal et plugin implementere IPlugin. Herefter kan den f.eks. implementere IProvidesEditor hvis pluginet vil have en editor integreret i backenden så brugere kan redigere indhold. Et plugin kan også implementere en masse andre plugins, for nærmere information og tilgængelige interfaces så kig i dokumentationen.

Et plugin kan også udbyde interfaces som andre plugins så kan implementere og på den måde kan integrer i hinanden. F.eks. udbyder Help pluginet et interface kaldet CMS3_Help_IProvidesHelp dette interface definere en metode kaldet GetHelp(), som returnere et array med hjælpe sider for et plugin. På måde kan plugins få præsenteret deres hjælpe sider i Help pluginet. Denne plugin arkitektur bliver også brugt som eventmodel, hvilket er beskrevet i teori afsnittet.

Frontend

Frontenden er den grafiske brugerflade som slutbrugere der besøger hjemmesiden bliver præsenteret for. Her er der et template system, som forklaret i afsnittet om template system. Men det er sådan at frontenden er delt op i namespaces. Fjern ordets oprindelige betydning, da denne ikke eksistere i php5, når jeg gennem resten af rapporten snakker om namespaces er højest sandsynligt med henblik på frontenden. Det er nemlig sådan at plugins kan registrere et namespace, så vil den information de udbyder til frontenden blive vist i det namespace. F.eks. en CMS3 server findes på ip'en 127.0.0.1, så vil man tilgå administrations delen ved at gå til 127.0.0.1/System. Fordi systemet har registreret System som namespace. Et plugin der udbyder information til frontenden, skal have registreret et namespace og skal implementere IProvidesContent.

Backend

I projekt beskrivelsen er backenden beskrevet som en web 2.0 applikation. Det er det også blevet til. Ajax er en god og meget brugervenlig ting, det samme er mod_rewrite, det har derfor været lidt svært at vægte med hvornår man skulle benytte Ajax og hvornår man skulle lade brugeren glide videre til en ny adresse. Jeg har valgt at ænder URL'en rimeligt ofte, blandt andet for at plugins skulle kunne linke til en backend editor fra frontenden (omtalte featureer ikke implementeret, men der er pladsholder til den og lignede features). På den anden side benyttes der Ajax mange når de forskellige plugins udfører opgaver, sådan at brugeroplevelsen bliver sammenhængende. CMS3 benytter et javascript framework kaldet Dojo, til at skabe flotte widgets og enkle ajax funktioner. Valget af Dojo bliver diskuteret under afsnittet CMS3 fra klientsiden. Backenden skaber også et kontrolpanel, som giver mulighed for at plugins kan lave konfigurations sider, har vil man f.eks. også finde frontenden til pakkemanageren.

Pakkehåndtering

CMS3 har et pakkehåndterings system, det er ikke så robust og vil få svært ved at overleve pakker der ikke er opbygget i henhold til pakkeformat specifikationen, se Appendiks D: CMS3 Pakke specifikation. Pakkemanageren kan installere, fjerne og verificere installerede plugins og pakker. Der skelnes mellem pakke og plugin, vi kalder det et plugin når det er installeret og en pakke når det ikke er installeret. Installationen foregår ved at brugeren uploader en pakke, eller vælger en pakke allerede uploaded på serveren. Dernæst verificeres pakken med asynkron kryptering, som forklaret i teori afsnittet. Når dette er sket kan pakke installeres, dette sker ved at nogle filer bliver pakket ud og evt. installations script til oprettelse af tabeller etc. bliver udført.

Når et plugin er installeret, kan man med pakkemanageren verificere deres integritet. Alle pakker indeholder en liste med sha1 tjek summer for alle vigtige filer, samt en digitale signatur af denne liste. På den måde kan brugeren lynhurtigt tjekke om et plugin er blevet modificeret, f.eks. af en black hat cracker. Det er også muligt at fjerne plugins igen. Hvorved brugerenkan uploade en nyere version af pakken og installere den, sådan kan et installeret system holdes opdateret. Men pakkemanageren er på ingen måder på højde med lignede systemer som f.eks. dpkg, rpm, portage, pear osv.

Template system

Som lovet i projekt beskrivelse har CMS3 en innovativ tilgangsvinkel til outputhåndtering. Hvis man kigger i dokumentationen for IProvidesContent, som er det interface et plugin skal implementere for at udbyde indhold til frontenden, vil man se at metoden GetPage, tager parameteret OutputHandler (overført som reference). Som det fremgår af dokumentationen er OutputHandler af type IOutputHandler. IOutputHandler definere det API som plugins skal bygge sig opad når de vil levere indhold til frontenden. Den implementere f.eks. metoden SetBody, der har parametret Body, og BodyList. Body parametret kan bruges hvis et plugin bare levere indhold. Men hvis et plugin vil levere indhold opdelt i afsnit, kan den sætte Body som hoved afsnit og BodyList som et array af strenge der hver især repræsentere et underafsnit. På den måde kan et evt. blog plugin skrive indholdet ud som forskellige afsnit og OutputHandleren kunne i virkeligheden genere et RSS[4] feed. Altså en form for adskillelse af indhold og præsentation.

Gallery pluginet benytter denne feature til at liste alle billederne ved dens toplevel side. StandartOutputHandler som genere normalt HTML, sætter bare afsnittene ind, som nye afsnit. Hvorimod SemanticOutputHandler parser det som XML, og har et tilhørende XSL stylesheet. SemanticOutputHandler er desuden et meget godt billede på hvordan indholdet kunne renders til andre formater. Idéen med template systemet i CMS3 er nemlig at få adskilt indhold og præsentation, i template systemets nuværende form er dette gjort meget simpelt. Template systemet er en af de ting der kunne være blevet lagt flere kræfter i, men som tidligere nævnt var det fra et programmerings synspunkt bedre at fokusere på php og ikke klientsiden.

