Inköpsverktyg för projektet RCMS och ramverket Symfony

Jani Rapo

Examensarbete

Informations- och medieteknik

2012

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informations- och medieteknik
Identifikationsnummer:	3341
Författare:	Jani Rapo
Arbetets namn:	Inköpsverktyg för projektet RCMS och ramverket Symfony.
Handledare (Arcada):	Hanne Karlsson
Uppdragsgivare:	Crafthouse Oy

Sammandrag:
I en detaljhandelskedja kan man lätt identifiera att verksamheten är uppdelad i två olika processer. Den ena processen sker vid kassan då man säljer varor till kunderna, medan den andra processen handlar om administrativa uppgifter inom kedjan. För att kontrollera dessa två processer använder man två olika programsystem. Uppdraget med arbetet var att planera och skapa Inköpsverktyget, dvs. ett delprojekt av administrationsprogramvaran RCMS. Inköpsverktyget kan användas för att beställa varor från leverantörer och administrera dessa inköpsbeställningar, lägga till och administrera leverantörer, automatiskt generera inköpsbeställningar samt registrera mottagning av hela inköpsbeställningar eller enskilda produkter.

Den skriftliga delen av examensarbetet är indelad i en teoretisk del och en praktisk del. I den teoretiska delen beskrivs Symfony-ramverkets huvudsakliga delar, en överblick av designmönstret som ramverket grundar sig på och hur designmönstret är realiserat i ramverket. Målsättningen med den praktiska delen är att redogöra för de tekniska lösningarna bakom projektet och hur databasmodellen är uppbyggd.

Endast den grundläggande funktionaliteten hos Inköpsverktyget ingick i examensarbetet, medan den slutliga finslipningen av Inköpsverktyget och resten av funktionaliteten i RCMS kommer att utvecklas senare. Val av plattform, utvecklingsspråk och ramverk gjordes av uppdragsgivaren.

Resultatet av arbetet blev en fullständig kravspecifikation och en effektiv samt utvidgningsbar databasuppbyggnad för Inköpsverktyget som en del av projektet RCMS. Själva praktiska realiseringen av Inköpsverktyget utvecklades så långt att man kan använda verktyget, men det kräver en hel del vidareutveckling och omfattande testning innan det är färdigt för produktionsanvändning. Ramverket Symfony visade sig också vara ett väldigt mångsidigt, användbart och effektivt ramverk för webbapplikationsutveckling.

Nyckelord:	Symfony, ramverk, webbapplikation, PHP, Crafthouse Oy, lagerhantering, programvaruprojekt, databasdesign
Sidantal:	59
Språk:	Svenska
Datum för godkännande:	20.1.2012

DEGREE THESIS	
Arcada	
Degree Programme:	Information and Media Technology
Identification number:	3341
Author:	Jani Rapo
Title:	Buyer´s desktop for the RCMS project and the Symfony framework.
Supervisor (Arcada):	Hanne Karlsson
Commissioned by:	Crafthouse Oy

Abstract:
In a retail chain you can easily identify that the activity is divided into two separate processes. One process takes place by the counter, when selling goods to customers, while the other takes place in the background, transparent to the customer, and involves administrative tasks. Two different pieces software systems are used to control these processes. The objective of this thesis was to design and create the Buyerôs Desktop, which is a part of the administrative software, the RCMS. The Buyerôs Desktop can be used for ordering goods from vendors, to administer these purchase orders, to add and administer vendors, to automatically generate purchase orders and to receive complete purchase orders or individual products.

This thesis report is divided into a theoretical part and into a practical part. The theoretical part describes the Symfony frameworkôs main components, an overview of the design pattern that the framework is based on and how the design pattern is implemented into the framework. The aim of the practical part is to describe the technical solutions behind the project and how the database model is built.

Only the basic functionality of the Buyerôs Desktop was a part of the thesis, while the final refinement of the Buyerôs Desktop and the rest of the RCMS will be developed later. The choice of platform, programming language and framework was made by the commissioner.

The result was a complete requirement specification and an effective and expandable database for the Buyerôs Desktop as a part of the RCMS project. The practical implementation of the Buyerôs Desktop in itself was developed to such a point, that it can be used, but it requires fine tuning and extensive testing before it is ready for production use. The Symfony framework proved to be a very versatile, useful and effective framework for development of web applications.

