ARCADA

Webbplatsutveckling genom moduler i ramverket Drupal

Jens Nilsson

Examensarbete

Informationsteknik

2012

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	3726
Författare:	Jens Nilsson
Arbetets namn:	Webbplatsutveckling genom moduler i ramverket Drupal
Handledare (Arcada):	Thomas Forss
Uppdragsgivare:	Activeark Ab

Sammandrag:
Målet med examensarbetets praktiska del var att delta i utvecklandet av en webbplats med webbinnehållshanteringssystemet Drupal. Webbplatsen utvecklades för företaget Aberdeen Asset Management PLC, ett företag specialiserat på fastighetsinvestering och beställdes av digitala byrån Activeark. Webbplatsen skulle fungera som en marknadsföringskanal för företaget, där fastigheter och lokaler skulle presenteras och information gällande dessa upprätthållas. Målet nåddes genom att utvidga den funktionalitet som Drupal i sig själv erbjuder. Den utvidgade funktionaliteten består av anpassade moduler som ansvarar för att webbplatsens funktionalitet motsvarar den fördefinierade, samt av teman som sköter om att webbplatsens utseende följer den färdiggjorda layouten. Vid utvecklingen användes Drupal 7, vilken var den senaste stabila versionen av Drupal då projektet utfördes.

I detta examensarbete förklaras vad Drupal är och hur det är uppbyggt. Olika Drupal-relaterade tekniker och begrepp beskrivs kort för att ge en bättre helhetsbild. I arbetet läggs stor vikt på hur egna anpassade moduler skapas och hur Drupals färdiga funktionalitet kan utnyttjas av dem. I arbetet redogörs för en exempelmoduls funktionalitet genom att förklara olika funktioners betydelse med hjälp av kodsnuttar och skärmkapningar.

Nyckelord:	Drupal, Webbinnehållshanteringssystem, Webbplatsutveckling, Moduler, Webbapplikationsramverk
Sidantal:	53
Språk:	Svenska
Datum för godkännande:	14.6.2012

DEGREE THESIS	
Arcada	
Degree Programme:	Information Technology
Identification number:	3726
Author:	Jens Nilsson
Title:	
Supervisor (Arcada):	Thomas Forss
Commissioned by:	Activeark Ab

Abstract:
The objective with this thesis was to participate in the development of a website with the content management system Drupal. The website was developed for Aberdeen Asset Management PLC, a company specialized in real estate investment, and put into practice by the digital agency Activeark. The website was planned to work as a marketing channel for the company, where real estates and apartments would be presented and information regarding these could be maintained. The objective was reached by extending the built-in functionality in Drupal. The extended functionality consists of custom modules which are responsible of making the websites functionality match the predefined one, and themes that make the websites visual appearance match the predefined layout. Drupal 7 was used for the development as it was the latest stable version at the time of production.

In this thesis Drupal and its building blocks are explained. Also Drupal-related techniques and terms are explained for a better overall picture. Focus is laid on creation of custom modules and how the built-in functionality is used by them. An example module, created for this thesis, is explained and demonstrated by code snippets and screenshots.

Keywords:	Drupal, Web content management system, Website development, Modules, Web application framework
Number of pages:	53
Language:	Swedish
Date of acceptance:	14.6.2012

INNEHÅLL / CONTENTS

Figurer / Figures

FÖRORD

Idén för examensarbetet uppstod hösten 2011 på min arbetsplats Activeark. Jag hade redan tidigare bekantat mig med Drupal i samband med mina arbetsuppgifter och blivit speciellt intresserad av modulutveckling för Drupal, vilket gjorde valet av ämnet för examensarbetet lätt. Projektet blev färdigt och publicerades på vårvintern 2012.

Projektet var mycket intressant och lärorikt. Under projektet fördjupade jag mina existerande kunskaper samt lärde mig mycket nytt om Drupal.

Jag vill tacka min handledare Thomas Forss för all hjälp i skrivandet av examensarbetet. Jag vill också tacka Activeark och Aberdeen Asset Management PLC för möjligheten att göra praktiska delen av examensarbetet för dem.

1 INLEDNING

Detta programutvecklingsprojekt utfördes för digitala byrån Activeark Ab. Projektet går ut på att skapa en webbplats för företaget Aberdeen Asset Management PLC, ett företag specialiserat på fastighetsinvestering. Webbplatsen skall fungera som en marknadsföringskanal, där bl.a. fastigheter och lokaler skall presenteras på ett intuitivt sätt. Informationen i webbtjänsten skall vara lätt upprätthållbar i innehållshanteringsverktyget Drupal.

1.1 Målsättning

Arbetets målsättning är att redogöra för Drupals modularitet, uppbyggnad och dataflöde genom litteraturstudier och praktisk erfarenhet som fås via utvecklandet av webbplatsen för uppdragsgivaren. En målsättning är också att skapa och presentera en exempelmodul. I arbetet förklaras hur Drupal är uppbyggt och hur man genom att skapa egna moduler utvidgar den inbyggda funktionaliteten. För en bättre helhetsbild ges också en generell inblick i webbplatser och -teknologier.

1.2 Avgränsningar

Arbetets fokus ligger på att klargöra för de delar och egenskaper av Drupal som har varit relevanta i utvecklingen av den här webbapplikationen/webbplatsen. På grund av datasäkerhetsskäl innehåller arbetet inte tekniska lösningar som används vid utveckling av webbplatsen. Arbetet omfattar endast Drupal 7 som var den nyaste stabila versionen av Drupal när projektet utfördes.

2 WEBBPLATSER OCH WEBBTEKNOLOGIER

En webbplats är en samling HTML-dokument, bilder och andra filer som tillsammans utgör innehållet och funktionaliteten (interaktiviteten) för en webbplats. De här filsamlingarna befinner sig fysiskt lagrade på datorer som kallas webbservrar. Vanligen nås en webbplats med hjälp av en bläddrare som kommunicerar med webbservern genom datakommunikationsprotokollet HTTP. (Web site Encyclopædia Britannica Online 2012)

En webbplats kan likväl bestå av ett statiskt HTML-dokument som att utgöra en komplex webbapplikation med logik och datamanipulering (Web site DocForge 2011). För att underlätta och försnabba processen av applikationsutvecklingen är det vanligt att man använder ett webbapplikationsramverk som grund för applikationen (Web application framework DocForge 2011).

2.1 Webbapplikationsramverk

Ett lämpligt webbapplikationsramverk underlättar och hjälper utvecklaren i att bygga en applikation genom att erbjuda typiska kärnfunktioner för en webbapplikation. Exempel på sådana kärnfunktioner är användarhantering och datalagring. Detta betyder att utvecklaren inte behöver skriva alla funktioner själv utan kan använda färdiga funktioner som är integrerade i systemet. (Web application framework DocForge 2011)

Användningen av ett webbapplikationsramverk möjliggör återanvändning av kod vilket gör utvecklingstiden kortare samt förbättrar kvaliteten på koden. Koden är stabilare och säkrare p.g.a. att den är bättre testad och noggrannare granskad av flertal utvecklare. (Framework DocForge 2011)

Programmet blir också lättare att upprätthålla och uppdatera när ett mer färdigt känt botten används som grund. Webbapplikationsramverk brukar ha egna bästa praxis och regler när det kommer till att skriva kod, detta har också en stor betydelse för systemets upprätthållning samt möjlig vidareutveckling. (Framework DocForge 2011)

Exempel på webbapplikationsramverk är JavaServerFaces, CodeIgniter och ’Ruby on Rails’, vilka är alla implementerade i olika programmeringsspråk (Web application framework DocForge 2011). Ett system som också kan anses vara ett webbapplikationsramverk är innehållshanteringssystemet Drupal p.g.a. att det möter de egenskaper som ett webbapplikationsramverk förväntas ha. Till dessa hör t.ex. ett mallsystem, URL-hantering och abstraherad databasåtkomst. Förutom för innehållshanteringssystem specifika egenskaper erbjuder Drupal på funktioner genom vilka man kommer åt att ersätta fördefinierade kärnfunktioner och mallar med egna varianter. Drupal kategoriseras också som ett innehållshanteringsramverk för att det har egenskaper som tillhör både ett webbapplikationsramverk och ett innehållshanteringssystem. (Drupal Wikipedia 2012)

2.2 Innehållshanteringssystem

Ett innehållshanteringssystem är en webbapplikation som låter en eller flera användare skapa, editera och organisera innehåll som t.ex. text och bilder på en webbplats utan att göra ändringar i webb- platsens eller applikationens kod. Innehållshanteringssystem skapades för att göra publicering av innehåll på nätet lättare och för att göra publiceringssättet mer konsistent. Då publiceringssättet är mer konsistent är det också lättare för webbteam att tillsammans publicera och upprätthålla innehåll. (Content management system DocForge 2011)

Innehållshanteringssystemet nås via ett webbgränssnitt, där själva publiceringen och editeringen vanligen sker med hjälp av en editor som t.ex. WYSIWYG. En editor av den här typen påminner till utseendet om en simplifierad version av det kända ordbehandlingsprogrammet Microsoft Word. Editorns uppgift är att konvertera det innehåll som användaren har fyllt i och stylat i editorn till stylad HTML. I flera innehållshanteringssystem finns det tilläggsfält för innehållets namn, publiceringsdatum, kategori och författare, dessutom brukar innehållets placering på sidan anges t.ex. i form av en webbadress eller genom att välja under vilket navigationselement innehållet skall placeras i webbplatsens struktur. Ytterligare är det vanligt att ett innehållshanteringssystem har ett inbyggt användarhanteringssystem som sköter om registrering och inloggning till det administrativa webbgränssnittet och

möjliggör begränsad åtkomst till innehåll och inställningar baserat på användarens rättigheter. (Content management system DocForge 2011)

Huvuduppgiften för ett innehållshanteringsverktyg är att visa publicerat innehåll på en webbplats. P.g.a. att det finns olika typer av webbplatser finns det också flertal färdiga innehållshanteringssystem att välja mellan, en del med ett mer specifikt och andra med mer generella användningsändamål. Exempel på innehållshanteringssystem är Joomla, Wordpress och Drupal, som alla är utvecklade i programmeringsspråket PHP och har alla sina för- och nackdelar beroende på ändamålet. I vissa fall kan det anses lönsammare att bygga ett skräddarsytt innehållshanteringssystem för ett specifikt behov som inte möter någon funktionalitet de existerande erbjuder, i så fall lönar det sig att utgå från ett webbapplikationsramverk. (Content management system DocForge 2011)

Flera av innehållshanteringssystemen idag är byggda på ett modulärt sätt, vilket gör det enkelt för en administratör att selektivt välja vilka egenskaper som tas i bruk på webb-platsen eller applikationen. (Content management system DocForge 2011)

2.2.1 Modularitet

Ett modulärt uppbyggt innehållshanteringssystem gör det möjligt att utveckla moduler (tilläggsfunktioner) för ett existerande system vid behov. Modulerna är ofta beroende av funktioner som tillhör innehållshanteringssystemets funktionsbibliotek eller API medan innehållshanteringssystemets funktioner inte är beroende av modulens funktioner. Modulerna utvecklas skilt för sig och appliceras på sådana webbplatser och -applikationer där modulens funktionalitet behövs. (Content management system DocForge 2011)

Utgångspunkten är att en modul skall ansvara för en specifik uppgift som tas i bruk när modulen installeras. Om man vill utvidga funktionaliteten på en modul så att den drastiskt skiljer sig från modulens ursprungliga uppgift men ändå kräver funktionalitet från den, skapas en ny modul som är beroende av den ursprungliga. Detta betyder att ena modulen måste vara installerad för att den andra skall fungera. (Content management system DocForge 2011)

2.3 Programmerings- och skriptspråk

Ett programmeringsspråk är ett konstgjort språk skapat för att kommunicera med datorn. Ett programs beteende och funktionalitet bestäms av människoläsbar programkod med instruktioner för datorn, formulerade enligt det använda programmeringsspråkets regler. För att datorn skall förstå instruktionerna översätts programkoden till maskinkod med hjälp av en kompilator eller en tolk. En kompilator skapar en plattformspecifik exekverbar fil av programkoden medan en tolk exekverar programkoden medan den körs i en speciell exekveringsmiljö. Skriptspråk är programmeringsspråk som exekveras med hjälp av en tolk. (Programming language DocForge 2011)

I webbutveckling är användning av skriptspråk vanligt. Exempel på skriptspråk som används på webbservrar är PHP (Hypertext preprocessor) och ASP (Active Server Pages). Både PHP och ASP möjliggör interaktion med databaser och möjliggör producering av dynamiska dokument för webben. Exempel på sådana dynamiskt producerade dokument är webbsidor med blogginlägg och nyhetsflöden. (Php Wikipedia 2012)

I klienten (bläddraren) används vanligen skriptspråket Javascript, som används för att manipulera olika element i ett webbdokument och för att sköta interaktionen mellan användaren och webbplatsen. Det är också möjligt att göra förfrågningar till andra webbsidor i bakgrunden för att hämta ytterligare innehåll eller skicka inmatad data för vidare processering, denna metod heter AJAX (Asynchronous JavaScript and XML). (Php Wikipedia 2012)

2.4 Databaser

En databas är en samling data organiserad på ett för ändamålet relevant sätt, där data är lätt åtkomlig i rätt format. För att kunna skapa, upprätthålla och använda en databas behövs ett databashanteringssystem. MySQL är världens populäraste relationsdatabashanteringssystem med öppen källkod och används ofta på webbservrar i samband med Apache och PHP. (MySQL Wikipedia 2012) Ett annat databashanteringssystem som används på webbservrar är Microsoft SQL Server. Både

MySQL och Microsoft SQL Server baserar sig på en relationsdatabas modell. I en relationsdatabas lagras data i tabeller med rader och kolumner, där rader i en tabell kan ha relationer till rader i andra tabeller. (Relational Model Wikipedia 2012)