Selve template systemet har ikke rigtig mulighed for at man nemt og enkelt kan ændrer menuen eller skifte templaten. Man kan i backenden, under system indstillingerne vælge hvilken OutputHandler der skal benyttes, og man kan selvfølgelig også installere nye OutputHandlers. StandartOutputHandler, har et sitemap, men ikke en menu. Alle plugins der udbyder indhold til frontenden, kan nemlig implementere et interface kaldet ILinkable hvorved de udbyder links. Det var oprindeligt idéen at den skulle gøre det lettere for plugins at linke til hinandens undersider, hvilket den også kan bruges til. Men i nuværende form bliver ILinkable kun benyttet til at lave et sitemap.

Udvidelser

Nu er CMS3 et plugin baseret system, og hvis man fjerner alle pakkkerne er der faktisk ingen funktionalitet tilbage. Derfor er der selvfølgelig udviklet et lille udvalg af plugins der viser nogle af funktionerne i CMS3. Disse kan både installeres, verificeres og fjernes med pakkermanageren. Dette er ikke en komplet gennemgang af alle plugins, da dette ville være mildest talt uoverskueligt.

4 RSS, Really Simple Syndication er en XML baseret standart for nyhedsfeeds, blog feeds og andre lister.

CMS3 giver mulighed for statistiske HTML sider gennem pluginet pPlainHTML. Dette plugin implementere en Ajax baseret WYSIWYG[5] HTML editor. Da PlainHTML benytter dojo frameworket til sine Ajax funktioner virker dette plugin også selvom Javascript er slået fra, dette er ikke tilfældet med alle andre Ajax implementeringer i systemet, men vist som et eksempel på at det kan lade sig gøre. Den gemmer så siderne i en tabel, der bliver oprettet ved installation af pakken. I frontenden kan man så få vist alle de sider som er oprettet i PlainHTML pluginet, bare ved at gå til det namespace som pluginet har registreret. Under kontrolpanelet kan man indstille hvilket namespace pluginet skal benytte, samt hvilken forsiden for det namespace pluginet har registret. Desuden registrere PlainHTML antallet af besøg, ved at ligge en cookie på alle besøgenes computer.

CMS3 giver også mulighed for at lave et simpelt galleri, med Gallery pluginet. Dette plugin benytter sig ikke af Ajax i editoren, dette skyldes både behovet for at vise at det kan lade sig gøre, men også den tekniske begrænsning af filtransaktioner med Ajax er meget komplekst, på grund af sikkerheds foranstaltninger på klientsiden. Gallery pluginet benytter sig ikke af databasen, i stedet benytter den det lokale filsystem. Dette plugin kan selvfølgelig både installeres og fjernes. I nuværende form indeholder pakken en række screenshots. Disse screenshots er med på listen over sha1 tjek summer, hvis du sletter nogle af de automatisk installerede screenshots og uploader nye billeder med samme navn, vil du ved verificering se at pluginet er blevet modificeret. En hurtig måde at lege lidt med plugin verifikations systemet. Det er altså med vilje at Gallery pakken indeholder screenshots fra start.

CMS3 har også et Wiki plugin. Som navnet antyder skaber dette plugin en wiki. Der er ingen backend editor til wiki'en, der er nogle få indstillinger af namespace i kontrolpanelet. Men ellers skal alt redigeres fra frontenden. Her kan man både oprette nye sider, se eksisterende sider, se oversigt over gamle revisioner eller få vist en af de gamle revisioner. Wiki pluginet benytter selvfølgelig en database tabel, og revisioners numrene er unikke, det vil sige at 2 sider aldrig opnår det samme revisions nummer. Wiki pluginet benytter TinyMCE som WYSIWYG editor, i stedet for et mærkeligt Wiki markup language som mange andre systemer gør det, heriblandt f.eks. MediaWiki.

I backenden giver CMS3 også mulighed for at man kan gennemse systemet log fil. Dette kan gøres med pluginet LogViewer, der kan findes i kontrolpanelet efter installation. Dette er også et plugin, men den udbyder ingen form for frontend til slutbrugeren. Der er altså mange måder at lave plugins i CMS3 systemet.

Bygge værktøj

For at bygge pakker, af plugins der er i overensstemmelse med pakkeformatet, som specificeret i Appendiks D: CMS3 Pakke specifikation, er der udviklet et værktøj kaldet C3Builder. C3Builder er en kommando linje applikation implementeret i php, og henter en smule inspiration fra Debians pbuilder værktøj. C3Builder er distribueret som en almindelig tarball[6] og kan bygges og installeres med GNU Make. C3Builder tager en del parametre, og har tilhørende man[7] side. C3Builder gør det let at bygge pakker til CMS3 systemet. Man opsætter blot en installation, begynder at oprette og redigere filer. Når man så har fået sit plugin til at virke, kan man bygge det med C3Builder. Hvorved der bliver skabt en sha1 liste med tjeksummer, digital signatur og hvad der nu ellers skal til for at skabe en CMS3 pakke. For yderlige information om C3Builder kan man kigge i Appendiks E: CMS3 Pakke vejledning, eller Appendiks I: C3Builder Manual Page for at se manual siden.

5 WYSIWYG, What You See Is What You Get er en editor med grafisk brugerflade, lidt ligesom Word, Frontpage osv.
6 Tarball, En tar.gz fil der oftest indeholder kildekode der ofte kan kompileres og installeres med GNU Make.
7 man, er den GNU Core utility man benytter til at vise manual sider i terminalen på en standart GNU/Linux distribution. En man side er så en manual side skrevet til man systemet.