Keywords:	Symfony, framework, web application, PHP, Crafthouse Oy, inventory management, software project, database design
Number of pages:	59
Language:	Swedish
Date of acceptance:	20.1.2012

INNEHÅLL

Figurer

Förkortningar

AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
ARTS	Association For Retail Technology Standards
BSD	Berkley Software Distribution
CRUD	Create, Retrieve, Update Delete
CSS	Cascading Style Sheets
CSRF	Cross-Site Request Forgery
CVS	Concurrent Versions System
DOM	Document Object Model
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
MVC	Model-View-Controller
NFR	National Retail Federation
OOP	Object-Oriented Programming
ORM	Object-Relational Mapping
SCM	Software Configuration Management
PHP	PHP Hypertext Preprocessor
POS	Point Of Sale
RAD	Rapid Application Development
RCMS	Retail Chain Management System
SQL	Structured Query Language
STL	Standard Template Library
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WAPP	Windows, Apache, PostgreSQL, PHP
XLIFF	XML Localization Interchange File Format
XSS	Cross-Site Scripting
YAML	YAML Ain't Markup Language

1 INLEDNING

1.1 Utgångspunkt

I en detaljhandelskedja kan man lätt identifiera att verksamheten är uppdelad i två olika processer. Den ena processen sker vid kassan då man säljer varor till kunderna, medan den andra processen handlar om administrativa uppgifter inom kedjan. Som exempel på dessa administrativa uppgifter kunde nämnas skapande och upprätthållande av varor, kampanjer och användare, samt beställning av varor för försäljning och övervakning av rapporter etc.

För att kontrollera dessa två processer använder man två olika programvaror. I projektet som detta examensarbete är en del av, går kassaprogramvaran under arbetsnamnet POS (*eng. Point Of Sale*) och administrationsprogramvaran under arbetsnamnet RCMS (*eng. Retail Chain Management System*).

Uppdraget med arbetet var att planera och skapa ett delprojekt av RCMS. Meningen var att ersätta uppdragsgivarens nuvarande detaljhandelshanteringsprogram med en ny, mer smidig, mer dynamisk och lättare hanterbar programvara, som också är skapad med modernare verktyg. Den nya programvaran skall också kunna säljas utomlands; den nuvarande produkten har stöd endast för finska, och lämpar sig därför inte för den internationella marknaden.

Min andel av RCMS-projektet är Inköpsverktyget, där programmets användare kan beställa varor från leverantörer och administrera dessa inköpsbeställningar, lägga till och administrera leverantörer, automatiskt generera inköpsbeställningar samt registrera leverans av hela inköpsbeställningar eller enskilda produkter.

Som uppdragsgivare för detta arbete fungerar Crafthouse Oy och projektet är uppbyggt med PHP-ramverket Symfony medan webbplatsen upprätthålls i WAPP-miljö (Windows-Apache-PostgreSQL-PHP).

1.2 Målsättning

Målsättning för examensarbetet var att planera och utveckla Inköpsverktyget för RCMS samt att planera och skapa därtill hörande databasstruktur. Den utvidgade databasstrukturen skulle smidigt integreras med den existerande databasstrukturen utan att data i något skede skulle dupliceras, dvs. man önskade undvika problem med redundans. Ett ytterligare mål med arbetet var en utredning och evaluering av hur ramverket Symfony fungerar och hur man kan utnyttja det för programvaruutveckling. Den skriftliga delen av examensarbetet är indelad i en teoretisk del och en praktisk del.

I den teoretiska delen beskrivs Symfony-ramverkets huvudsakliga delar, en överblick av designmönstret som ramverket grundar sig på och hur designmönstret är realiserat i ramverket. Målsättningen med den praktiska delen är att redogöra för de tekniska lösningarna bakom Inköpsverktyget och hur databasmodellen är uppbyggd.

När hela RCMS är färdigt kommer vi att ha ett redskap som kan användas av alla slags butiker från enskilda med en dator till stora detaljhandelskedjor med hundratals butiker och tusentals kassaenheter i Finland eller utomlands. Vi kommer att ha ett pålitligt system som är lätt att installera, utvidga, använda och konfigurera.

1.3 Avgränsningar

Endast en prototyp av Inköpsverktyget görs av mig, medan den slutliga finslipningen av Inköpsverktyget och övrig funktionalitet i RCMS utvecklas av uppdragsgivarens andra resurser i ett senare skede.