På webbplatser används databaser och databashanteringssystem t.ex. för att lagra webbplatsens innehåll och inställningar. Skriptspråk som PHP har API:n genom vilka applikationen kan utföra förfrågningar till databashanteringssystemet. Olika databashanteringssystem kräver särskilda API:n för varje skriptspråk. En webbapplikation kan byggas så att den använder sig av ett visst databashanteringssystem via en specifik API, vilket innebär att databasförfrågningarna måste skrivas om vid byte av databashanterare. Genom att använda ett webbapplikationsramverk som abstraherar databasförfrågningarna krävs ingen omskrivning vid bytet eftersom webbapplikationsramverket formulerar förfrågningarna beroende på det använda databashanteringssystemet. (Heinisuo 2001)

3 DRUPAL

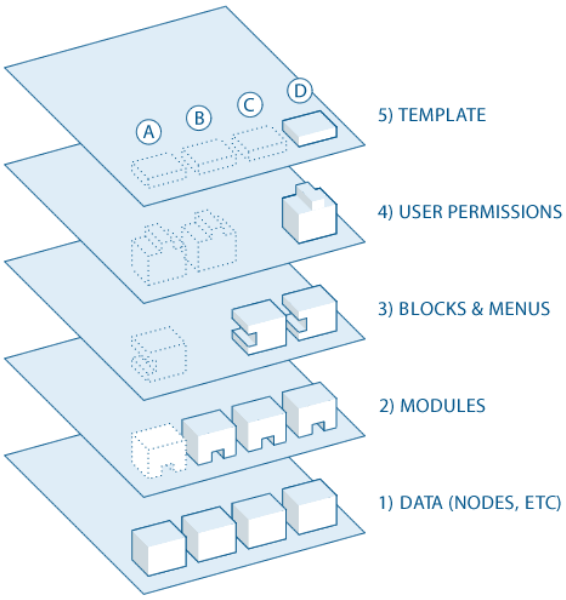
Drupal är ett modulärt webbinnehållshanteringssystem och ett kraftfullt ramverk för webbapplikationer (Melançon 2011). Med Drupal skapas många olika typer av webbplatser i olika storlekar som t.ex. nyhets publicering, företags presentation, internet butiker och sociala nätverk.

Drupal är öppen programvara licensierad enligt GNU General Public License, vilket innebär att källkoden är tillgänglig att använda och vidare distribuera för allmänheten. Drupal är en gemenskap och ett projekt som upprätthålls och vidareutvecklas av mer än 630 000 användare och utvecklare. (About Drupal 2012)

3.1 Uppbyggnad och dataflöde

Kärnan (eng. core) är standard paketet man laddar ner från drupal.org för att installera Drupal. Kärnan innehåller den basfunktionalitet som resten av systemet bygger på. Funktionalitet för bl.a. innehålls-, användar- och sessionshantering, samt allt som behövs för att skapa fundamentala sidor, finns inbyggt i kärnan. (Tomlinson & VanDyk 2010) Denna basfunktionalitet utvidgas med hjälp av moduler skapade av medlemmar i Drupal gemenskapen. Kärnan i sig själv är också uppbyggd av moduler.

För att demonstrera dataflödet genom Drupal kan systemet delas i fem lager: data, moduler, block och menyer, användarrättigheter och teman (The Drupal overview 2010). Dessa lager presenteras i figur 1 nedan.



Figur 1, Drupal delat i fem lager (The Drupal overview 2010)

Data lagras i dataobjekt som har en speciell uppsättning beroende på innehållet, som kan vara t.ex. en bloggartikel eller en produkt i en webbutik. Modulerna sköter om att lagra, hämta och bearbeta dataobjekten. Bearbetad data visas antingen på en eller flera sidor eller i återanvändbara block, beroende på hur det är inställt att visas. För att data skall visas på sidor eller i block måste användaren vara berättigad att se det, användarrättigheterna konfigureras med hjälp av funktionalitet inbyggd i kärnan. Temalagret bestämmer hur sidan och data på den ser ut, genom att lägga bearbetad data på bestämda ställen i ett HML dokument och applicera stilar och interaktivitet till det med hjälp av CSS och Javascript. (The Drupal overview 2010)

3.2 Entiteter

I Drupal lagras data i dataobjekt som kallas entiteter. En entitet är en särskild instans av en entitetstyp som t.ex. en bloggartikel eller en användare. En entitetstyp beskriver entitetens egenskaper. Exempel på entitetstyper är innehållsnoder och taxonomier. Entitetstypen delas ytterligare i knippen, t.ex. kan entitetstypen innehållsnod delas i bloggartiklar och produktbeskrivningar. (An Introduction to Entities 2011)

Konceptet med entiteter introducerades i Drupal 7 och härstammar från Drupals huvudsakliga innehållsobjekt noder. En nod är en entitetstyp och en nod kan delas i innehållstyper som motsvaras av knippen för entitetstyper. Idén med entiteterna var att få alla innehållsobjekt att bete sig som noder. (Melançon 2011)

3.2.1 Noder

En nod kan vara en bloggartikel eller ett recept, oberoende av vilken typ av innehåll det är frågan om är den underliggande datastrukturen samma. Detta möjliggör att modulutvecklare kan tillägga funktionalitet utan att ta i hänsyn av vilken typ innehållet är. Administratören kan sedan plocka ut väsentliga funktioner för respektive innehållstyper. Exempel på sådana funktioner är kommenteringsmöjlighet och möjlighet att lägga till filbilagor. (Tomlinson & VanDyk 2010)

När en nod skapas sparas grundläggande information om innehållet som titel, författare, datum för skapandet och innehållstypen. Ytterligare data som sparas är innehållstyps specifikt och definieras genom att lägga till fält (eng. fields) för innehållstyper. (About nodes 2011)

3.2.2 Fält

Fields-modulen gör det möjligt att lägga till anpassade datafält till entiteter som t.ex. noder, användarprofiler och kategorier. Modulen sköter om lagring, hämtning, editering och rendering av anpassad fältdata. Olika fälttyper hör inte till Fields-modulen utan sköts av tilläggsmoduler medan Fields-modulen sköter om infrastukturen för fälten och deras fästning till entiteter. I Drupals kärna finns stöd för följande fälttyper: text, numeriska, listor, taxonomier, bilder och filer. I flesta fall används inte Fields-modulen direkt utan man använder Field UI-modulen. (Working with Drupal 7 core Field module 2011)

Field UI-modulen bjuder på ett administrativt användargränssnitt för att skapa och hantera fält. De huvudsakliga valen som görs vid skapandet av ett fält är namnet på fältet, vilken typs data skall sparas i fältet, hur data matas in och visas samt hur många värden som kan sparas i fältet. (Working with content types and fields (Drupal 7) 2012)

När fälten är en gång skapade kan de återanvändas för olika entitetstyper eller knippen. Med att återanvända fält sparar man tid eftersom inte nya fält behöver definieras, dessutom kan innehåll visas, filtreras, grupperas och sorteras mellan olika innehållstyper med hjälp av fälten. (Working with content types and fields (Drupal 7) 2012)

3.2.3 Taxonomier

I Drupal används taxonomier för att klassificera webbplatsens innehåll och de kan vara en viktig del av webbplatsens informationsstruktur. Taxonomier kan ses som kategorier och webbplatsen innehåll går att gruppera under olika taxonomi termer. (About Taxonomy 2012)

Taxonomi termer grupperas i vokabulär, som t.ex. kunde musik vara namnet på en vokabulär som skulle innehålla termerna klassisk och jazz. Termerna går at organisera hierarkiskt med föräldratermer och barntermer, likaväl kan hierarkin lämnas bort som t.ex. med taggar. Det är möjligt att skapa relationer mellan termer genom att skapa term referensfält för vokabulären där termerna ingår. Vokabulären och termerna identifieras av unika id siffror, genom vilka de refereras och länkas till. (About Taxonomy 2012)

3.2.4 Användarhantering

Användarhanteringen sköts av användarmodulen (eng. User module) som möjliggör bl.a. användarregistrering och inloggning. Användarmodulen möjliggör också skapande av roller som går att tilldela användare. Varje användare tilldelas en eller flera roller. Rollerna kan delas olika rättigheter för att utföra olika operationer i systemet. I en ny installation av Drupal finns tre roller färdigt definierade: administrator, autentiserad användare och icke autentiserad användare. Det finns en sida med ett gränssnitt där användarna kommer åt att editera deras användarkonton. Icke autentiserade användare som också kallas gäster tilldelas ett unikt id som fungerar som en nyckel för att hämta data från servern. (User: access and management settings 2011)

3.3 Teman

Ett tema är en samling filer som bestämmer över sidans utseende. Teman består av CSS, JavaScript, bilder och mallfiler. Mallfilerna består huvudsakligen av HTML men innehåller också kodsnuttar som ersätts med dynamiskt innehåll då sidan genereras. (Tomlinson & VanDyk 2010)

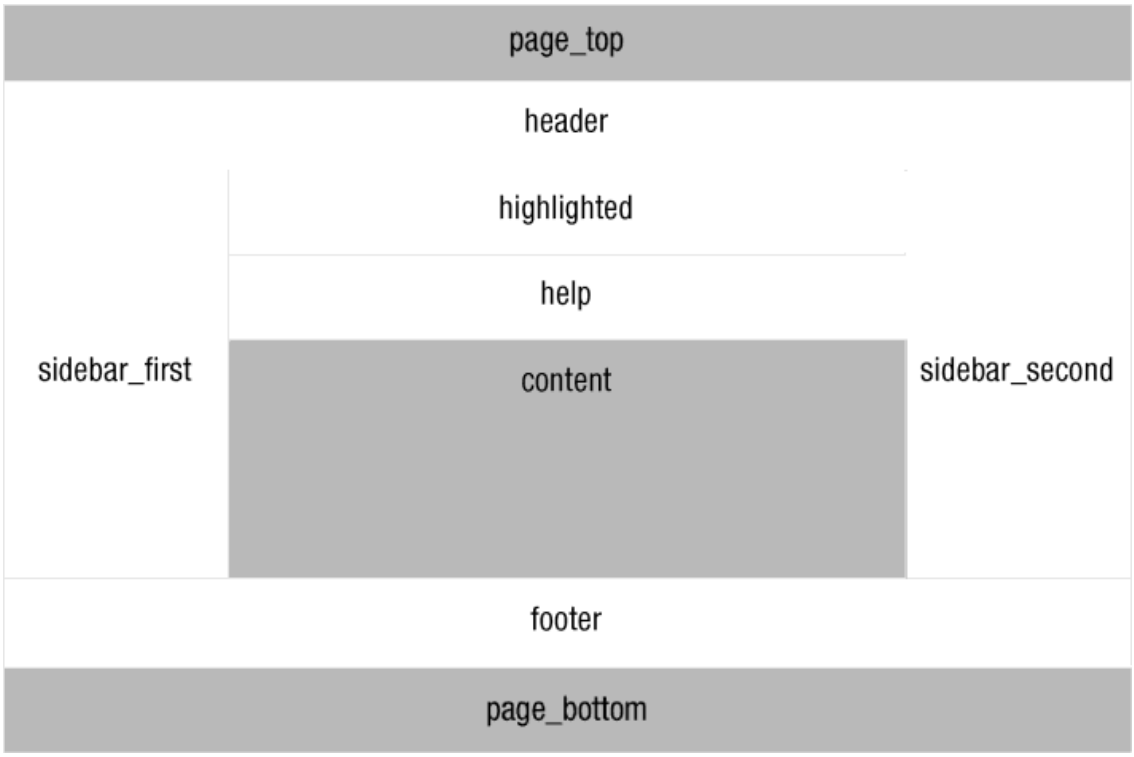
Drupals kärna innehåller fyra basteman: Bartik, Garland, Seven och Stark. Bartik är standard temat för nya Drupal installationer och Seven är standard temat för administrationsgränssnittet. Standard temat för både hela sidan och för den administrativa delen går att ställa in i administrationsgränssnittet, där det även går att ändra temaspecifika inställningar. Också globala temainställningar, såsom sidans namn och logo går att ställa in i administrationsgränssnittet. Kärnans teman är grundläggande och utgör endast en bas vars mål är att tillfredsställa alla. För att få ett för ändamålet passande utseende måste ett anpassat tema skapas. (Luisi 2011)

Varje tema har en infofil där basinformation och basegenskaper är definierade. Sådan information är bl.a. temats namn, beskrivning, vilken Drupal version som temat är gjort för samt vilka CSS-filer temat använder sig av. Om temat skall basera sig på ett annat tema och ärva dess egenskaper skall även bastemat anges i infofilen. I infofilen definieras också vilka regioner temat använder. (Luisi 2011)

3.3.1 Regioner

Största delen av innehållet på Drupal webbsidor är placerat i regioner. Typiska regioner är: huvud (eng. header), sidfot (eng. footer), sidofält (eng. sidebars) och innehåll (eng. content). Dessa regioner utgör ofta HTML-dokumentets struktur på högsta nivån. Temat har full kontroll över regionernas placering och visuella utseende. (Luisi 2011)

I Drupals kärna är nio regioner definierade (se figur 2 nedan). Om det inte finns några regioner definierade i temats infofil används regionerna som är definierade av kärnan. Regionerna som bör finnas med i varje Drupal tema för att upprätthålla en funktionell sida är: page_top, content och page_bottom. (Luisi 2011)