Teori

Forklaring af nogle af den mest eksotiske teknologier som er har benyttet i CMS3.

Interfaces som eventmodel

Da php5 ikke har understøttelse for events[8] er man nød til at udvikle sin egen eventmodel. Formålet med en eventmodel er at videre give hændelser fra en klasse til en anden klasse, uden at den første klasse behøver være bekendt med eksistensen af den anden klasse ved implementering. En eventmodel er derfor meget vigtig når man bygger en plugin baseret applikation.

Den eventmodel som er benyttet i CMS3 er ikke særlig fleksibel og begrænser sig til kun at tillade kommunikation mellem forskellige plugins. Det er altså ikke en eventmodel som forskellige plugins kan benytte internt, men kun til kommunikation mellem et plugin og systemet eller andre plugins. Når man kigger nærmere på CMS3 kildekoden kan det godt være svært at se at der er tale om en eventmodel, den er da heller ikke benyttet som en traditionel eventmodel. Et godt eksempel på brugen af eventmodellen er IRemove og dens metode Remove, som bliver kaldet når et plugin bliver fjernet (uninstalled).

Eventmodellen er skabt med inspiration fra Java. I Java implementere en klasse et interface[9] for hvert event den abonnere på. Klasser der udbyder event har så en liste med klasse som implementere det event som den udbyder, når et event skal kaldes bliver listen med abonnerende klasser så løbet i gennem en loop og metoderne defineret i interfacet udført. I praksis benytter man nested classes[10] til at implementere de interfaces der repræsentere event med.

Nu er eventmodellen fra Java meget fleksibel og kompleks, derfor er det ikke helt den der bliver benyttet i CMS3. Personligt synes jeg Javas eventmodel mest af alt ligner et stort hack, dette skyldes nok min fortid i C#. Jeg har dog også kigget på en eventmodel for php inspireret af .Net, men denne var for stor, kompleks og lignede mest af alt et mindre framework i sig selv. En anden eventmodel, som også ofte er benyttet i php er hooks. Det går ud på at man har et array som indeholder tekst strenge med funktioner, som den skal kalde på når et event sker. Ved denne eventmodel, benytter man oftest ikke objekter og man har et array for hver event der kan forekomme. Denne model er blandt andet benyttet i MediaWiki.

I CMS3 skal alle klasser der vil modtage events være plugins, hvilket de bliver ved at implementere IPlugin. Derefter kan plugins implementere forskellige andre interfaces, som kan findes i mappen /share/*/, disse interfaces definere metoder som vil blive kaldet på forskellige tidspunkter. F.eks. bliver Install metoden fra IInstall, kaldet umidbart efter den fysiske installation af en pakke. Dette sker ved at alle plugins bliver lagret i et array på klassen CMS3_MainClass. Derefter kan man signalere en hændelse ved at løbe array'et igennem med en loop og udføre en bestemt metode på alle klasser som implementere et bestemt interface. Dette giver også mulighed for at event implementeringer kan returnere data.

8 Events, med danske termer hændelser.
9 Interface, med danske termer grænseflade.
10 Nested class, En klasse der kun eksitere som medlem på den klasse den er defineret i, se Javabog.dk for grundig forklaring af event modellen i Java.

Asynkron kryptering

For at kunne verificere integriteten af pakker og installerede plugins, som det er nødvendigt for en pakkemanager, skal man have en eller anden form for asynkron kryptering eller digital signatur, som jævne mennesker normalt omtaler det. CMS3 benytter sig af asynkron kryptering gennem GPG[11], som skaber OpenPGP signaturer som specificeret i RFC 2440. Denne signatur er ikke svarende til den danske digitale signatur som TDC udbyder.

Asynkronkryptering eller asymmetrisk kryptering om man vil, er det der ligger bag digital signatur systemer som f.eks. GPG. Det virker ved at man genere et nøgle sæt, når man gør det vil man få en offentlig nøgle og en privat nøgle. Afhængig af hvilken algoritme man har benytte kan man udfører forskellige opgaver med nøglerne. Man kan f.eks. signere et stykke data. Dette gør man ved at man først tager en hash sum af de data man vil signere. Derefter kryptere man hash summen med sin private nøgle, hvorved man får en signatur. Nu kan alle der har den tilhørende offentlige nøgle så dekryptere signaturen og verificere at hash summe matcher den man selv har generet af ens data. På denne måde kan man verificere integriteten af et stykke data. Man kan også kryptere et stykke data med en offentlige nøgle, herefter det krypterede stykke data kun kan dekrypteres af den private nøgle. På den måde kan man sende krypterede beskeder til en person uden at have et fælles password, som det er nødvendig ved almindelig synkron kryptering.