Jag fick inte själv välja varken plattform, utvecklingsspråk eller ramverk, utan PHP-ramverket Symfony och WAPP-miljön valdes av uppdragsgivaren. En objektorienterad databas kunde inte heller utnyttjas eftersom den existerande databasen som utvidgades var en relationsdatabas.

2 RAMVERKET SYMFONY

2.1 Introduktion

Ett ramverk förenklar applikationsutveckling genom att automatisera många av de mönster som används vid olika skeden av utvecklingen. Ett ramverk ger också struktur åt koden och uppmanar därmed utvecklaren att skriva bättre, mer lättläst och lättare underhållbar kod. Slutligen gör användningen av ett ramverk programmering lättare, eftersom det packar in komplexa operationer i enkla uttryck.

Symfony är ett komplett ramverk och planerat för att optimera utvecklingen av webbapplikationer med hjälp av sina mångsidiga egenskaper. Ramverket innehåller flera verktyg och klasser, som strävar efter att förkorta utvecklingstiden av komplexa webbapplikationer. Dessutom automatiserar ramverket generella uppgifter, så att utvecklare helt och hållet kan fokusera på detaljerna i ett program. Allt detta innebär att man inte behöver uppfinna hjulet på nytt varje gång man skall bygga upp en ny webbapplikation.

Symfony är helt och hållet skriven i PHP. Den är kompatibel med de flesta tillgängliga databasmotorerna, inklusive MySQL, PostgreSQL, Oracle och Microsoft SQL Server. Symfony är körbar på Unix- och Windows-plattformar. Användningen av Symfony-ramverket i PHP baserade webbapplikationer förbättrar kvaliteten på koden genom att strukturera och ordna filer samtidigt som ramverket skapar ett nytt användningssätt för existerande teknik (Jarmoğowicz et al. 2008 s. 597).

Den första versionen av Symfony lanserades i oktober 2005 av projektets grundare Fabien Potencier. Fabien är verkställande direktör för Sensio, en fransk webbyrå känd för sin innovativa syn på webbutveckling. År 2003 studerade Potencier vilka befintliga öppen källkodsverktyg som fanns tillgängliga för utveckling av webbapplikationer, skrivna i PHP. Han kom till den slutsatsen att det inte fanns sådana som skulle uppfylla hans krav. När PHP 5 lanserades, bestämde han att de tillgängliga verktygen hade nått en tillräckligt mogen nivå för att kunna integreras till ett fullt utrustat ramverk. Han tillbringade därefter ett år med att utveckla kärnan till Symfony och han baserade sitt

arbete på ramverket *Mojavi Model-View-Controller* (MVC), på *Propel* objekt-relationell kartläggning (ORM) och på *Ruby on Rails* hjälpfunktioner for template-hantering. (Potencier & Zaninotto s. 12-13)

Man kan använda sig av Symfony oavsett om man är en PHP-expert eller en nybörjare i webbapplikationsprogrammering. När man bestämmer sig för om man skall använda sig av Symfony eller inte, är storleken på projektet den viktigaste faktorn.

Om man vill utveckla en enkel webbplats med fem till tio sidor, begränsad databastillgång och inga krav på prestanda eller dokumentation, då vinner man inte så mycket med att använda ett ramverk, utan det kan vara lönsamt att hålla sig till enbart PHP. Å andra sidan, om man vill utveckla mer komplexa webbapplikationer, med mångsidiga databasförfrågningar kan det hända att enbart PHP inte räcker till.

Då man planerar att effektivt upprätthålla eller utvidga sin applikation i framtiden krävs det att koden är lätt, läsbar och effektiv. Om man vill åstadkomma detta på ett snabbt och effektivt sätt lönar det sig att bekanta sig med ramverket Symfony. (Potencier & Zaninotto s. 13-14)

2.2 Grundläggande begrepp

Innan man kan börja arbeta med Symfony finns det en del grundläggande begrepp man bör förstå. Bland dessa är de viktigaste: PHP, objektorienterad programmering (Object-Oriented Programming, OOP), magic methods, Object-Relational Mapping (ORM), Rapid Application Development (RAD) och YAML (YAML Ainât Markup Language).

Ramverket Symfony är utvecklat i programmeringsspråket *PHP* och är främst avsett för att bygga webbapplikationer med samma språk. Därför krävs det att man har goda kunskaper i PHP och i objektorienterad programmering för att få ut det mesta av ramverket**.** Den lägsta versionen av PHP, som krävs för att köra Symfony är PHP 5.2.4. Wikipedia (2011a) beskriver objektorienterad programmering som en programmeringsmetod i vilken ett program kan innehålla en varierande uppsättning

objekt som interagerar med varandra. PHP realiserar objektorienterade paradigmer av klasser, objekt, metoder, arv och mycket mer.