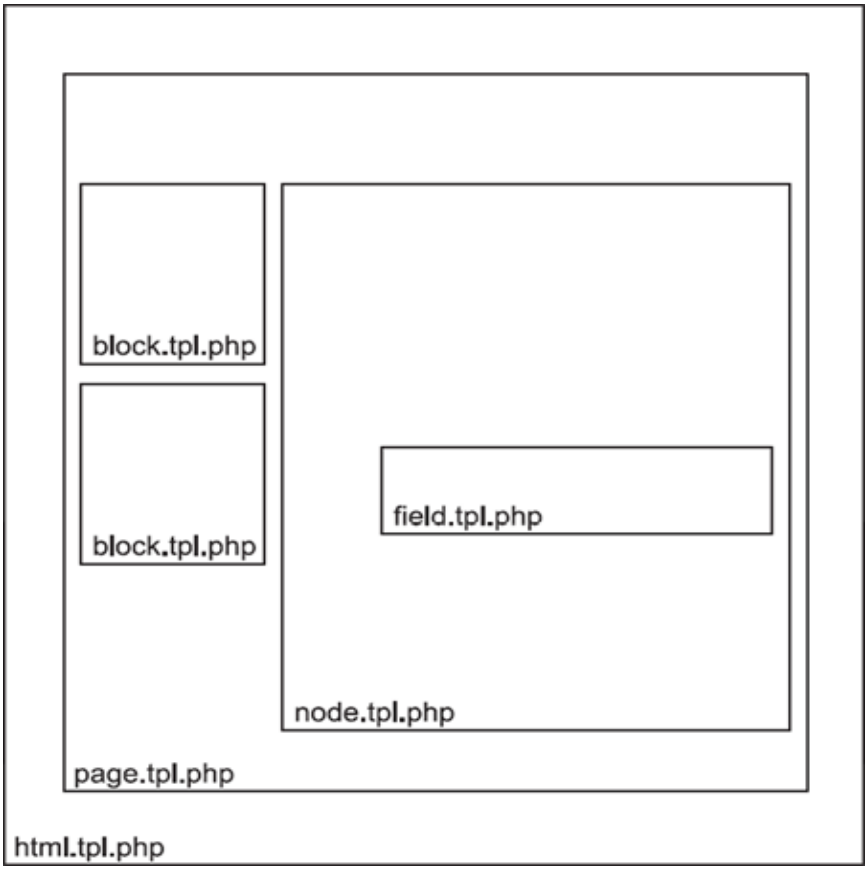


Figur 2. Drupals standard uppläggning av regioner (Luisi 2011 s. 285)

I planeringen av regioner finns det mycket att ta i beaktande som t.ex. designen, hurdan sorts innehåll det kommer att finnas, och hurdan roll regionerna allmänt skall ha i sidans layout. Förutom att regionerna skall definieras i infofilen skall de också skrivas ut i lämpliga mallfiler. (Luisi 2011)

3.3.2 Mallfiler

I Drupal sköts utskrift av HTML-kod till en stor del av mallfiler, de används både av kärnan och av moduler för att forma utskriften av sidan. Mallfilerna innehåller HTML-kod och PHP-variabler. En typisk webbsida skapad med Drupal består av mallfiler innanför mallfiler (se figur 3 nedan) samt tema-funktioner (eng. theme functions) som också kan påverka utskriften. (Luisi 2011)



Figur 3. Mallfiler innanför varandra (Tomlinson & VanDyk 2010 s. 199)

Mallfilerna som kommer med kärnan och modulerna utgör standardutskriften för respektive delar av sidan. Standardutskriften bestäms av ursprungliga utgivaren eller teamet. Drupals kärna innehåller över 40 mallfiler varav sex stycken är ansvariga för att bygga upp största delen av varje sida. De sex större mallfilerna är: html.tpl.php (se exempel i figur 4 nedan), page.tpl.php, region.tpl.php, block.tpl.php, node.tpl.php, comment.tpl.php och field.tpl.php. Om mallfilens standardutskrift inte är passande kan utvecklaren ersätta den med en egen mallfil. Ersättandet sker genom att hitta och duplicera den ursprungliga mallfilen. Ändringarna görs i den nya versionen medan den ursprungliga filen inte modifieras. (Luisi 2011)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+RDFa 1.0//EN"
"http://www.w3.org/MarkUp/DTD/xhtml-rdfa-1.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="<?php print $language->language;
?>"
version="XHTML+RDFa 1.0" dir="<?php print $language->dir; ?>"<?php print
$rdf_namespaces;
?>>

<head profile="<?php print $grddl_profile; ?>">
  <?php print $head; ?>
  <title><?php print $head_title; ?></title>
  <?php print $styles; ?>
  <?php print $scripts; ?>
</head>
<body class="<?php print $classes; ?>" <?php print $attributes;?>>
  <div id="skip-link">
    <a href="#main-content"><?php print t('Skip to main content'); ?></a>
  </div>
  <?php print $page_top; ?>
  <?php print $page; ?>
  <?php print $page_bottom; ?>
</body>
</html>
```

Figur 4. Exempel på html.tpl.php fil (Tomlinson & VanDyk 2010 s. 201)

3.3.3 Temafunktioner

Temafunktionernas mening liksom också mallfilernas är att forma HTML-utskriften så att den är överskrivbar av anpassade teman (Luisi 2011). När Drupal vill generera HTML-utskrift för en komponent som t.ex. en nod eller ett block söker den efter en mallfil eller en temafunktion som genererar utskriften för ifrågavarande komponent (2010).

Vanligen definieras temafunktioner av kärnan eller av moduler men de kan också definieras av teman. För att Drupal skall veta att en temafunktion existerar används hook_theme-funktionen (se exempel i figur 5 nedan). (Luisi 2011)

```php
<?php
/**
* Implements hook_theme().
*/
function mymodule_theme(){
  return array(
    'my_theme_hook' => array(
      'variables' => array('parameter' => NULL),
    ),
  );
}
?>
```

Figur 5. Exempel på användning av hook_theme-funktionen (Luisi 2011 s. 301-302)

```php
<?php
function theme_my_theme_hook($variables){
  $parameter = $variables['parameter'];
  if (!empty($parameter)){
    return '<div class="my-theme-hook">' . $parameter . '</div>';
  }
}
?>
```

Figur 6. Exempel på en temafunktion (Luisi 2011 s. 302)

I figur 5 visas hur man i en modul informerar Drupal om att en my_theme_hook temafunktion existerar, och tack vare denna information söker Drupal efter en temafunktion med namnet theme_my_theme_hook som visas i figur 6. Temafunktioner kallas inte direkt med funktionsnamnet utan med hjälp av theme-funktionen. För att kalla på theme_my_theme_hook funktionen kallas theme-funktionen istället på följande sätt: theme(’my_theme_hook’, $variables). Genom att kalla på temafunktionen via theme-funktionen dirigeras funktionsanropet till den lämpliga temafunktionen. (Luisi 2011)

Temafunktioner går att ersätta i ett anpassat tema. Detta sker genom att man kopierar ursprungliga temafunktionen till anpassade temats template.php fil och ersätter ’theme_’ med ’yourthemename_’ i början av funktionsnamnet. (Luisi 2011)

3.4 Block

Block är bitar av information som kan visas i ett temas regioner. Ett block kan vara en nod, en lista av noder, en kalender eller en online-enkät. I en standard installation av Drupal finns flera block färdigt definierade. Exempel på sådana är inloggningsformulär, sökformulär och en lista på senaste kommentarer. Flera av kontribuerade moduler innehåller block som en del av sin funktionalitet. (Tomlinson & VanDyk 2010)

För att definiera nya block används Drupals webbgränssnitt eller alternativt kan de skapas genom programmering med hjälp av block API i moduler. Block som innehåller statisk HTML kan vara bra att skapa som ett anpassat block via webbgränssnittet medan block som kommer att innehålla dynamiskt innehåll skapat med PHP kan vara bättre att definiera i en modul med hjälp av block API. På det här sättet skiljer man på logik och innehåll, ifall det behövs ändringar i logiken görs ändringarna i modulen medan ändringar i innehåll kan göras via webbadministrationsgränssnittet. (Tomlinson & VanDyk 2010)

3.4.1 Konfigurering av block

Ett vanligt konfigurationsalternativ som ställs in är blockets synlighet. Ett block kan göras synligt för specifika sidor, innehållstyper, användare och användarroller. Konfigureringen sker på blockadministrationssidan i Drupals administrationsgränssnitt. På blockadministrationssidan ställs också in i vilka regioner blocken visas samt i vilken ordning blocken visas i en region om det finns flera block i en och samma region. (Tomlinson & VanDyk 2010)

3.4.2 Block via programmering

Programmatiskt definieras block i hook_block_info-funktionen av moduler, där flera block kan definieras på samma gång. När blocket är definierat blir det också synligt och konfigurerbart på blockadministrationssidan. (Tomlinson & VanDyk 2010)

```
/**
 * Implements hook_block_info().
 */
function approval_block_info() {

  $blocks['pending_comments'] = array(
    'info'             => t('Pending Comments'),
    'status'           => TRUE,
    'region'           => 'sidebar_first',
    'weight'           => 0,
    'visibility'       => 1,
  );
  return $blocks;
}
```

Figur 7. Exempel på användning av hook_block_info-funktionen (Tomlinson & VanDyk 2010 s. 231)

I exempelanvändningen av hook_block_info-funktionen i figur 7 här ovan visas hur en modul definierar pending_comments-blocket. Vid det här laget är blocket synligt och konfigurerbart på blockadministrationssidan men själva blocket saknar innehåll. I figur 8 nedan demonstreras användningen av hook_block_view-funktionen där blockets innehåll och rubrik definieras. (Tomlinson & VanDyk 2010)

```
/**
 * Implements hook_block_view().
 */
function approval_block_view($delta = '') {

  switch ($delta) {
    case 'pending_comments':
      $block['subject'] = t('Pending Comments');
      $block['content'] = approval_block_content($delta);
      return $block;
      break;
  }
}
```

Figur 8. Exempel på användning av hook_block_view-funktionen (Tomlinson & VanDyk 2010 s. 233)

3.5 Moduler

Moduler är grundläggande byggstenar som utgör grunden för Drupal. Med hjälp av moduler utvidgas Drupal-kärnans funktionalitet. Moduler kan förklaras med hjälp av

legoblock, vilka går att kombinera enligt fördefinierade riktlinjer och kan tillsammans bilda komplexa lösningar. (Tomlinson & VanDyk 2010)

Allmänt sett finns det två olika typer av moduler, moduler som tillhör kärnan och kontribuerade moduler som är utvecklade av Drupal-gemenskapen. Moduler som tillhör kärnan är de som kommer med standardinstallationen av Drupal som t.ex. enkäter, menyer och taxonomier. Kontribuerade moduler är de tusentals moduler som går att ladda ner på internet (http://drupal.org/project/modules) och som är skapade för att utvidga och förbättra Drupals standardfunktionalitet. Ett exempel på en simpel moduls funktionalitet kunde vara att visa aktuella datumet och tiden medan ett komplext exempel kunde vara ett skyltfönster för en e-handel. (Tomlinson & VanDyk 2010)

Vid byggandet av en Drupal webbplats är det mycket sannolikt att en kontribuerad modul med den funktionalitet man är ute efter existerar. Planeringen av en Drupal webbplats är mycket viktig eftersom det är viktigt att välja vilka moduler som används på basis av de funktionalitetskrav som ställts på webbplatsen som utvecklas. Det är även viktigt att endast de moduler som verkligen behövs är installerade, eftersom varje modul drar ner på webbplatsens prestanda. Förutom att prestandan dras ner blir också webbplatsen mer komplex, vilket gör den svårare att utveckla och upprätthålla. (Nordin, Hakimzadeh & Melançon 2011)

Kontribuerade moduler placeras i katalogen sites/all/modules/contrib. Allting som läggs till en Drupal-installation bör läggas i sites katalogen. Med detta separeras kärnan från allt annat och gör den lättare att uppdatera. Att modulen placeras i katalogen all betyder att den är användbar av alla webbplatser definierade i samma Drupal-installation. Katalogen contrib existerar inte förrän den skapas, iden med den är att separera kontribuerade moduler från eventuella egna anpassade moduler som placeras i sites/all/modules/custom som också bör skapas. (Nordin, Hakimzadeh & Melançon 2011)

3.5.1 Struktur

Simplaste formen av en modul består av två filer i en katalog. Den ena filen med .info-ändelsen identifierar modulen medan den andra filen med .module-ändelsen innehåller

programkoden. En modul ges ett människoläsbart- och ett maskinnamn som används vid namngivning av modulens katalog och modulfilerna innanför den. Maskinnamnet är modulnamnet skrivet med små bokstäver utan mellanrum och specialtecken, dvs. om t.ex. modulens namn är X-ray blir maskinnamnet xray. Filen som identifierar modulen skulle i så fall heta xray.info och filen med programkoden xray.module. (Melançon 2011)

Infofilen, filen med .info-ändelsen, berättar för Drupal om att modulen existerar och anger t.ex. modulens namn. Då en modul inte är aktiverad läser Drupal endast modulens infofil för att visa informationen på modulkonfigurationssidan där modulen kan aktiveras. För att en modul skall fungera måste namn och kompatibla Drupal-versionen vara definierade i infofilen, eftersom modulen inte kan aktiveras utan dessa. En beskrivning av modulen ses också som ett minimum krav fast det inte tekniskt påverkar modulens funktion (se exempel i figur 9 nedan). I infofilen kan beroenden av andra moduler anges liksom också paketet modulen tillhör om den tillhör en modulgrupp. (Melançon 2011)

```
name = X-ray
description = Shows internal structures and connections of the web site
core = 7.x
```

Figur 9. Filen xray.info, exempel på en infofil med minimum definitioner (Melançon 2011 s. 384)

Modulfilen (.module-filen) berättar vad modulen skall göra. En modulfil börjas alltid med <?php öppningstaggen precis som vilken som helst PHP-fil, varefter kommentarer och logik tilläggs efter behov. I figur 10 nedan demonstreras hur en modul med maskinnamnet xray lägger till varje formulärs id ovanför själva formuläret med hjälp av funktionen hook_form_alter(). (Melançon 2011)