Kilde**:** http://en.wikipedia.org/wiki/Public-key_cryptography

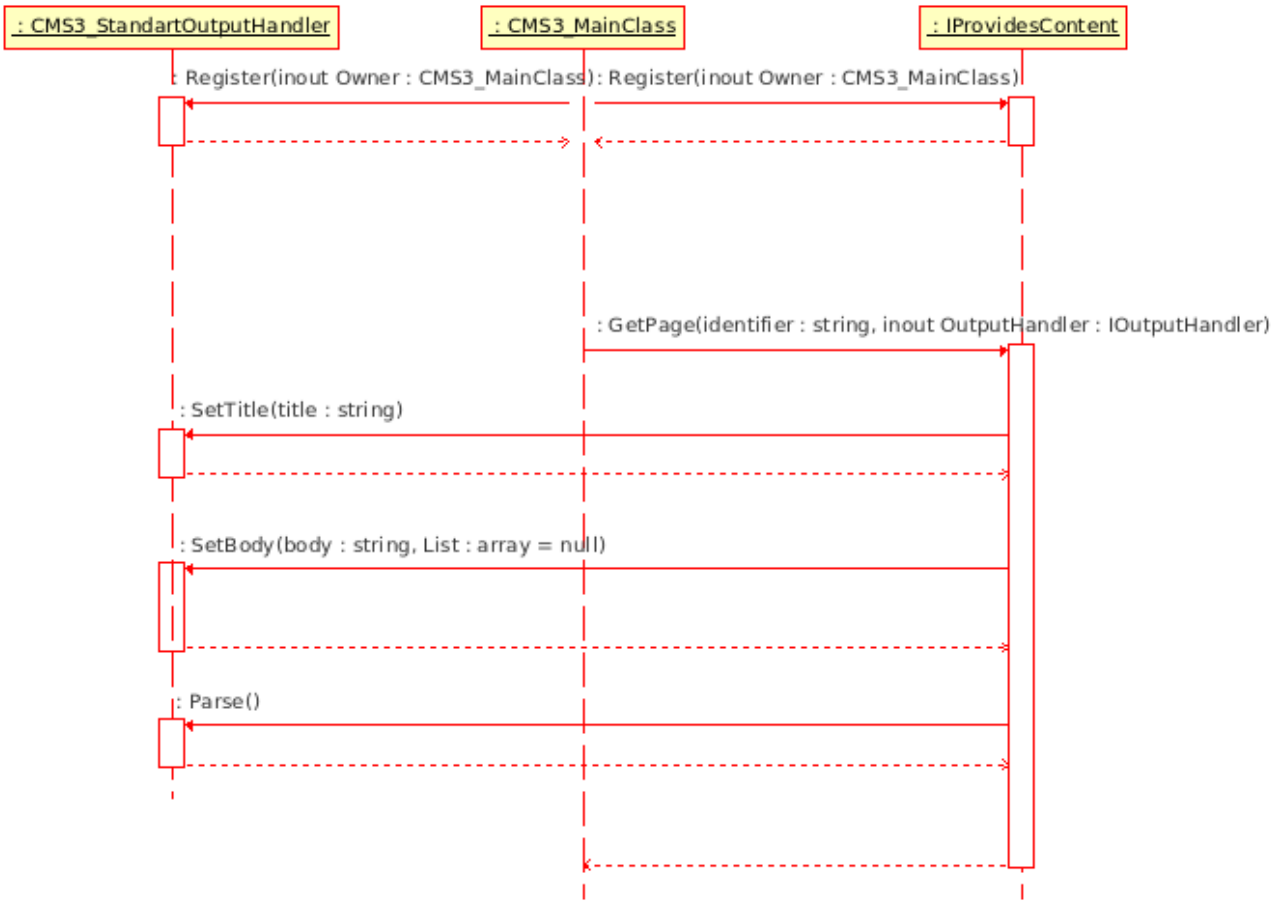
11 GPG, Gnu Privacy Guard en fri implementering af PGP, som defineret i RFC 2440.

CMS3 Grundsystem

En gennemgang af grundsystemet og hvordan det virker og opfører sig når det modtager en HTTP anmodning.

CMS3 grundsystemet består af klasserne CMS3_MainClass og CMS3_System samt en række interfaces, en abstrakt klasse og en implementering afIEmbedmentOutputHandler. Disse klasser udgør altså systemet. CMS3_MainClass ligger i default.php, og har struktur lidt lige som en C++ applikation, det vil sige at den har en Main metode som blive afviklet når filen bliver kaldt. CMS3_MainClass hoster alle plugins og benytter faktisk bare IProvidesEditor samt IOutputHandler. CMS3_System virker som et hvert andet plugin, bortset fra at den altid har namespacet System og hoster de backend relaterede interfaces som IProvidesEditor, ISystemCallback og IConfigure.

Nedenunder ses et UML sekvensdiagram som er blevet skabt før systemet blev implementeret. Jeg har ændret navnet på nogle enkelte metoder og sådan nogle småting, men ellers er der tale om det sekvensdiagram jeg opstillede før implementering:



Når serveren modtager en HTTP anmodning vil apache gå ind og læse mod_rewritereglerne. Her står der at hvis ikke der findes et bibliotek eller en mappe med det anmodede navn skal anmodningen behandles af default.php. Det er rigtigt at det kun er ./bin/ mappen der bør være HTTP adgang til, men er blot et spørgsmål om at få opsat de rigtige mod_rewrite regler, desværre var mine evner med regular expressions ikke nok til gøre dette på stående fod, og jeg valgte i stedet for at fokusere på andre områder.

Når anmodningen er landet hos default.php, bliver hele CMS3_MainClass klassen indlæst. Hvorefter en enkelt linje php kode nemlig "CMS3_MainClass::Main($_GET['i'])" starter CMS3 systemet. Først bliver konstruktøren for CMS3_MainClass udført, denne loader alle delte interfaces[12]. Dernæst loader den alle plugins, et plugin bliver loaded ved at man inkludere dens fil og opretter en udgave af objektet, og eksekvere metoden Register, som er defineret i IPlugin. Alle de loadede plugins bliver gemt i et array på CMS3_MainClass. Når det er gjort begynder bliver metoden ResolveIdentifier udført, denne metode opdeler den identifier der blev givet i querystring som parametret i, hvilket også er den sidste del af URL fra HTTP anmodningen, som blev parset af mod_rewrite. Ud fra identifieren kan man finde et namespace, dette namespace finder man registranten af og kalder denne op med den resterende del af identifieren. Ved opkald til registranten, som selvfølgelig implementere IProvidesContent, bliver systemet forvalgte IOutputHandler givet som parameter til GetPage metoden defineret i IProvidesContent. Herefter vil namespace registranten IProvidesContent, sætte indhold, titel og eventuel header udvidelse på IOutputHandleren, og bede denne om at parse indholdet. Hvis identifieren er tom, bliver default namespace som forvalgt i konfigurationen af systemet valgt.