En stor fördel som man får av PHP:s objekt är deras förmåga att använda sig av magiska metoder (*Magic Methods*). Dessa är metoder som kan användas för att åsidosätta standardbeteendet av klasser utan att behöva modifiera koden som ligger utanför. De gör PHP-syntaxen mer kortskriven och mer utbyggbar. De är lätta att känna igen, eftersom namnen på de magiska metoderna börjar med två understreck (__). Till exempel vid visning av ett objekt letar PHP underförstått efter en *__toString( )*-metod för objektet i fråga, för att se om ett anpassat visningsformat har definierats av utvecklaren:

```
$myObject = new myClass();
echo $myObject;

// motsvaras av:
echo $myObject->__toString();
```

Databaser är idag oftast relationella. PHP och Symfony är objektorienterade. För att komma åt databasen på ett objektorienterat sätt, krävs ett gränssnitt för att översätta objektlogik till relationslogik. Detta gränssnitt kallas för objekt-relationell kartläggning, eller ORM (eng. *Object-Relational Mapping*). En ORM består av objekt, som ger tillgång till data och innehåller funktionalitetsregler.

En fördel med ett objekt-relationellt abstraktionslager är att det hindrar programmeraren från att använda databasspecifik syntax vid programmering. Det översätter automatiskt modellobjektskallelser till SQL-förfrågningar, som är optimerade för den databas man använder. Detta innebär att det går lätt till att byta till ett annat databassystem mitt i ett projekt. Ett praktiskt exempel på denna fördel är om man föreställer sig att man måste snabbt skapa en prototyp av ett program, men kunden inte ännu har bestämt sig för vilket databassystem som bäst skulle passa hans behov. Då kan man börja bygga applikationen med till exempel SQLite och sedan vid behov byta till MySQL, PostgreSQL eller Oracle, då klienten är redo för att fatta ett beslut. Eftersom man har använt sig av en ORM, går byte av databassystemet genom att enbart ändra en rad i en konfigurationsfil, i övrigt fungerar allt på samma sätt.

Genom att använda objekt istället för databasrader, och klasser istället för tabeller, får man också en annan fördel: man kan lägga till nya accessmetoder till tabellerna. Till exempel, om man har en tabell som kallas *client* med två kolumner, *FirstName* och *LastName*, kan man vilja kunna skriva ut enbart *Name* istället för dessa två skilt för sig. I en objektorienterad värld behöver man endast lägga till en ny accessmetod till *Client* för att åstadkomma detta, så här:

```
public function getName()
{
      return $this->getFirstName().' '.$this->getLastName();
}
```

Symfony stöder de två populäraste öppen källkods ORM:s i PHP: *Propel* och *Doctrine*. Båda är sömlöst integrerade i Symfony. När man skapar ett nytt Symfony projekt, måste man välja vilkendera man kommer att använda sig av. I detta arbete behandlas endast Doctrine, eftersom det var uppdragsgivarens val.

Det förväntas ofta av programvaruutvecklare att de skall kunna ändra applikationsstrukturen i ett projekt väldigt snabbt. Lyckligtvis gör användningen av skriptspråk såsom Python, Ruby och PHP det lätt att tillämpa utvecklingssmetoder såsom snabb utveckling av applikationer (eng. R*apid Application Development, RAD*) eller agila metoder.

En av huvudidéerna i dessa utvecklingsmetoder är att man börjar applikationsutvecklingen så fort som möjligt, så att kunden kan bedöma en fungerande prototyp och erbjuda vidare vägledning redan i ett tidigt skede. Efter det byggs resten av applikationen upp i en iterativ process, där mer och mer utvecklade versioner släpps ut i korta utvecklingscykler.