```
<?php
/**
 * @file
 * Helps site builders and module developers investigate a site
 */

/**
 * Implements hook_form_alter() to show each form's identifier
 */
function xray_form_alter(&$form, &$form_state, $form_id){
  $form['xray_display_form_id'] = array(
    '#type' => 'item',
    '#title' => t('Form ID'),
    '#markup' => $form_id,
    '#weight' => -100,
  );
}
```

Figur 10. Filen xray.module, exempel på en modulfil (Melançon 2011 s. 384)

3.5.2 Krokar och Hjälpfunktioner

I utveckling av moduler är det meningen att använda sig av verktyg tillgängliggjorda av Drupal. Dessa verktyg kallas API:n. Drupals API:n består av stödjande hjälpfunktioner och krokar (eng. hooks) som gör det möjligt för moduler att observera och ingripa olika händelser i systemet. (Melançon 2011)

En händelse kan ingripas med en krok om en sådan är definierad för händelsen. Drupals kärna innehåller 251 krokar vilka låter vilken som helst modul observera eller ingripa olika händelser utlösta av kärnan. Krokar kan skapas förutom av kärnan också av alla andra moduler med hjälp av någon variant av funktionerna avsedda för skapandet av krokar. Då en krok beskrivs används ordet hook i början av krokens namn som t.ex. hook_form_alter, ordet hook ersätts med maskinnamnet på modulen där kroken används (figur 10). (Melançon 2011)

3.5.3 Moduler och teman

Moduler och teman utgör en flexibel helhet, en välskriven modul låter teman ersätta element den producerar. Detta nås genom att alltid använda Drupals theme-funktion då modulen skall producera utskrift, som jag redan tog upp i kapitel 3.3.3 om temafunkt-

ioner. I modulen definieras en funktion med standard utskrift som sedan kan skrivas över av teman enligt design och behov. Moduler kan också innehålla egna CSS-filer för att ge deras utskrift ett standardutseende. CSS-filer läggs till moduler genom att definiera dem i modulens info-fil (se figur 11). (Melançon 2011)

```
name = X-ray technical site map
description = Shows internal structures and connections of the web site
package = Development
core = 7.x
stylesheets[all][] = xray.css
```

Figur 11. Infofil med direktiv för en CSS-fil (Melançon 2011)

Drupal levereras med jQuery och jQuery UI, vilka möjliggör avancerade effekter och element i användargränssnittet. Anpassad JavaScript/jQuery kan läggas till en webbsida på flera sätt i moduler och teman. Ett sätt är att lägga till dem på motsvarande sätt som CSS-filer läggs till genom definiering i infofilen (se figur 12). Om filen inkluderas på det här sättet kommer filen att laddas på alla sidladdningar. (Strawn & Gaskin 2011)

```
name = DGD7 Test Module
description = An example module
core = 7.x
files[] = dgd7_test.module
scripts[] = dgd7_test.js
```

Figur 12. Infofil med direktiv för en JavaScript-fil (Strawn, Gaskin 2011)

Andra alternativet för att ladda JavaScript-filer i en modul är att använda sig av funktionen drupal_add_js (se figur 13). Genom att använda alternativa metoden går det att bestämma noggrannare när filen laddas.

```
drupal_add_js(drupal_get_path('module', 'example') . '/example.js');
```

Figur 13. Inkludering av en JavaScript-fil med drupal_add_js-funktionen (Strawn, Gaskin 2011)

Det traditionella sättet att koppla beteenden till HTML-element med JavaScript och jQuery är att använda sig av jQuerys $(document).ready-funktion (se figur 14). Med användning av funktionen försäkrar man sig om att hela HTML-dokumentet är laddat före den egna kodens funktioner körs. (Strawn & Gaskin 2011)

```
$(document).ready(function(){
  // act on all h3 elements and give them a custom class
  $('h3').addClass('custom-css-class');
});
```

Figur 14. Exempel på användning av $(document).ready(); funktionen (Strawn, Gaskin 2011)

I Drupal används Drupals beteenden (eng. behaviors) istället för att se till att koden körs vid rätt tillfälle (se figur 15). Dessutom har användningen av beteenden den fördelen att om dokumentet manipuleras med AJAX kan alla beteenden kopplas till nya element med att kalla på Drupal.attachBehaviors-funktionen. (Strawn & Gaskin 2011)

```
Drupal.behaviors.myModuleHeaders = {
  attach: function(context, settings){
    // act on all h3 elements and give them a custom class
    $('h3').addClass('custom-css-class');
  }
}
```

Figur 15. Exempel på Drupals beteenden (eng. behaviors) (Strawn, Gaskin 2011)

3.5.4 Databaser

Drupal är beroende av en databas och nästan allting, som innehåll och kommentarer, sparas i en databas. Drupal har ett inbyggt abstraktionslager för databasen som sköter om skillnader i olika databaser. Drupals databasabstraktionslager är baserat på PDO (PHP Data Object) och har två syften, ena är att hålla koden oberoende av den använda databasen och det andra är att rensa av användaren inmatad data för att förhindra SQL-injektion attacker (eng. SQL injection attacks). Så länge som de använda SQL-förfrågningarna följer ANSI (American National Standards Institute) standarderna behöver de inte skrivas om för olika databaser. (Tomlinson & VanDyk 2010)

Förutom abstraktionslagret krävs också drivrutiner (eng. drivers) för att Drupal skall fungera ihop med en databas. Drupals kärna har stöd för MariaDB/MySQL, PostgreSQL och SQLite. (Melançon 2011)

SELECT förfrågningar görs till databasen med hjälp av db_query-funktionen (se figur 16). I funktionen används Drupal specifik syntax, tabellnamnet är innanför klammerparenteser och jämförelsens senare del i WHERE-satsen är ersatt med en

platshållare. Platshållaren ersätts med värden definierade i tabellen i funktionens andra del efter att de har blivit validerade av abstraktionslagret. Användning av klammerparenteserna gör att Drupal automatiskt lägger till rätt prefix för tabeller. Förfrågningen i figur 16 returnerar alla värden i namn kolumnen från tabellen role som har värdet 2 i kolumnen rid. (Tomlinson & VanDyk 2010)

```
$result = db_query('SELECT name FROM {role} WHERE rid = :rid', array(':rid' => 2));
```

Figur 16. Exempel på db_query-funktionen (Tomlinson & VanDyk 2010)

För att lägga till en rad i en tabell används db_insert-funktionen (se figur 17). Liknande funktioner finns tillgängliga för att uppdatera (db_update) och ta bort (db_delete) rader från tabeller. (Tomlinson & VanDyk 2010)

```
$nid = db_insert('joke')
  ->fields(array(
    'nid' => '4',
    'vid' => 1,
    'punchline' => 'And the pig said oink!',
  ))
  ->execute();
```

Figur 17. Exempel på db_insert-funktionen (Tomlinson & VanDyk 2010)

3.5.5 Installationsfilen

En modul kan ha en installationsfil (eng. .install file) som definierar vad som skall ske när modulen installeras. En databastabell kan t.ex. skapas med hjälp av hook_schema-kroken (se figur 18) och ett meddelande som visas efter installationen kan skapas med drupal_set_message-funktionen. En databastabell definierad med hjälp av hook_schema-kroken i installationsfilen skapas och tas bort automatiskt vid installation respektive avinstallation av modulen. (Melançon 2011)

```
/**
 * Implements hook_schema().
 */
function book_schema() {
  $schema['book'] = array(
    'description' => 'Stores book outline information. Uniquely connects each node in
the outline to a link in {menu_links}',
    'fields' => array(
      'mlid' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'default' => 0,
        'description' => "The book page's {menu_links}.mlid.",
      ),
      'nid' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'default' => 0,
        'description' => "The book page's {node}.nid.",
      ),
      'bid' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'default' => 0,
        'description' => "The book ID is the {book}.nid of the top-level page.",
      ),
    ),
    'primary key' => array('mlid'),
    'unique keys' => array(
      'nid' => array('nid'),
    ),
    'indexes' => array(
      'bid' => array('bid'),
    ),
  );
  return $schema;
}
```

Figur 18. Exempel på hook_schema-kroken (Tomlinson & VanDyk 2010)

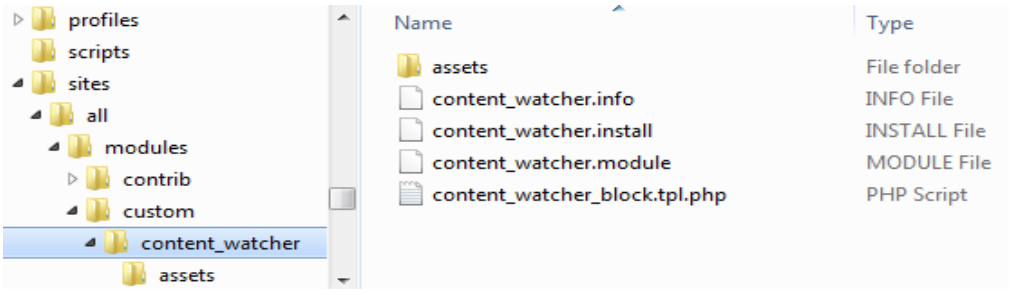
4 INNEHÅLLSÄNDRINGSVAKTEN

Innehållsändringsvakten (eng. Content watcher) är en anpassad modul som är skapad för att demonstrera användningen av Drupals krokar och hjälpfunktioner, av vilka en stor del användes i modulutveckling för fastighetsförmedlingswebbplatsen. I det här kapitlet förklaras modulens funktionalitet med hjälp av skärmkapningar och exempelkod. Kodexemplen innehåller sällan fullständiga filer eller funktionsdefinitioner, vilka kan hittas som bilagor. För exemplen visade i dessa kapitel har Drupals standard installation och teman använts.

Innehållsändringsvakten förser en Drupal-webbsida med ett block genom vilken användare kan beställa e-postmeddelanden gällande innehållsuppdateringar. Administratorn kan via administrationsgränssnittet bestämma i vilken region blocket skall visas och definiera för vilka innehållstyper vakten skall vara aktiverad.

4.1 Struktur och installation

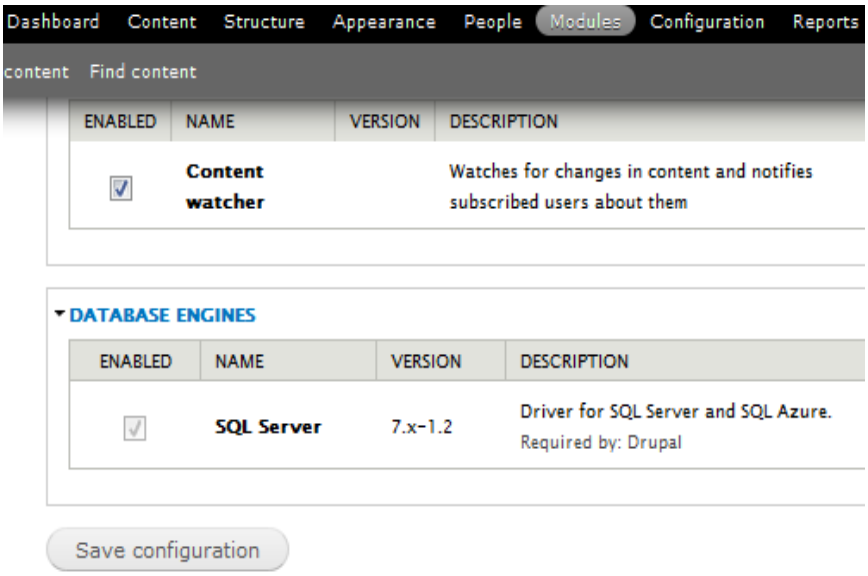
Innehållshanteringsvakten består av huvudkatalogen, en infofil, en installationsfil, en modulfil, en mallfil och en katalog som innehåller en CSS- och JavaScript-fil (se figur 19). Efter att huvudkatalogen med innehåll är placerad i sites/all/modules/custom katalogen kan modulen aktiveras, d.v.s. installeras, på modulkonfigurationssidan i Drupals webbadministrationsgränssnitt.



Figur 19. Filstrukturen på innehållsändringsvakten.

Moduler aktiveras och avaktiveras på modulkonfigurationsingssidan i webbadministrationsgränssnittet där alla valbara moduler listas (se figur 20). I listan

presenteras den information som modulerna anger i sin infofil. Då en modul aktiveras körs hook_install-kroken av systemet, innehållsändringsvakten skriver ut ett meddelande om att modulen har installerats. Kroken hook_uninstall körs inte vid avaktivering av modulen utan först vid avinstallering som sker via fliken uninstall på modul konfigureringssidan. Innehållsändringsvaktens hook_uninstall-krok tar hand om att radera variabler som den skapat och meddelar till slut om att modulen har avinstallerats. Både installations- och avinstallationskroken definieras i installationsfilen.



Figur 20. Drupals modul konfigureringssida.

4.2 Databasen

Innehållsändringsvakten kräver tre databastabeller för att funktionera. I den första tabellen sparas användare, i den andra länkas användare med innehållsnoder, medan det i den tredje sparas de innehållsnoder som har ändrats, och ett värde som anger om ett meddelande gällande den innehållsnoden skall skickas eller inte. Databasen skapas med hook_schema-kroken, och figur 21 visar hur den tredje tabellen content_watcher_mail skapas.

Drupal skapar databastabellerna definierade i hook_schema-kroken då modulen aktiveras. Databastabellerna definieras i en matris där tabellnamnen fungerar som index. Tabellens kolumner anges ytterligare som matriser där egenskaper som kolumnnamn, beskrivning och typ anges (se figur 21). De två andra tabellerna för användare och länkning av användare och innehåll definieras på motsvarande sätt. Vid avinstallering av modulen sköter Drupal om att databastabeller definierade i hook_schema-kroken med data raderas.

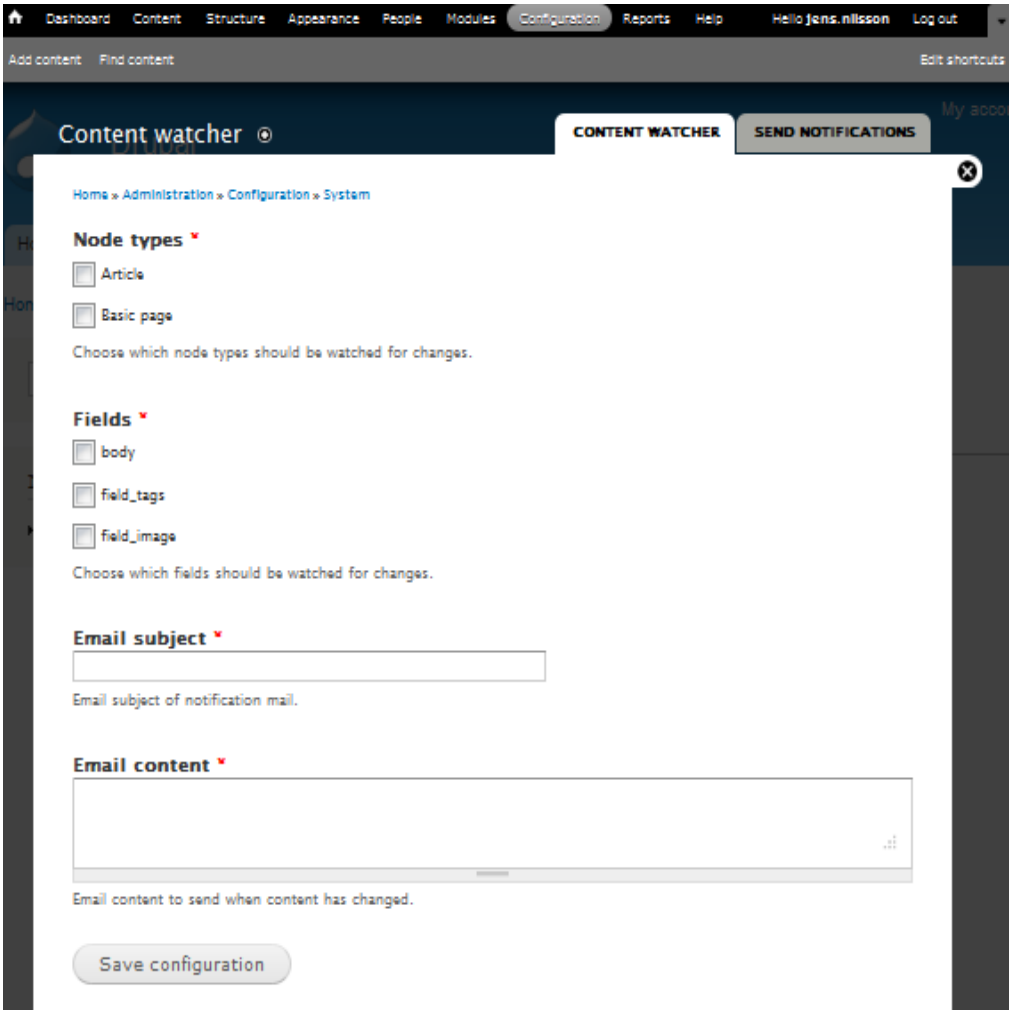
```
function content_watcher_schema() {
  $schema['content_watcher_mail'] = array(
    'description' => 'Content watcher mailed sent',
    'fields' => array(
      'nid' => array(
        'description' => 'The primary identifier for a node.',
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
      ),
      'mail' => array(
        'description' => 'Identifies if mail should be sent',
        'type' => 'int',
        'size' => 'small',
        'unsigned' => TRUE,
        'default' => 0,
      ),
    ),
  );
  return $schema;
}
```

Figur 21. Skapandet av databastabellen content_watcher_mail med hook_schema_kroken.

4.3 Konfigurationssidan

För att modulen skall kunna konfigureras genom Drupals webbadministrationsgränssnitt används, istället för att göra ändringar direkt i koden, Drupals API för att skapa en konfigurationssida (se figur 22). Innehållsändringsvaktens konfigurationssida består av en flik med basinställningar och en annan med val att manuellt skicka innehållsändrings-meddelanden. I basinställningarna väljs vilka nodtyper och fält som

skall vaktas för ändringar. Även e-postens rubrik och innehåll definieras i basinställningarna.



Figur 22. Innehållsändringsvaktens konfigurationssida i Drupals webbadministrationsgränssnitt.

4.3.1 Definiering av konfigureringssidan

I Drupal kan sidor skapas med hjälp av hook_menu-kroken. I kroken definieras sidans rubrik, webbadress och åtkomst, samt vilken funktion som returnerar sidans innehåll (se figur 23).

Hook_menu-kroken returnerar en matris med menyalternativ som är indexerade med alternativens webbadresser. Menyalternativens egenskaper definieras också med hjälp av en matris, där indexen är egenskapernas namn.