Hvis identifieren f.eks. er "System/Edit/PlainHTML/Test" vil registranten til "System" blive kaldt med en OutputHandler og identifieren "Edit/PlainHTML/Test". Nu er registranten til "System", som tidligere nævnt, altid CMS3_System. CMS3_System benytter ikke standart IOutputHandler, i stedet benytter den en masse dojo og slår derfor IOutputHandleren fra med metoden Disable. CMS3_System vil nu splitte identifieren en gang til, og finde ud af at den skal kalde ModifyContent metoden på PlainHTML som defineret i IProvidesEditor. Denne metode bliver så kaldet med identifieren "Test" og systemets implementering af IEmbedmentOutputHandler. Nu vil PlainHTML så gå ind i en database tabel og finde Test, smide den i en editor, og nu får brugeren en Ajax applikation serveret.

CMS3_MainClass, optræder som medlemmet Owner på stortset alle plugins, altså Owner er en reference til CMS3_MainClass. CMS3_MainClass implementere en masse metoder, som kan være nyttige. F.eks. Log, MySQLLog eller GetDatabaseHandle, disse vil ikke blive dokumenteret her, men er grundig dokumenteret i dokumentation, som er et stærkt anbefalet opslagsværk.

Plugin arkitektur

Et plugin er defineret, i Appendiks E: CMS3 Pakke vejledning, som en klasse der hedder CMS3_<PluginID>, implementere IPlugin og ligger i filen ./lib/<PluginID>/CMS3_<PluginID>.php. En sådan klasse vil, som tidligere beskrevet, blive loaded af CMS3_MainClass. Dette er minimum kravene for et plugins, derefter kan et plugin implementere en lang række andre interfaces der definere dens funktionalitet. Nogle af de vigtigste interfaces er forklaret nærmere i Appendiks C: CMS3 Plugin udviklings guide, resten kan findes i dokumentationen. Plugin arkitekturen bygger også på nogle andre vigtige konventioner som f.eks. Navngivnings konventionen i Appendiks A og Filsystems konventionen i Appendiks B. Disse var bestemt før implementering af CMS3, Appendiks A: CMS3 Navngivnings konventionen var faktisk skrevet før implementeringen af CMS3. Disse interfaces og konventioner er nøglen til plugin arkitekturen.

Pakkehåndtering

Noget af det mest innovative ved dette projekt er pakkemanageren. Mulighed for at installere, verificere og fjerne plugins. Jeg har fået meget at inspirationen til dette fra forskellige Linux distributioner. Målet for pakkemanageren var selvfølgelig primært at kunne installere og fjerne plugins, hvorved man også let kan opdatere plugins. Dette et ikke så svært en øvet terminal bruger kan gøre dette med en enkelt kommando. Det ville i øvrigt heller ikke være så meget anderledes end bare at uploade filerne over HTTP i stedet for FTP, mange systemer som f.eks. WordPress og

12 Delte interfaces, interfaces i mappen ./shared betragtes som delte, se Appendiks B: CMS3 Filsystems konvention.

MediaWiki gør det muligt at ligge plugins ind ved at ligge en fil ind i en mappe, gøres ofte over FTP.

Derfor valgte jeg at implementere en form for verificering af pakkernes integritet. Dette er også en meget interessant ting, for at sikre at man ikke installer pakker fra folk man ikke stoler på. At verificere pakker en er en meget normal feature, som mange pakkemanagere gør. F.eks. Debians dpkg eller RPM fra LSB[13], om ikke andet verificerede i hverfald deres repositories. Men verificering af installerede plugins er ikke en særlig normal feature, ikke destromindre er den meget interessant. Fordi servere har det med at få besøg af black hats en gang i mellem, ved at verificer integriteten af sine installerede plugins kan man sikre sig at systemet er uberørt. Derfor valgte jeg at implementere verificering af plugins og pakker.

Først og fremmest skal pakkemanageren installere filer, dette gør den ved at pakke en tar[14] fil ud, i roden på en CMS3 installation. Alle filerne der skal installeres er så placeret i den tar fil, i deres respektive mapper relativ til CMS3 installationens rod. På denne måde kan man installere alle filerne ved at ud pakke en tar fil. Tar filen indeholder udover alle de filer der høre til pluginet, 2 andre filer. Disse er PlainHTML og PlainHTML.sig som ligger i mappen ./data/PackageManager/checksums/. PlainHTML filen indeholder en liste med sha1 checksummer og fil adresser, man kan så løbe listen igennem og tjekke om alle filerne er umodificerede. PlainHTML.sig er så en digital signatur af sha1 listen, underskrevet af udvikleren. På denne måde kan et plugins integritet vurderes efter installation. Selve tar filen er så placeret i en anden komprimeret tar fil sammen med en digital signatur af den første tar fil. Så kan den også verificeres før installation, der findes en mere detaljeret gennemgang af pakke formatet i Appendiks D: CMS3 Pakke specifikation.

Ved fjernelse af installerede plugins benytter pakkemanageren sig af filsystem layoutet specificeret i Appendiks B: CMS3 Filsystems konvention. Det betyder at ved fjernelse af PlainHTML bliver alle mapper med navnet PlainHTML i ./share/, ./lib/ osv. slettet. Man kunne godt argumentere for dette ikke er en særlig robust løsning. Det mindste en pakke afviger fra filsystems konventionen er nok til at bryde hele systemet. I stedet kunne man have valgt at registrer de filer en pakke installere i en database tabel, hvorefter man kunne fjerne disse igen. Dette ville være en interessant løsning, fordi det ville løse lidt op for filsystems konventionen.