Symfony är det perfekta verktyget för RAD. I själva verket byggdes Symfony upp av en webbagentur som redan utnyttjade principen av RAD för sina egna projekt. Detta innebär att Symfony inte handlar om ett nytt programmeringsspråk, utan snarare om att använda rätt teknik för att bygga upp applikationer på ett mer effektivt sätt. (Potencier & Zaninotto s. 14-16)

Det sista grundläggande begreppet som man bör kunna är YAML. Enligt den officiella YAML-hemsidan är YAML en användbar dataserialiseringsstandard för alla programmeringsspråk. Med andra ord, är YAML ett mycket enkelt språk som används för att beskriva data på ett XML-liknande sätt men med en mycket enklare syntax (YAML 2011). Det är speciellt användbart för att beskriva data som kan översättas till arrayer och hashar, så här:

```
$house = array(
  'family' => array(
    'name' => 'Doe',
    'parents' => array('John', 'Jane'),
    'children' => array('Paul', 'Mark', 'Simone')
  ),
  'address' => array(
    'number' => 34,
    'street' => 'Main Street',
    'city' => 'Nowheretown',
    'zipcode' => '12345'
  )
);
```

Namnet YAML är en rekursiv akronym för ò*YAML Ain't Markup Language*ó och formatet har funnits sedan 2001, och det finns YAML-parsers tillgängliga för flera olika programmeringsspråk. YAML är mycket snabbare att skriva än XML (krävs inga sluttaggar eller explicita citationstecken), och det är mer kraftfullt än .ini-filer (som inte stöder hierarki) (Wikipedia 2011b). Det är därför Symfony använder YAML som utgångsspråk för att lagra konfigurationer.

2.3 Allmänt om Symfony

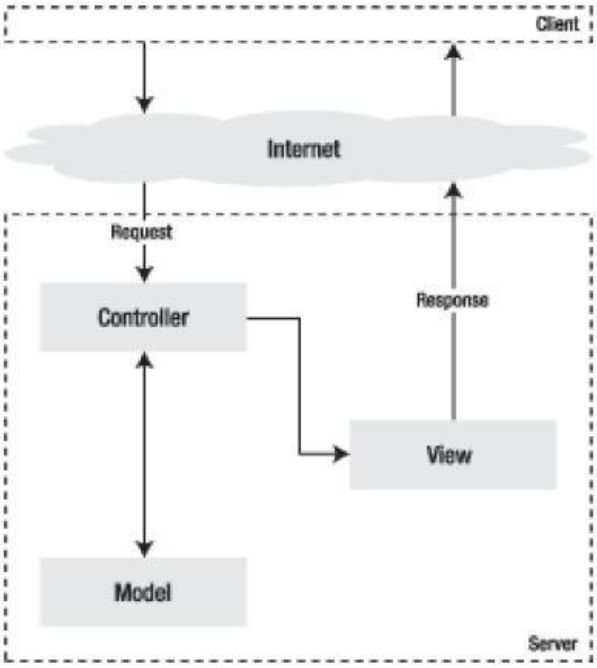
2.3.1 MVC

Symfony baserar sig på det klassiska designmönstret MVC (*Model-View-Controller*), som består av tre lager:

- Modellen, som representerar informationen på vilken applikationen driver - affärslogiken.
- Vyn, som renderar modellen till en webbsida, som anpassar sig för interaktion med användaren.

- Kontrollern, som reagerar på användarens handlingar och anropar förändringar i endera modellen eller vyn, beroende på handlingarna.

Genom att separera programkoden i vyer som innehåller information om hur webbsidan skall se ut, i en datamodell som hanterar information samt i en kontroller som hämtar data från modellen och använder den för att rita upp vyn får man en tydlig separation mellan programkod och HTML-kod. Det gör att man snabbt kan anpassa utseendet, utan att behöva röra affärslogiken och applikationen blir lättare att underhålla. Användningen av MVC-designmönstret är ett enkelt sätt att skapa kod som är lättare att underhålla, anpassa och felsöka. Figur 1 visar en grafisk representation av MVC-designmönstret. (Görling 2009 s. 133)



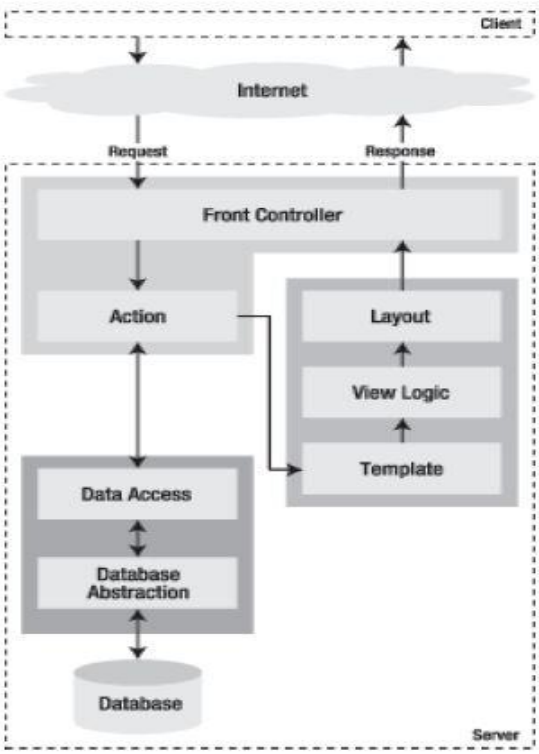
Figur 1. Grafisk representation av MVC-designmönstret. (Potencier & Zaninotto 2010)