```
function content_watcher_menu(){
  $items = array();
  $items['admin/config/system/content-watcher'] = array(
    'title' => t('Content watcher'),
    'description' => t('Watches for content changes and notifies about them.'),
    'page callback' => 'drupal_get_form',
    'page arguments' => array('content_watcher_settings'),
    'access arguments' => array('edit content watcher settings'),
  );
  ... return $items;
}
```

Figur 23. Definiering av en ny sida till webbadministrationsgränssnittet i hook_menu-kroken.

Sidans namn ges i indexet title och beskrivning i indexet description, vilka i sig är självförklarande. I indexet ’access arguments’ anges vilka rättigheter en användare skall ha för att få tillgång till sidan. Nya rättigheter går att skapa i modulen med hook_permission-kroken (se figur 24), men även existerande rättigheter kan användas. Rättigheterna delas ut till roller, och för att ha tillgång till sidan krävs att en användare har en roll som innebär den specifika rättigheten i fråga.

```
function content_watcher_permission(){
  return array(
    'edit content watcher settings' => array(
      'title' => t('Edit content watcher settings'),
      'description' => t('Edit content watcher settings'),
    ),
  );
}
```

Figur 24. Definiering av en ny rättighet med hook_permission-kroken.

Funktionen som returnerar utskriften för sidan anges med indexet ’page callback’. I det här faller används funktionen drupal_get_form som genererar och returnerar ett formulär. Formuläret genereras från en matris som skapas i funktionen content_watcher_settings, som är definierad som ett argument för sidan av indexet page arguments (se figur 24).

4.3.2 Konfigureringsformuläret

Konfigureringsformuläret skapas med hjälp av drupal_get_form-funktionen som är en del av Drupals Form API. Funktionen genererar formuläret av det emottagna

argumentet som består av en matris med formelement. Dessa formelement är alltså givna som matriser, och de innehåller bl.a. formelementets typ, namn och beskrivning (se figur 25).

```
function content_watcher_settings(){
  // Generate array with node types
  $types = node_type_get_types();
  $type_options = array();
  foreach($types as $key=>$type){
    $type_options[$key] = $type->name;
  }
  ... $form = array();
  // Form element with node types as checkboxes
  $form['content_watcher_node_types'] = array(
    '#type' => 'checkboxes',
    '#title' => t('Node types'),
    '#description' => t('Choose which node types should be watched for changes.'),
    '#default_value' => variable_get('content_watcher_node_types', array('')),
    '#options' => $type_options,
    '#required' => TRUE,
  );
  ... return system_settings_form($form);
}
```

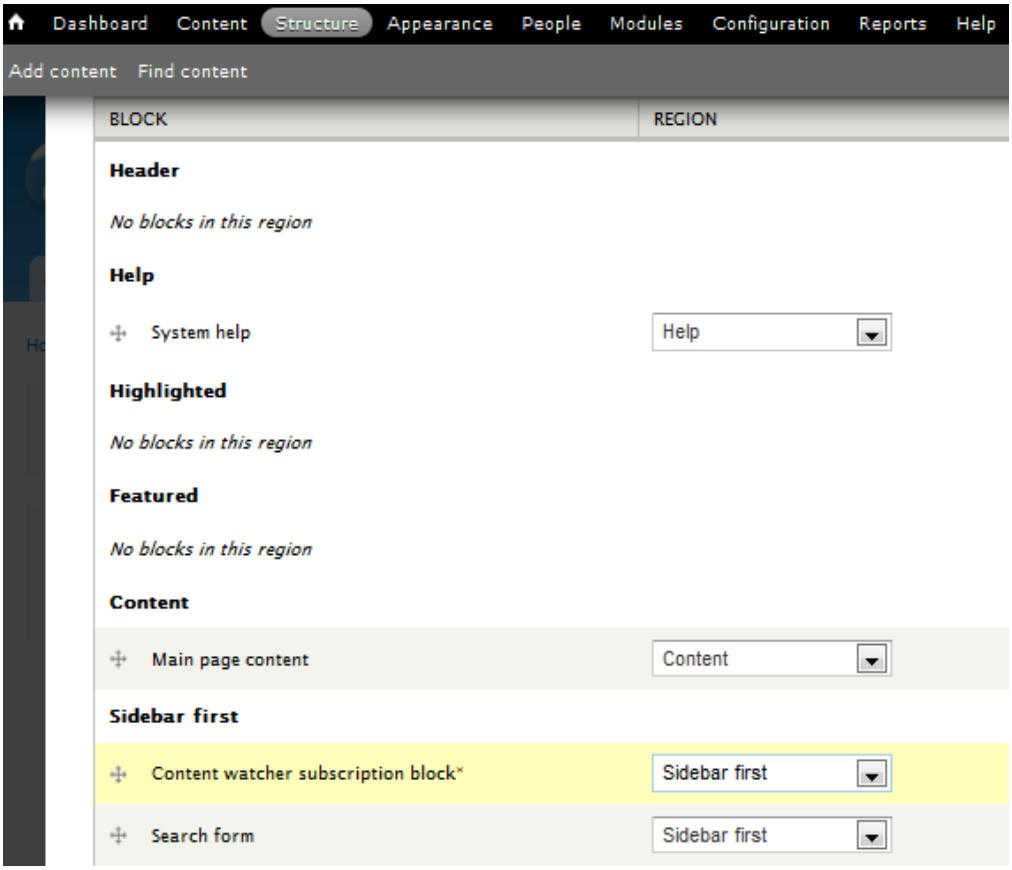
Figur 25. Utdrag från content_watcher_settings-funktionen.

Figur 25 visar all den kod som behövs för att definiera checkboxarna där nodtyperna väljs överst i formuläret i figur 22. Definierade nodtyperna söks med node_type_get_types()-hjälpfunktionen och ges som alternativ (’#options’) i formelementets matris. Formelementet görs nödvändig genom att sätta värdet TRUE för indexet #required. Aktuella eller standard värden av formelementet ges i indexet #default_value. I det här fallet är värdet lagrat i Drupals inbyggda variabelsystem och hämtas med variable_get-funktionen. Den första parametern som ges, d.v.s. ’content_watcher_node_types’, är den efterfrågade variabelns namn, medan den andra parametern är värdet som skall sparas om variabeln inte har tilldelats ett värde tidigare.

4.4 Prenumerationsformuläret som ett block

Innehållsändringsvaktens prenumerationsformulär förses som ett block som kan placeras i en valbar region via block-konfigurationssidan, där alla existerande block

visas. I den här demonstrationen har blocket placerats i Sidebar first regionen (se figur 26 nedan).



Figur 26. Block konfigurationssidan.

Blocket definieras med hook_block_info-kroken där beskrivningen som syns på block konfigureringssidan anges. Blockets innehåll genereras med hjälp av hook_block_view-kroken (se figur 27).

```
function content_watcher_block_view($block_name = ''){
  $block = array();
  if($block_name == 'content_watcher_box' && arg(0) == 'node' && is_numeric(arg(1))){
    $node = node_load(arg(1));
    $types = variable_get('content_watcher_node_types', array(''));
    if($types[$node->type] === $node->type){
      $form = drupal_get_form('content_watcher_notify_form');
      $block['subject'] = t('Get content updates');
      $block['content'] = theme('content_watcher_block', array('form' => $form));
    }
  }
  return $block;
}
```

Figur 27. Användning av hook_block_view-kroken i Content watcher-modulen.

I figur 27 ovan visas hur blockets innehåll genereras. Först kontrolleras det genom argumenten att den nuvarande sidan är av typen innehållsnod. Detta görs eftersom blocket endast skall visas för innehållsnoder. Sedan söks de på innehållsändringsvaktens konfigurationssida definierade nodtyperna och jämförs med den aktuella sidans nodtyp. Om dessa är lika så genereras prenumerationsformuläret.

Formuläret definieras i content_watcher_notify_form-funktionen och generas med hjälp av drupal_get_form-funktionen. Det genererade formuläret skickas som en parameter till content_watcher_block mallfilen som tar hand om utskriften av blocket. Det utskrivna blocket syns i figur 28 med rubriken Get content updates.

Figur 28. Innehållsändringsvaktens prenumerationsformulär placerat i Sidebar first-regionen.

4.4.1 Skickandet av prenumerationsformuläret

Prenumerationsformuläret skickas med AJAX, vilket innebär att ingen sidladdning görs utan meddelanden genereras dynamiskt i blocket. AJAX-funktionaliteten är inbyggd i Form API och triggas genom en enkel definition i content_watcher_notify_form-funktionen (se figur 29). Tack vare den inbyggda funktionen finns det även stöd för fall där JavaScript saknas från bläddraren.