Men filsystem konventionen har også andre fordele, hvis pakkemanageren på et fremtidigt tidspunkt skulle implementere support og avancerede opgraderinger. Så ville pakkemanageren nemlig kunne nøjes med at slette nogle af filerne, f.eks. kunne sletning af ./etc/<PluginID>/, undlades hvorved indstillingerne for det gamle plugin ville blive bibeholdt. Pakkemanageren implementere på nuværende tidpunkt ikke support for avancerede opgraderinger. I dens nuværende form er det kun muligt at tilføje/fjerne pakke, men der er reserveret plads i system designet til meget mere avancerede operationer. Filsystems konventionen muliggør også implementeringen andre operationer, som f.eks. backup af ./data/ og ./etc/, som så kunne genoprettes igen, hvis systemet skulle gå ned. Denne type backup operation er heller ikke implementeret på nuværende tidspunkt, men igen muliggør filsystems konventionen en interessant operation. Så selvom filsystems konventionen kan være træls, og ikke nødvendigvis er en særlig robust metode at implementere en pakkemanager på, så har metoden også mange klare fordele.

13 LSB, Forkortelsen for Linux Standart Base, som bliver implementere af de fleste distributioner heriblandt RedHat, Novells SuSE og mange andre.

14 Tar, Tape ARchive er en fil der indeholder andre filer ukomprimeret, altså et ukomprimeret arkiv.

CMS3 fra klientsiden

Nu er dette projekt godt nok programmering i php. Men da jeg har implementeret Ajax applikationer, der arbejder tæt sammen med serversiden, mener jeg at diskussionen om valg af klientside teknologier relevant.

Javascript framework

Da det var bestemt at CMS3 systemet skulle have en backend, der arbejdede som en Ajax applikation, var der mange overvejelser der skulle gøres. Jeg har tidligere selv kodet Ajax applikationer op fra bunden. Men dette er et halv stort arbejde, specielt når dette projekt skal laves i php. Derfor valgte at se mig om efter et javascript framework, gerne med tilhørende GUI[15] sæt. Der findes mange løsninger på dette område som er mere eller mindre lette at benytte.

Google har frigivet deres GWT[16]. GWT har nogle flotte og enkle widgets[17], de fleste kender dem nok fra Gmail, Google Calendar eller en af de mange andre Ajax applikationer fra Google. GWT er skabt til at benytte en Java på serversiden, og GWT skal da også programmeres i Java, hvorefter klientside logikken bliver komplieret med en specielt kompiler som oversætter det til Javascript, i stedet for bytecode. GWT kan vist nok benyttes med php, men på grund af dens tætte relationer til Java, fravalgte jeg den til dette projekt.

Yahoo har deres Yahoo! UI Library som også er opensource. Dette bibliotek er ikke afhængigt af noget serverside sprog, og har da heller ingen fortrukne sprog. Den leverede simple og enkle widgets, man havde ikke så mange af dem og de manglede nogle fede effekter. Der er, i modsætningen til GWT, heller ikke ret mange som er bekendt med Yahoo! UI Library. Derfor blev Yahoo! UI Library ikke valgt.

Der findes mange javascript frameworks derude, men til sidst faldt valget på Dojo. Dojo er et opensource javascript framework, uden officielt tilknytning til nogen serverside sprog. Derudover har den mange fede effekter, og rigtig mange widgets. En anden ting der er værd at nævne om Dojo er det store community der omgiver den, samt dens mange eksempler, gennemførte brugervejledning og store dokumentation. Det var mange rosende ord omkring Dojo, men jeg har da også oplevet problemer med Dojo. Dette skyldes måske at mit Javascript efterhånden har en 5-6 år på bagen, men bestemt også at Dojo ikke er specielt let, for folk der ikke er meget hardcore til Javascript. Heldigvis kan man komme meget langt med de eksempler som er givet, og de første 3-4 kapitler af Dojo håndbogen. En anden meget vigtig feature ved Dojo er at den bygger sig oven på eksisterende HTML, det vil sige at Dojo laver en textbox defineret med HTML om til en Dojo widget. Men hvis ikke javascript er aktiveret vil textboxen bare være der som en almindelig textbox er det. Altså en Dojo applikation er ikke nødvendigvis fuldstændig død, bare fordi Javascript er deaktiveret. Fordi Dojo kan modificere eksisterende HTML. Nogle gange skal man dog lave andre foranstaltninger, et eksempel er PlainHTML pluginet, som faktisk virker hundrede procent uden javascript aktiveret.

15 GUI, Graphical User Interface, på dansk en grafisk brugerflade.
16 GWT, Google Web Toolkit, opensource Java framework udviklet af Google Inc.
17 Widgets, En widget er en knap, tekstboks eller andet grafisk objekt som slutbrugere bliver præsenteret for. I Microsoft terminologi hedder det en controller, men storset alle andre steder bruger man widget.

Klientside struktur

Under dette afsnit burde klientside strukturen af være diskuteret, men sandheden er at der ikke findes nogen defineret klientside struktur. Dette har, til sidst i udviklingen, givet anledning til små konflikter hvor forskellige CSS definitioner overskriver hinanden. Manglen på enklientside konvention, skyldes nok primært at dette projekt har fokus på programmering i php og ikke Javascript. Derfor udgør klientsiden så lille en del af applikationen som muligt. Men det skal lige slås fast at en klientside konvention og en klientside API, nok ville have høj prioritet hvis udviklingen af dette system skulle fortsætte. På nuværende tidspunkt er der ikke nogen stor chance for konflikter, men hvis man f.eks. begyndte af opgradere Dojo versionen, ville man muligvis komme i problemer, da der ikke er defineret nogen procedure for denne type operation.

Konsekvenser

Nu vil jeg ikke våge at påstå at min applikation vil påvirke verden særligt meget eller overhovedet have nogle konsekvenser. Selvom jeg har planer om at frigiver kildekoden under GNU GPL på min blog, så er dette projekt ikke et af de projektet der nogensinde kommer ind i et produktions miljø[18], som nogle af mine andre projekter tidligere på året har gjort det. Alligevel kan den få konsekvenser for andre har lyst til at implementere et lignede system, og i deres research muligvis støder på mit. Det kunne også ske at jeg hev CMS3 frem fra skuffen hvis en af mine venner, lige pludselig står og mangler en hjemmeside med et brugervenligt redigerings system. Og i sådanne tilfælde ville CMS3 selvfølgelig få konsekvenser, men stadig kun i meget begrænset omfang. Hvis CMS3 havde været et stort projekt med mange udviklere bag sig, ville konsekvenserne muligvis være anderledes. Men jeg tvivler på at det ville få folk til at skifte til et nyt CMS system.

Perspektivering

Der findes lignede projekter i verden, på punktet med pakkehåndtering og plugin arkitektur er CMS3 ikke helt unikt. Her findes f.eks. PEAR, PHP Extension and Applikation Repository, som også er en pakke manager for php applikationer. Men jeg er ikke bekendt med nogle projekter som benytter en grafisk brugerflade oven på PEAR til at installere plugins med. Da jeg startede projektet regnede jeg heller ikke med at fandtes specielt mange andre systemer med denne feature. Det har dog vist sig at TYPO3 implementere en extensions manager. Desuden har den et specificeret pakkeformat og online repositories. Jeg er ikke sikker, men jeg tvivler på at den implementere verificering af installerede plugins på samme måde som CMS3. Jeg må dog indrømme at jeg ikke på stående fod er bekendt med andre systemer som implementere lignende features. Det er dog ikke unormalt at CMS systemer implementere en plugin arkitektur.

Videre udvikling

Herunder findes en kort præsentation af idéer som kunne være interessante at implementere i CMS3.

Avancerede opgraderinger

Avancerede opgraderinger dække over at opgrader uden at slette bruger information og indstillinger. Som kort forklaret i afsnittet om pakkemanageren er dette muligt såfremt filsystems konventionen overholdes. Dette ville være interessant at arbejde med, fordi det ville gøre opgradering processen meget hurtigere og endnu mindre smerteful. Dette ville nok også kræve en lille ændring i pakkeformatet, så det understøtter versions nummerering.

18 Produktions miljø, med dette begreb refereres der til det det engelsk term ”production environment”, som er en betegnelse for et kørende system, der benyttes til noget og derfor er relativt ”mission critical”.

Online pakke repositories

Mange andre pakke manager som f.eks. PEAR, har mulighed for at pakkerne kan downloades automatisk fra pakke arkiver på internettet. Dette ville gøre verificeringen endnu vigtigere, samt gøre det lettere at implementere mulighed for afhængigheder. Afhængigheder er f.eks. at PlainHTML pluginet implementere CMS_Help_IProvidesHelp interfacet, som ligger i ./shared/Help/ mappen, fordi den bliver installeret af Help pluginet. Hvis ikke Help pluginet er installeret, vil installationen af PlainHTML bryde CMS3 serveren, fordi interfacet CMS3_Help_IProvidesHelp ikke er defineret. Et plugin kunne så have afhængigheder, altså pakker der skal være installeret før installationen af selve pakken kan foregå.

C3Builder, MySQL support

En anden interessant feature ville være at implementere support for MySQL dumps i C3Builder. Hvis ellers navngivnings konventionen overholdes, burde det være muligt at loade MySQL indstillingerne fra system filen, og udfører en MySQLDump fra kommandolinjen. Hvorefter SQL filerne kunne inkluderes i pakken, dette vil selvfølgelig kræve en lille modifikation pakkeformatet. Men det ville lette plugin udviklings arbejdet betydeligt.

Internationalisation

I18n[19] ville faktisk være en meget interessant ting, som muligvis skulle udvides til at foretage små ændringer i pakkeformatet. I18n er den teknologi som benyttes til l10n (localization), altså i18n går ud på at skabe et API til oversættelse af brugerflade. En sådan arkitektur blev oprindeligt designet til CMS3, men den blev aldrig implementeret, da det ville være en stor opgave at portere eksisterende plugins til den i18n arkitektur der var planlagt. Den oprindelig idé var at lave en mappe der hed ./i18n/ hvor alle plugins så kunne have en mappe med deres PluginID, i deres respektive mapper kunne de så gemme php filer der definerede konstanter. Så kunne hver fil starte med en sprog kode, f.eks. da for danish og en for english. Systemet skulle så stå for at loaded den rigtige fil afhængig af sprog kode. Sproget skulle selvfølgelig automatisk opdages ved hjælp af HTTP headeren.

IEmbedable

Den opmærksomme læser har måske lagt mærke til at jeg har nævn IEmbedmentOutputHandler før. Dette er ikke nogen tilfældighed, systemet blev nemlig designet til at skulle kunne håndtere såkalte embedments. Altså sider der er inkluderede i andre sider. Dette skulle ske ved at alle plugins, der udbyder embedments skulle implementere IEmbedable. Som var designet til at specificere en metode til at få en bestemt embedment, lidt lige som GetPage på IProvidesContent bare med en IEmbedmentOutputHandler, og en metode til at returnere et træ med tilgængelige embedments. Det er nogle enkelte ud kommenterede funktioner, der viser at implementeringen af denne funktionalitet var planlagt. Den blev dog aldrig gennemført, fordi det var mere interessant at sætte fokus på andre områder.