```
function content_watcher_notify_form($form, &$form_state){
...
  $form['box']['send'] = array(
    '#type' => 'submit',
    '#value' => t('Submit'),
    '#ajax' => array(
      'callback' => 'content_watcher_notify_form_submit_callback',
      'wrapper' => 'content-watcher-notify-form',
      'effect' => 'fade',
    ),
  );
  return $form;
}
```

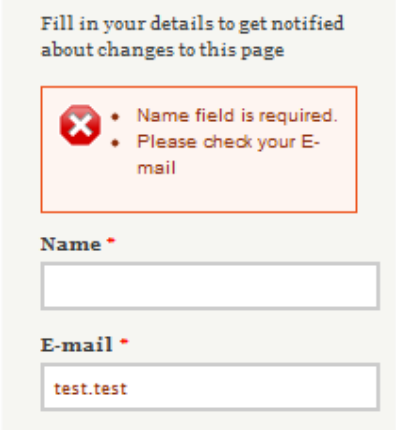
Figur 29. Definition av submit-knappen med AJAX-funktionalitet i content_watcher_notify_form-funktionen.

Då formuläret skickas iväg skickas den aktuella innehållsnodens id, det ifyllda namnet samt e-postadressen till content_watcher_notify_form_validate-funktionen, som bl.a. kollar att e-postadressen är av rätt typ (se figur 30).

```
function content_watcher_notify_form_validate($form, &$form_state){
...
  $email = $form_state['values']['email'];
  if(!valid_email_address($email) || $email == ''){
    form_set_error('email', t('Please check your E-mail'));
  }
...
}
```

Figur 30. E-postadress validering i content_watcher_notify_form_validate-funktionen.

Om de värden som skickats går igenom valideringen körs content_watcher_notify_form_submit-funktionen. Om de inte gått igenom returneras ett felmeddelande för användaren (se figur 31).



Figur 31. Felmeddelande då prenumerationsformuläret inte går igenom valideringen.

Om formuläret går igenom valideringen sparar content_watcher_notify_form_submit-funktionen information gällande användare samt kopplingen mellan användare och innehållsnoden i databasen.

4.5 Notering av ändringar

För att bevaka ändringar i innehållsnoder används krokarna hook_node_presave och hook_node_update. Med en kombination av dessa krokar kan den ursprungliga och uppdaterade nodens fält jämföras. Kroken hook_node_presave körs av Drupal då en ny nod har blivit validerad men inte ännu sparad i databasen. Vid det här skedet laddar innehållsändringsvakten den ursprungliga noden från databasen med node_load-funktionen och sparar den i en global variabel för senare användning (se figur 32).

```
function content_watcher_node_presave($node){
  global $content_watcher_original_node;
  $content_watcher_original_node = node_load($node->nid);
}
```

Figur 32. Sparandet av ursprungliga noden i en global variabel i content_watcher_node_presave-kroken.

Kroken hook_node_update (se figur 33) körs efter att den uppdaterade noden har sparats i databasen, vid det här skedet laddas den uppdaterade noden från databasen med node_load-funktionen för jämföring med den ursprungliga noden. Den här gången är det viktigt att förse funktionsanropet med den tredje parametern TRUE för att försäkra sig

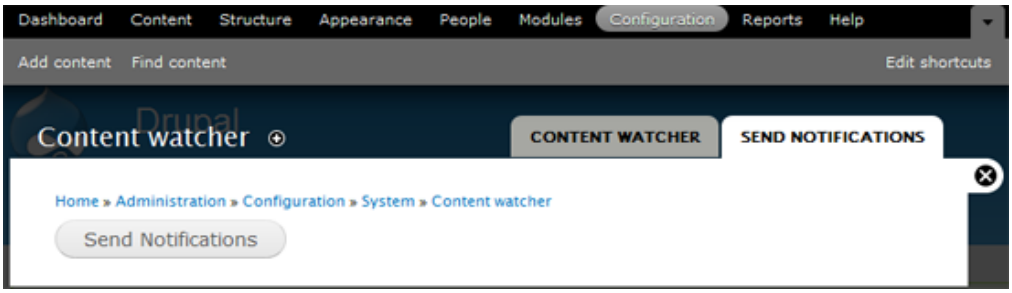
om att noden verkligen laddas och inte returneras från mellanminnet. När den uppdaterade noden har hämtats kan den jämföras med den ursprungliga, som tidigare sparats i watcher_node_presave-kroken. Endast de fält som definierats på innehållsändringsvaktens konfigurationssida jämförs mellan noderna. Om något av fälten har ändrats och det finns användare som har beställt uppdateringar gällande den editerade innehållsnoden, sparas information om att noden ändrats i content_watcher_mail databastabellen.

```
function content_watcher_node_update($node) {
  global $content_watcher_original_node;
  $n = node_load($node->nid, NULL, TRUE);
  $defined_content_types = variable_get('content_watcher_node_types', array(''));
  $defined_fields = variable_get('content_watcher_fields', array(''));
  if(in_array($node->type, $defined_content_types)){
    $changed = FALSE;
    // Loop through fields
    foreach($defined_fields as $field){
      // Check if field exists and has changed
      if(isset($content_watcher_original_node->$field) && isset($n->$field) && $con-
tent_watcher_original_node->$field != $n->$field){
        $changed = TRUE;
      }
    }
    // Check if node is watched by anyone
    $count = db_query("SELECT COUNT(cws.nid) FROM {content_watcher_subscriptions} cws
WHERE cws.nid = :cws", array(':cws' => $node->nid))->fetchField();
    if($changed && $count > 0){
      db_merge('content_watcher_mail')
        ->key(array('nid' => $node->nid))
        ->fields(array(
          'mail' => 1,
        ))
        ->execute();
    }
  }
}
```

Figur 33. Jämförande av ursprungliga och uppdaterade noden i content_watcher_node_update-kroken.

4.6 E-post gällande uppdateringar

Som tidigare nämnts finns det en flik på innehållsändringsvaktens konfigurationssida där e-post meddelanden gällande innehållsuppdateringar kan skickas ut. Flikens innehåll består enbart av en knapp med texten Send Notifications (se figur 34).



Figur 34. Fliken för att skicka e-post gällande innehållsuppdateringar på innehållsändringsvaktens konfigurationssida.

Då man trycker på knappen med texten Send Notifications (i figur 34 ovan) körs funktionen content_watcher_send_notifications_form_submit. Funktionen gör då en databasför-frågning där alla e-postadresser och länkade innehållsnoder som har uppdaterats hämtas.

Skickandet av e-postmeddelanden delas i batch-operationer för att undvika timeouts om det är frågan om stora mängder e-postadresser. Genom en progressbar hålls användaren även jämnt uppdaterad av hur långt processen är.

E-postmeddelanden skickas slutligen med API-funktionen drupal_mail (se figur 35).

```
function content_watcher_send_notification($params) {
  drupal_mail('content_watcher', 'notification', $params['mail'], 'en', $params);
}
```

Figur 35. Kallandet på drupal_mail-funktionen i batch-operationen content_watcher_send_notification.

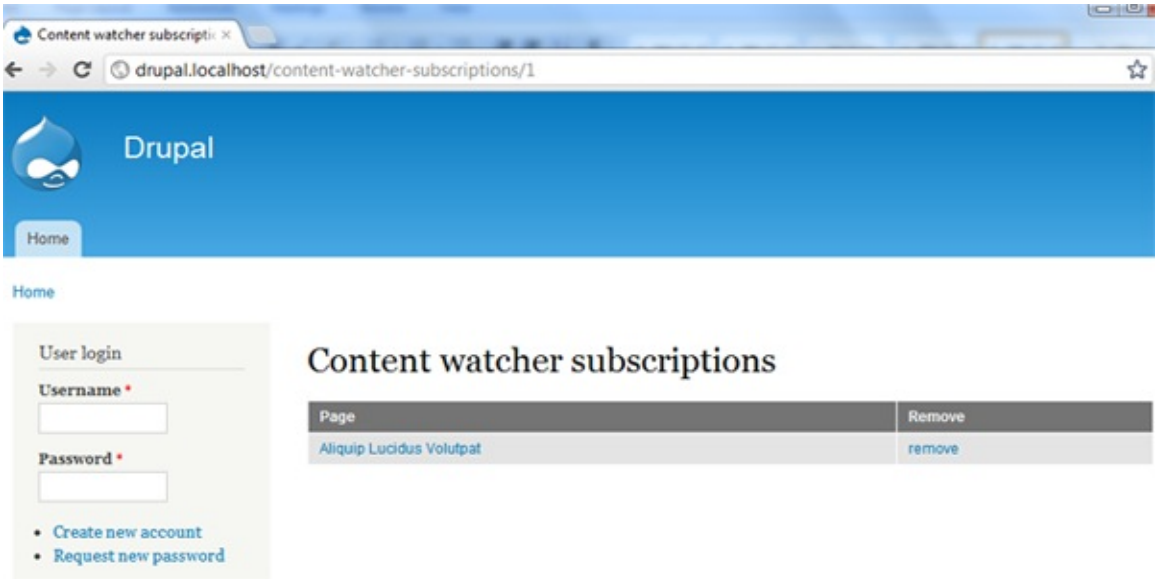
Formulerandet av e-postmeddelandets innehåll sker i hook_mail-kroken (se figur 36). I kroken hämtas rubriken och meddelandet som ställts in på innehållsändringsvaktens konfigurationssida. I meddelandet tilläggs dessutom länkar till ändrat innehåll, samt en länk till en sida där en användare kan avboka information om innehållsuppdateringar.

```
function content_watcher_mail($key, &$message, $params) {
  switch($key) {
    case 'notification':
      $message['subject'] = t(variable_get('content_watcher_subject'));
...
}
```

Figur 36. Extrakt från content_watcher_mail-kroken.

4.7 Visa och avboka information om innehållsuppdateringar

E-postmeddelandet som skickas gällande uppdateringarna innebär en länk, där man kommer åt att se vilka innehållsnoder man prenumererat uppdateringar på. Länken leder till en sida som ser ut som sidan i figur 37 nedan.



Figur 37. En sida med enanvändare innehållsuppdaterings prenumerationer.

Sidan med prenumereringslistningen definieras i content_watcher_menu-kroken (se figur 38). Från definitionen kan förutom sidans titel och beskrivning också läsas att besökaren måste ha åtkomst till innehåll (eng. access content) på sidan för att kunna se sina prenumereringar. I definitionen framgår även sidans adress, samt att den tar emot ett argument som sista parameter i URL:n, vilket anges med procenttecknet. Sidans innehåll returneras av funktionen content_watcher_list_subscriptions (se figur 39), som tar emot argumentet plockat ur sista segmentet av URL:n.

```
function content_watcher_menu(){
...
  $items['content-watcher-subscriptions/%'] = array(
    'title' => t('Content watcher subscriptions'),
    'description' => t('Content watcher subscriptions'),
    'page callback' => 'content_watcher_list_subscriptions',
    'page arguments' => array(1),
    'access arguments' => array('access content'),
    'type' => MENU_NORMAL_ITEM,
  );
...
}
```

Figur 38. Definition av prenumereringslistningssidan i content_watcher_menu-kroken.

I figur 39 nedan visas content_watcher-funktionen som returnerar innehållet till prenumereringslistningssidan beroende på parametern som den får från URL:n. Parametern är innehållsändringsvaktens användarens id som är kopplat ihop med e-postadressen som användaren matat i innehållsändringsvaktens block för att prenumerera på ändringar. Funktionen kollar att den inmatade parametern är ett siffervärde, om det är frågan om ett siffervärde anropas _content_watcher_get_subscriptions-funktionen som returnerar alla prenumereringar för en specifik användare från databasen. Till slut formas utskriften med hjälp av Drupals theme-funktionen.

```
function content_watcher_list_subscriptions($s){
  $output = '';
  if(is_numeric($s)){
    $subscriptions = _content_watcher_get_subscriptions($s);
    if(!$subscriptions){
      $output = t('No subscriptions.');
    }
    else{
      $output .= theme('table', $subscriptions, array('id' => 'content-watcher-
subscriptions-list'));
    }
  }
  return $output;
}
```

Figur 39. Funkitonen som returnerar innehållet till prenumereringslistningssidan.

4.8 Periodisk kontroll med cron

I Drupal körs periodiska uppgifter genom cron som är en inbyggd funktionalitet för att sköta om dessa uppgifter i fråga. Cron kan köras både manuellt och ställas in att köras på olika tidsintervall via Drupals webbadministrationsgränssnitt.

Idén med innehållsändringsvakten är att den automatiskt skall informera om ändringar i innehållet till prenumeranter. Detta skulle kunna ske omedelbart vid sparning av ändringar men innehållsändringsvakten är designad så att den skickar meddelanden vid cron-körningar. Drupals API utnyttjas för att i modulen komma åt cron-händelsen genom krokar.

För att använda sig av cron måste två krokar användas; den ena är hook_cron_queue_info var en cron-kö definieras, medan den andra är hook_cron som körs i samband med varje cron-körning. I figur 40 nedan är hook_cron_queue_info definierad. I definitionen framgår cron-köns namn, vilken funktion som skall processera kön och hur lång tid denna kö har för att processeras per cron-körning. Om hela kön inte töms vid den aktuella cron-körningen fortsätter processeringen av kön vid nästa körning. Själva kön byggs upp i hook_cron-kroken.

```
function content_watcher_cron_queue_info() {
  $queues['content_watcher'] = array(
    'worker callback' => 'content_watcher_send_notification', // This is the callback
function for each queue item.
    'time' => 15, // This is the max run time per cron run in seconds.
  );
  return $queues;
}
```

Figur 40. Användning av hook_cron_queue_info-kroken i innehållsändringsvakten.

I figur 41 nedan används hook_cron-kroken för att göra en databassökning som returnerar alla e-postadresser och tillhörande innehåll som prenumererats. Varje returnerad e-postadress med tillhörande innehåll tilläggs sedan i cron-kön för processering vid en cron-körning. Efter att kön bildats nollställs värden i databasen som berättar att innehållsändringen har skickats.