19 I18n, internationalization sammen trækkes til i18n, da der er 18 tegn mellem i og n.

Klientside struktur og API

Som nævnt i afsnittet om klientside struktur, er der ikke nogen klientside struktur i CMS3. Dette skyldes som nævnt at dette projekt handler om programmering i php. Men ved fremtidig udvikling ville en klientside struktur og definition af et lille klientside API være meget relevat. F.eks. ville det være genialt hvis systemet implementerede en dialog boks, der gjorde det muligt at vælge mellem alle links udbudt med ILinkable. Sådan at f.eks. PlainHTML kunne benytte en link dialog fra systemet til at tillade brugen af linke til alle andre sider, der blot implementere ILinkable. Lige nu bliver ILinkable nemlig kun brugt til at skabe et sitemap med, oprindeligt var det idéen et ILinkable skulle ligge til grund for alle opret link dialoger.

WebDAV plugin

Nu vil jeg ikke nævne ti tusinde forskellige plugins som man kunne implementere. Men jeg vil godt lige nævne en idé jeg havde. Nu har jeg nemlig tidligere på året arbejdet med en WebDAV implementering i php. Og hvis CMS3 skulle videre udvikles i fremtiden ville implementeringen af et WebDAV plugin til f.eks. uploading af billeder for Gallery pluginet være interessant. Et WebDAV plugin skulle måske i virkeligheden blot udbyde et interface som andre plugins kunne implementere hvis de vil give brugeren filsystems adgang til en undermappe.

Output formatering

En anden ting der også kunne være interessant at arbejde videre med er output håndteringen. Den kunne godt være en andelse mere fleksibel. Det kunne også være rart hvis der var nogle OutputHandlers til f.eks. at levere indholdet som pdf. Så kunne OutputHandleren implementere IProvidesContent og registre namespacet pdf, og tilbyde alle andre sider i pdf formatet. Samme ting kunne gøres med RSS, og andre interessante formater.

Vurdering

Jeg vil vurdere at CMS3 systemet ikke er klar til produktions miljøer, men at det er et sjovt undervisnings projekt som der sikkert kan laves en masse hacks på endnu. CMS3 viser nogle funktioner som ikke ses mange andre steder, og forsøger at tage et alternativt og innovativt kig på verdenen af CMS systemer. Jeg vil vurdere projektet til at være gået meget godt, jeg er kommet længere end jeg beskrev i projekt beskrivelsen. Det kan godt være at jeg har omtalt nogle af de features som ikke er implementerede som værende designet og planlagt på forhånd. Dette betyder ikke at jeg ikke har nået det jeg satte mig for. Det betyder at jeg har vidst hvilken retning projektet skulle gå, og har været bevidst med at min arkitektur lukkede og åbnede for forskellige muligheder. Da jeg designede arkitekturen for CMS3, vidste jeg godt jeg ikke skulle regne med at lave online repositories eller WebDAV plugins. Men jeg var bevidst med at det skulle være muligt at implementere sådanne features i fremtiden, og at det derfor var nødvendigt med en hvis mængde fleksibilitet i systemet. Jeg vil også vurdere sikkerheden lidt, nemlig fordi der ikke er nogen. Det er faktisk den primære grund til at CMS3 ikke er egent til produktions miljøer. Der er ikke det der ligner foranstaltninger til at forhindre f.eks. simple SQL injektions. Jeg ved godt at sikkerhed er et nøgle ord i dag, specielt når vi snakker web applikationer. Men jeg har gennem dette projekt valgt ikke at fokusere på sikkerheden. Derfor må jeg slutte med at vurdere at CMS3 systemet er blevet meget godt. Men det er ikke egent til produktions miljøer, hvilket heller aldrig rigtigt har været planen.

Konklusion

Jeg må jo konkludere at det er muligt at lave en simpel, enkelt, flot og brugervenligt system, som kan lade bruger udvide systemet meget enkelt og sikkert. Jeg må konkludere at det er muligt at lave en plugin arkitektur, og at installere plugins som pakker. Dette kan i sig selv ikke være nogen overraskelse for nogen. Men jeg vil også konkludere at opsætning af server applikationer der benytter GPG er meget besværligt. Faktisk så besværligt at jeg ikke tror det er let at distribuere CMS3. Så selvom idéen er meget god, er min implementering ikke nødvendigvis perfekt. Det vil ganske enkelt være for svært for almindelige slutbrugere at opsætte systemet, det vil kræve en halv erfaren *nix ekspert. Derfor må jeg også være kommet frem til at verificerings systemer i php, der benytter asynkron kryptering, ikke bør wrappe omkring GPG. Men i stedet benytte brugerens GPG installation gennem et browser plugin som f.eks. FireGPG eller lave sin egen implementering af en asynkron krypterings algoritme i php. Jeg er også kommet frem til at en brugervenlig backend uden nogen særlig høj lærings kurve er mulig, uden at udelukke mulighederne for at udvide antallet af features efterfølgende.

Bilags oversigt

Oversigt over vedlagte bilag, disse bør benyttes som opslagsværker underlæsningen af denne rapport.

Appendiks A: CMS3 Navngivnings konvention

Appendiks B: CMS3 Filsystems konvention

Appendiks C: CMS3 Plugin udviklings guide

Appendiks D: CMS3 Pakke specifikation

Appendiks E: CMS3 Pakke vejledning

Appendiks F: CMS3 Server opsætnings vejledning

Appendiks G: CMS3 Kildekode udsnit

Appendiks H: CMS3 Screenshots

Appendiks I: C3Builder Manual Page

Digitale medier

en dvd med CMS3Server.

en dvd med CMS3Server Network Patched Edition.

en cd med:

- CMS3 Kildekoden
- CMS3 MySQLdump
- CMS3 Kildekode dokumentation