```
function content_watcher_cron() {
  $queue = DrupalQueue::get('content_watcher');
  $result = db_query('SELECT email, cwu.id as uid, GROUP_CONCAT(cws.nid SEPARATOR \',\')
as nids FROM {content_watcher_users} cwu
    JOIN {content_watcher_subscriptions} cws ON cws.uid = cwu.id
    JOIN {content_watcher_mail} cwm ON cwm.nid = cws.nid
    WHERE cws.deleted = 0 AND cwm.mail > 0 GROUP BY cwu.id
  ');
  $ids = array();
  foreach($result as $record){
    $params = array('mail' => $record->email, 'uid' => $record->uid, 'ids' => ex-
plode(',', $record->nids));
    $queue->createItem($params);
    $ids = array_merge($ids, explode(',', $record->nids));
  }
  $num_updated = db_update('content_watcher_mail')
    ->fields(array(
      'mail' => 0,
    ))
    ->condition('nid', $ids, 'IN')
    ->execute();
}
```

Figur 41. Användning av hook_cron kroken i innehållsändringsvakten.

I content_watcher_cron_queue_info-kroken (se figur 40) definieras att elementen I cron-kön skall hanteras med hjälp av content_watcher_send_notification-funktionen, som är den samma som användes för att skicka e-postmeddelanden i kapitel 4.6. Funktionen tar emot en matris som parameter. Matrisen innehåller information om e-postadressen vart meddelandet skall skickas, samt det innehåll som skall visas i meddelandet.

5 AVSLUTNING

Webbinnehållshanteringssystemet Drupal gör det möjligt att skapa enkla webbplatser som t.ex. bloggar med mycket liten kunskap i webbprogrammering. Detta är möjligt genom den funktionalitet som Drupals kärna och tusentals kontribuerade moduler erbjuder. Tack vare sitt API är Drupal också ett webbapplikationsramverk som kan användas som grund för avancerade webbapplikationer och -platser. Genom att bygga anpassade moduler kan Drupals funktionalitet utvidgas. Anpassade modulerna har tillgång till systemets funktioner och kan växelverka med övriga delar av applikationen genom Drupals API. Exempelmodulen, Innehållsändringsvakten som presenteras i kapitel 4 är ett exempel på en anpassad modul som utvidgar Drupals basfunktionalitet. Innehållsändringsvakten vakten använder sig av Drupals API för att skicka e-post meddelanden gällande innehållsuppdateringar som sker i systemet.

Den funktionalitet och de möjligheter som Drupal erbjuder, och som detta arbete omfattar en stor del av, står som grund för den utvecklade webbplatsen. För webbplatsen skapades tio anpassade moduler samt två stycken teman, ett för publika vyn och ett annat för inloggade användare. Anpassade modulerna och temat är ansvariga för största delen av det synliga medan t.ex. fastigheter och lokaler sparas som innehållsnoder i databasen genom inbyggd funktionalitet i Drupal.

Slutprodukten blev en webbplats med fastigheter och lokaler som presenteras för användaren på en karta med hjälp av Google Maps eller alternativt i en listvy. Fastigheter och lokaler kan filtreras med olika kriterier, läggas till som favoriter och skickas till andra genom e-post. På webbplatsen finns också bloggartiklar, nyheter samt författarbeskrivningar tillgängliga för användaren. Webbplatsen är synlig i adressen http://www.toimitila.fi och en skärmkapning av webbplatsens framsida hittas som bilaga.

Det fördefinierade verktyget Drupal visade sig vara ett utmärkt val för detta projekt. Både dess egenskaper som webbinnehållshanteringssystem och som ett webbapplikationsramverk var till stor hjälp vid utvecklingen av webbplatsen. Förutom den funktionalitet som kom med Drupals kärna utnyttjades också några av de tusentals kontribuerade moduler utvecklade av Drupal-gemenskapen.

KÄLLOR

About Drupal | 2012, [www], Hämtat 19.03.2012
http://drupal.org/about

About nodes | 2011, [www], Hämtat 30.03.2012
http://drupal.org/documentation/modules/node

About Taxonomy | 2012, [www], Hämtat 31.03.2012
http://drupal.org/node/774892

An Introduction to Entities | 2011, [www], Hämtat 23.03.2012
http://drupal.org/node/1261744

Content management system DocForge | 2011, [www], Hämtat 27.02.2012
http://docforge.com/wiki/CMS

Drupal Wikipedia | 2012, [www], Hämtat 25.02.2012
http://en.wikipedia.org/wiki/Drupal

Framework DocForge | 2011, [www], Hämtat 23.02.2012
http://docforge.com/wiki/Framework

Heinisuo, Rami (2001), PHP ja MySQL, Talentum Media Oy

Luisi, Jacine (2011), The Definitive Guide to Drupal 7. APress

Melançon, Benjamin (2011), The Definitive Guide to Drupal 7. APress

MySQL Wikipedia | 2012, [www], Hämtat 25.02.2012
http://en.wikipedia.org/wiki/Mysql

Nordin, Dani, Hakimzadeh, Dan & Melançon, Benjamin (2011), The Definitive Guide to Drupal 7. APress

Php Wikipedia | 2012, [www], Hämtat 27.02.2012
http://en.wikipedia.org/wiki/Php

Programming language DocForge | 2011, [www], Hämtat 17.03.2012
http://docforge.com/wiki/Programming_language

Relational Model Wikipedia | 2012, [www], Hämtat 27.02.2012
http://en.wikipedia.org/wiki/Relational_model

Strawn, Jake & Gaskin, Dmitri (2011), The Definitive Guide to Drupal 7. APress

The Drupal overview | 2010, [www], Hämtat 27.02.2012
http://drupal.org/getting-started/before/overview

Todd Tomlinson, John K. VanDyk. 2010, Pro Drupal 7, Development: Third Edition. APress

User: access and management settings | 2011, [www], Hämtat 31.03.2012
http://drupal.org/documentation/modules/user

Web application framework DocForge | 2011, [www], Hämtat 23.02.2012
http://docforge.com/wiki/Web_application_framework

Web site DocForge | 2011, [www], Hämtat 23.02.2012
http://docforge.com/wiki/Web_site

Web site Encyclopædia Britannica Online | 2012, [www], Hämtat 22.02.2012
http://www.britannica.com/EBchecked/topic/690679/Web-site

Working with content types and fields (Drupal 7) | 2012, [www], Hämtat 30.03.2012
http://drupal.org/documentation/modules/field-ui

Working with Drupal 7 core Field module | 2011, [www], Hämtat 30.03.2012
http://drupal.org/documentation/modules/field

BILAGOR

BILAGA 1: content_watcher.info

```
name = Content watcher
description = Watches for changes in content and notifies subscribed users about them
package = "Custom"
core = 7.x

stylesheets[all][] = assets/content_watcher.css
scripts[] = assets/content_watcher.js
```

BILAGA 2: content_watcher.install

```
<?php
/**
 * @file
 * Install file for the Content watcher module.
 */

/**
 * Implementation of hook_install()
 */
function content_watcher_install() {
  drupal_set_message('Content watcher has been successfully installed!');
}

/**
 * Implementation of hook_uninstall().
 */
function content_watcher_uninstall(){
  variable_del('content_watcher_subject');
  variable_del('content_watcher_text');
  variable_del('content_watcher_node_types');
  variable_del('content_watcher_fields');

  drupal_set_message('Content watcher has been successfully uninstalled!');
}

/**
 * Implementation of hook_schema().
 */
function content_watcher_schema() {
  $schema['content_watcher_users'] = array(
    'description' => 'Users',
    'fields' => array(
      'id' => array(
        'description' => 'Id',
        'type' => 'serial',
        'unsigned' => TRUE,
        'not null' => TRUE,
      ),
      'name' => array(
        'description' => 'Name of person',
        'type' => 'varchar',
        'length' => 255,
        'not null' => TRUE,
        'default' => '',
      ),
      'email' => array(
        'description' => 'Content watcher subscribers email address',
        'type' => 'varchar',
```

```
      'length' => 255,
      'not null' => TRUE,
      'default' => '',
    ),
  ),
  'primary key' => array('id'),
);
$schema['content_watcher_subscriptions'] = array(
  'description' => 'Content watcher subscriptions',
  'fields' => array(
    'nid' => array(
      'description' => 'The primary identifier for a node.',
      'type' => 'int',
      'unsigned' => TRUE,
      'not null' => TRUE,
    ),
    'uid' => array(
      'description' => 'User id',
      'type' => 'int',
      'unsigned' => TRUE,
      'not null' => TRUE,
    ),
    'deleted' => array(
      'description' => 'Soft delete',
      'type' => 'int',
      'size' => 'small',
      'unsigned' => TRUE,
      'default' => 0,
    ),
  ),
);
$schema['content_watcher_mail'] = array(
  'description' => 'Content watcher mailed sent',
  'fields' => array(
    'nid' => array(
      'description' => 'The primary identifier for a node.',
      'type' => 'int',
      'unsigned' => TRUE,
      'not null' => TRUE,
    ),
    'mail' => array(
      'description' => 'Identifies if mail should be sent',
      'type' => 'int',
      'size' => 'small',
      'unsigned' => TRUE,
      'default' => 0,
    ),
  ),
);
return $schema;}
```

BILAGA 3: content_watcher.module

```
<?php

/**
 * @file
 * Watches for content changes and notifies about them.
 */

/**
 * Implements hook_menu().
 */
function content_watcher_menu(){
  $items = array();
  $items['admin/config/system/content-watcher'] = array(
    'title' => t('Content watcher'),
    'description' => t('Watches for content changes and notifies about them.'),
    'page callback' => 'drupal_get_form',
    'page arguments' => array('content_watcher_settings'),
    'access arguments' => array('edit content watcher settings'),
  );
  $items['admin/config/system/content-watcher/settings'] = array(
    'title' => t('Content watcher'),
    'type' => MENU_DEFAULT_LOCAL_TASK,
  );
  $items['admin/config/system/content-watcher/send'] = array(
    'title' => t('Send notifications'),
    'description' => t('Send Content watcher notifications'),
    'page callback' => 'drupal_get_form',
    'page arguments' => array('content_watcher_send_notifications_form'),
    'access arguments' => array('edit content watcher settings'),
    'type' => MENU_LOCAL_TASK,
  );
  // List subscriptions page
  $items['content-watcher-subscriptions/%'] = array(
    'title' => t('Content watcher subscriptions'),
    'description' => t('Content watcher subscriptions'),
    'page callback' => 'content_watcher_list_subscriptions',
    'page arguments' => array(1),
    'access arguments' => array('access content'),
    'type' => MENU_NORMAL_ITEM,
  );
  // Delete subscription
  $items['content-watcher-subscriptions/%/%/%/del'] = array(
    'title' => t('Delete Content watcher subscriptions'),
    'description' => t('Delete Content watcher subscriptions'),
    'page callback' => 'content_watcher_delete_subscriptions',
    'page arguments' => array(1,2,3),
    'access arguments' => array('access content'),
    'type' => MENU_NORMAL_ITEM,
```

```
  );
  return $items;
}

/**
 * Implements hook_permission().
 */
function content_watcher_permission(){
  return array(
    'edit content watcher settings' => array(
      'title' => t('Edit content watcher settings'),
      'description' => t('Edit content watcher settings'),
    ),
  );
}

/**
 * Creates the admin settings form.
 */
function content_watcher_settings(){
  // Generate array with node types
  $types = node_type_get_types();
  $type_options = array();
  foreach($types as $key=>$type){
    $type_options[$key] = $type->name;
  }
  // Generate array with all field names
  $fields_raw = field_info_fields();
  // Disable comment body from fields
  $not_accepted_fields = array('comment_body');
  $field_names = array();
  foreach($fields_raw as $field){
    if(!in_array($field['field_name'], $not_accepted_fields)){
      $field_names[$field['field_name']] = $field['field_name'];
    }
  }
  $form = array();
  // Form element with node types as checkboxes
  $form['content_watcher_node_types'] = array(
    '#type' => 'checkboxes',
    '#title' => t('Node types'),
    '#description' => t('Choose which node types should be watched for changes.'),
    '#default_value' => variable_get('content_watcher_node_types', array('')),
    '#options' => $type_options,
    '#required' => TRUE,
  );
  // Form element with field names as checkboxes
  $form['content_watcher_fields'] = array(
    '#type' => 'checkboxes',
    '#title' => t('Fields'),
```

```
    '#description' => t('Choose which fields should be watched for changes.'),
    '#default_value' => variable_get('content_watcher_fields', array('')),
    '#options' => $field_names,
    '#required' => TRUE,
  );
  // Form element with textbox for default e-mail content
  $form['content_watcher_subject'] = array(
    '#type' => 'textfield',
    '#title' => t('Email subject'),
    '#description' => t('Email subject of notification mail.'),
    '#default_value' => variable_get('content_watcher_subject', ''),
    '#required' => TRUE,
  );
  // Form element with textbox for default e-mail content
  $form['content_watcher_text'] = array(
    '#type' => 'textarea',
    '#title' => t('Email content'),
    '#description' => t('Email content to send when content has changed.'),
    '#rows' => 10,
    '#default_value' => variable_get('content_watcher_text', ''),
    '#required' => TRUE,
  );
  return system_settings_form($form);
}

/**
 * Creates the admin settings form.
 */
function content_watcher_send_notifications_form($form, &$form_state){
  $form['send_notifications'] = array (
    '#value' => t('Send Notifications'),
    '#type' => 'submit',
  );
  return $form;
}

/**
 * Handler for sending content watcher notifications.
 */
function content_watcher_send_notifications_form_submit($form, &$form_state){
  $result = db_query('SELECT email, cwu.id as uid, GROUP_CONCAT(cws.nid SEPARATOR \',\')
as nids FROM {content_watcher_users} cwu
    JOIN {content_watcher_subscriptions} cws ON cws.uid = cwu.id
    JOIN {content_watcher_mail} cwm ON cwm.nid = cws.nid
    WHERE cws.deleted = 0 AND cwm.mail > 0 GROUP BY cwu.id
  ');
  $ops = array();
  $ids = array();
  foreach($result as $record){
```

```
    $params = array('mail' => $record->email,'uid' => $record->uid,'ids' => explode(',',
$record->nids));
    $ops[] = array('content_watcher_send_notification',array($params));
    $ids = array_merge($ids, explode(',', $record->nids));
  }
  $num_updated = db_update('content_watcher_mail')
    ->fields(array(
    'mail' => 0,
    ))
    ->condition('nid', $ids, 'IN')
    ->execute();
  $batch = array(
    'title' => t('Send notifications'),
    'operations' => $ops,
    'finished' => 'content_watcher_send_batch_complete',
    'file' => drupal_get_path('module', 'content_watcher').'/content_watcher.module',
  );
  batch_set($batch);
}

/**
 * Callback function for batch sending notifications.
 */
function content_watcher_send_batch_complete($success, $results, $operations){
  if($success){
    drupal_set_message(t('Complete'));
  }else{
    drupal_set_message(t('Error'));
  }
}

/**
 * Batch function for sending content watcher notifications.
 */
function content_watcher_send_notification($params) {
  drupal_mail('content_watcher', 'notification', $params['mail'], 'en', $params);
}

/**
 * Implements hook_mail().
 */
function content_watcher_mail($key, &$message, $params) {
  switch($key) {
    case 'notification':
      $message['subject'] = t(variable_get('content_watcher_subject'));
      $links = "\r\n\r\nChanged pages:";
      foreach($params['ids'] as $link){
        $links .= "\r\n".$GLOBALS['base_url'].'/node/'.$link;
      }
```

```
      $own_subscriptions            =            "\r\n\r\nYour            subscriptions:
".$GLOBALS['base_url'].'/content-watcher-subscriptions/'.$params['uid'];
      $message['body'][]                              =                              varia-
ble_get('content_watcher_text').$links.$own_subscriptions;
      break;
  }
}

/**
 * Implements hook_block_info().
 */
function content_watcher_block_info(){
  $block['content_watcher_box']['info'] = t('Content watcher subscription block');
  $block['content_watcher_box']['cache'] = DRUPAL_CACHE_PER_PAGE;
  return $block;
}

/**
 * Implements hook_block_view().
 */
function content_watcher_block_view($block_name = ''){
  $block = array();
  if($block_name == 'content_watcher_box' && arg(0) == 'node' && is_numeric(arg(1))){
    $node = node_load(arg(1));
    $types = variable_get('content_watcher_node_types', array(''));
    if($types[$node->type] === $node->type){
      $form = drupal_get_form('content_watcher_notify_form');
      $block['subject'] = t('Get content updates');
      $block['content'] = theme('content_watcher_block', array('form' => $form));
    }
  }
  return $block;
}

/**
 * Implements hook_theme().
 */
function content_watcher_theme($existing, $type, $theme, $path) {
  $items = array(
    'content_watcher_block' => array(
      'template' => 'content_watcher_block',
      'variables' => array('form'),
    )
  );
  return $items;
}

/**
 * Implements hook_form().
 */
```

```
function content_watcher_notify_form($form, &$form_state){

  $nid = 0;
  if(is_numeric(arg(1)) && arg(0) == 'node'){
    $nid = arg(1);
  }
  $form['box'] = array(
    '#prefix' => '<div id="content-watcher-form-container">',
    '#suffix' => '</div>',
  );
  $form['box']['nid'] = array(
    '#type' => 'hidden',
    '#value' => $nid,
  );
  $form['box']['name'] = array(
    '#title' => t('Name'),
    '#type' => 'textfield',
    '#required' => 1,
  );
  $form['box']['email'] = array(
    '#title' => t('E-mail'),
    '#type' => 'textfield',
    '#required' => 1,
  );
  $form['box']['send'] = array(
    '#type' => 'submit',
    '#value' => t('Submit'),
    '#ajax' => array(
      'callback' => 'content_watcher_notify_form_submit_callback',
      'wrapper' => 'content-watcher-notify-form',
      'effect' => 'fade',
    ),
  );
  return $form;
}

/**
 * Content watcher notify form validation.
 */
function content_watcher_notify_form_validate($form, &$form_state){

  $nid = $form_state['values']['nid'];
  $email = $form_state['values']['email'];
  if(!valid_email_address($email) || $email == ''){
    form_set_error('email', t('Please check your E-mail'));
  }
  if(!is_numeric($nid)){
    form_set_error('nid', t('Wrong format'));
  }
```

```
  $result = db_query('SELECT * FROM {content_watcher_subscriptions} cws JOIN {con-
tent_watcher_users} cwu ON cwu.id = cws.uid WHERE cws.nid = :nid AND cwu.email =
:email', array(':nid' => $nid, ':email' => $email));
  if($result->rowCount() > 0){
    form_set_error('email', t('You have already subscribed to this page.'));
  }
}

/**
 * Content watcher notify form submit.
 */
 function content_watcher_notify_form_submit($form, &$form_state){
  $uid = 0;
  $result = db_query('SELECT id FROM {content_watcher_users} cwu WHERE cwu.email =
:email', array(':email' =>  $form_state['values']['email']));
  if($result->rowCount() > 0){
    foreach($result as $record){
      $uid = $record->id;
    }
    $num_updated = db_update('content_watcher_users')
      ->fields(array(
        'name' => $form_state['values']['name'],
      ))
      ->condition('id', $uid, '=')
      ->execute();
  }
  else{
    $uid = db_insert('content_watcher_users')
    ->fields(array(
      'name' => $form_state['values']['name'],
      'email' => $form_state['values']['email'],
    ))
    ->execute();
  }

  $nid = db_insert('content_watcher_subscriptions')
  ->fields(array(
    'nid' => $form_state['values']['nid'],
    'uid' => $uid,
  ))
  ->execute();
  drupal_set_message(t('Thank you for your subscription.'));
}

/**
 * Content watcher notify form callback.
 */
function content_watcher_notify_form_submit_callback($form, &$form_state){

  $errors = form_get_errors();
```

```
  if(sizeof($errors) > 0){
    return drupal_render($form);
  }
  else{
    return '<div class="success"><p>'.t('Thank you for your subscrip-
tion.').'</p></div>';
  }
}

/**
 * Generate list subscriptions page output.
 */
function content_watcher_list_subscriptions($s){
  $output = '';
  if(is_numeric($s)){
    $subscriptions = _content_watcher_get_subscriptions($s);
    if(!$subscriptions){
      $output = t('No subscriptions.');
    }
    else{
      $output .= theme('table', $subscriptions, array('id' => 'content-watcher-
subscriptions-list'));
    }
  }
  return $output;
}

/**
 * Get subscriptions.
 */
function _content_watcher_get_subscriptions($s){
  $result = db_query('SELECT cws.nid, cws.deleted, n.title, cwu.email FROM {con-
tent_watcher_subscriptions} cws JOIN {content_watcher_users} cwu ON cwu.id = cws.uid
JOIN node n ON n.nid = cws.nid WHERE cwu.id = :id AND n.status = :status', array(':id'
=> $s, ':status' => 1));
  if($result->rowCount() == 0){
    return FALSE;
  }
  else{
    $table = array();
    $table['attributes'] = array('class' => array('content-watcher-subscription'));
    $table['header'] = array(array('data'=>'Page','class'=>'label'), ar-
ray('data'=>t('Remove'),'class'=>'del'));
    $table['rows'] = array();
    $email = '';
    foreach($result as $record){
      $delete = ($record->deleted == '0')?'1':'0';
      $rmtext = ($record->deleted == '0')?t('remove'):t('cancel');
      $email = $record->email;
      $title = $record->title;
```

```
      $row = array();
      $row['class'] = array('deleted-'.$delete);
      $row['data'] = array(
        array(
          'data'=>l($title, 'node/'.$record->nid),
          'class'=>'label'
        ),
        array(
          'data'=>l($rmtext,            'content-watcher-subscriptions/'.$s.'/'.$record-
>nid.'/'.$delete.'/del'),
          'class'=>'del'
        )
      );
      $table['rows'][] = $row;
    }
    return $table;
  }
}

/**
 * Soft delete subscriptions.
 */
function content_watcher_delete_subscriptions($user, $nid, $op){
  if(is_numeric($user)){
    $u = 0;
    $user = db_query('SELECT id FROM {content_watcher_users} cwu WHERE cwu.id = :id LIM-
IT 1', array(':id' => $user))->fetchField();

    if($op != 1)$op = 0;
    $num_updated = db_update('content_watcher_subscriptions')
    ->fields(array(
      'deleted' => $op,
    ))
    ->condition('uid', $user)
    ->condition('nid', $nid)
    ->execute();
  }
  return t('Done.') . ' ' . l(t('Go back to subscriptions list.'), 'content-watcher-
subscriptions/'. $user .'');
}

/**
 * Implements hook_node_presave().
 */
function content_watcher_node_presave($node){
  global $content_watcher_original_node;
  $content_watcher_original_node = node_load($node->nid);
}
```

```
/**
 * Implements hook_node_update().
 */
function content_watcher_node_update($node) {
  global $content_watcher_original_node;
  $n = node_load($node->nid, NULL, TRUE);
  $defined_content_types = variable_get('content_watcher_node_types', array(''));
  $defined_fields = variable_get('content_watcher_fields', array(''));
  if(in_array($node->type, $defined_content_types)){
    $changed = FALSE;
    // Loop through fields
    foreach($defined_fields as $field){
      // Check if field exists and has changed
      if(isset($content_watcher_original_node->$field)  &&  isset($n->$field)  &&  $con-
tent_watcher_original_node->$field != $n->$field){
        $changed = TRUE;
      }
    }
    // Check if node is watched by anyone
    $count  =  db_query("SELECT  COUNT(cws.nid)  FROM  {content_watcher_subscriptions}  cws
WHERE cws.nid = :cws", array(':cws' => $node->nid))->fetchField();
    if($changed && $count > 0){
      db_merge('content_watcher_mail')
        ->key(array('nid' => $node->nid))
        ->fields(array(
          'mail' => 1,
        ))
        ->execute();
    }
  }
}

/**
 * Implementation of hook_cron_queue_info()
 */
function content_watcher_cron_queue_info() {
  $queues['content_watcher'] = array(
    'worker  callback'  =>  'content_watcher_send_notification',  //  This  is  the  callback
function for each queue item.
    'time' => 15, // This is the max run time per cron run in seconds.
  );
  return $queues;
}

/**
 * Implementation of hook_cron()
 */
function content_watcher_cron() {
  $queue = DrupalQueue::get('content_watcher');
```

```
  $result = db_query('SELECT email, cwu.id as uid, GROUP_CONCAT(cws.nid SEPARATOR \',\')
as nids FROM {content_watcher_users} cwu
    JOIN {content_watcher_subscriptions} cws ON cws.uid = cwu.id
    JOIN {content_watcher_mail} cwm ON cwm.nid = cws.nid
    WHERE cws.deleted = 0 AND cwm.mail > 0 GROUP BY cwu.id
  ');
  $ids = array();
  foreach($result as $record){
    $params = array('mail' => $record->email, 'uid' => $record->uid, 'ids' => ex-
plode(',', $record->nids));
    $queue->createItem($params);
    $ids = array_merge($ids, explode(',', $record->nids));
  }
  $num_updated = db_update('content_watcher_mail')
    ->fields(array(
      'mail' => 0,
    ))
    ->condition('nid', $ids, 'IN')
    ->execute();
}
```

BILAGA 4: content_watcher_block.tpl.php

```
<div id="content-watcher-block-wrapper">
  <div class="content-watcher-form">
    <p><?php print t('Fill in your details to get notified about changes to this page');?></p>
    <?php print render($form); ?>
  </div>
</div>
```

BILAGA 5: content_watcher.css

```
#content-watcher-block-wrapper h2{
  font-size: 1.071em;
}
#content-watcher-block-wrapper{
  font-size: 0.8em;
}
#content-watcher-block-wrapper div.messages ul{
  margin: 0;
}
#content-watcher-block-wrapper div.messages{
  padding: 8px 0 16px 50px;
  margin: 0 0 10px 0;
}
#content-watcher-block-wrapper .content-watcher-form{
  color: #3a3a3a;
}
#content-watcher-block-wrapper .content-watcher-form input.form-text{
  border-color: #cccccc;
  width: 100%;
}
#content-watcher-block-wrapper .content-watcher-form .form-required{
  color: #ff0000;
}
```

BILAGA 6: content_watcher.js

```
(function ($) {
  Drupal.behaviors.content_watcher_subscription = {
    attach: function (context, settings) {
      $('table.content-watcher-subscription td.del a').click(function(){
        $.ajax({
          url: $(this).attr('href'),
          context: this,
          success: function(){
            $(this).parent().parent().toggleClass('deleted-1 deleted-0');
            var urlparts = $(this).attr('href').split('/');
            var newurl = '';
            var newtext = '';
            for(var part in urlparts){
              if(part == 4){
                if(urlparts[part] == 0){
                  urlparts[part] = 1;
                  newtext = Drupal.t('remove');
                }
                else{
                  urlparts[part] = 0;
                  newtext = Drupal.t('cancel');
                }
              }
              newurl += urlparts[part];
              if(part < urlparts.length - 1) newurl += '/';
            }
            $(this).attr('href',newurl)
            $(this).text(newtext)
          }
        });
        return false;
      });
    }
  };
}(jQuery));
```

BILAGA 7: Skärmkapning från adressen http://www.toimitila.fi