Modellen går dessutom ännu att dela in i ett data-accesslager och i ett databas-abstraktionslager. På det sättet behöver inte data-accessfunktionerna använda databasspecifika förfrågningar, utan de kan kalla på andra funktioner, som i sig själva skapar förfrågningarna. Nyttan är att om man bestämmer sig för att byta databassystem senare, räcker det med att man uppdaterar abstraktionslagret.

Vyn kan också dra nytta av lite kodseparation. En webbsida innehåller oftast överensstämmande element inom hela applikationen: sidhuvuden, den grafiska layouten, sidfoten, och den globala navigeringsmenyn. Endast den inre delen av sidan ändras. Det är därför vyn indelas i en layout och en template (mall). Layouten är oftast likadan hos hela applikationen eller hos en grupp av sidor. Templaten ger bara form åt variablerna som tillhandahålls av kontrollern. Viss logik behövs för att dessa komponenter skall kunna samverka och detta lager för vylogiken går under namnet vy (eng. *view*).

I en större applikation har kontrollern mycket arbete och en viktig del av detta arbete är gemensamt för alla kontrollers i applikationen. Till de gemensamma uppgifterna hör hantering av förfrågningar, säkerhetshantering, laddning av applikationskonfigurationen och liknande sysslor. Av denna orsak delas kontrollern upp i en frontkontroller, vilken är gemensam för hela applikationen, och actions (handlingar), vilka innehåller endast sidspecifika kontrollers. En av de stora fördelarna med en separat frontkontroller är att det erbjuder en unik ingång till hela programmet. Om man bestämmer sig för förhindra tillgången till programmet, behöver man bara ändra skriptet för frontkontrollern. I en applikation utan frontkontroller, måste tillgången till varje enskild kontroller förhindras. (Potencier & Zaninotto s. 23-27)

Figur 2 visar en grafisk representation av MVC-designmönstret med vidare uppdelning.



Figur 2. Grafisk presentation av MVC-designmönstret med vidare uppdelning. (Potencier & Zaninotto 2010)

2.3.2 Projektstruktur

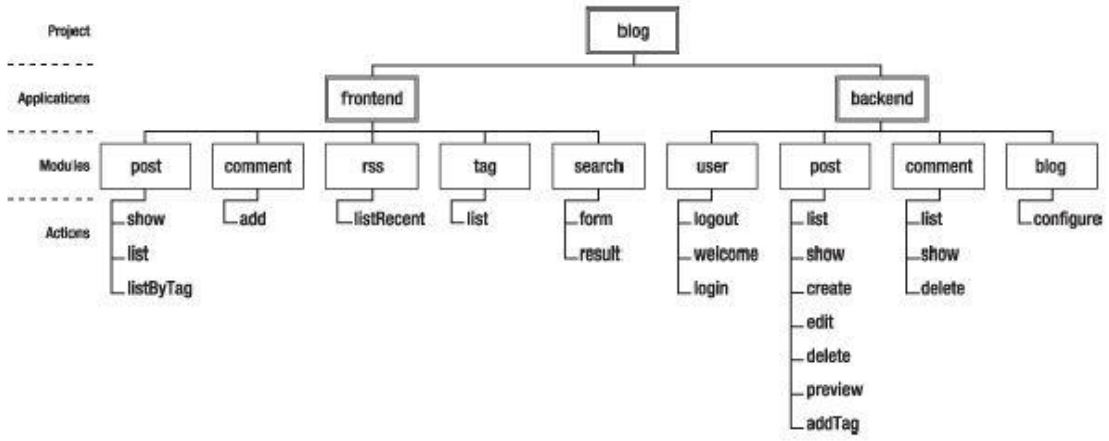
I ett Symfony-projekt är verksamheten logiskt grupperad i applikationer. Vanligtvis kan en applikation köras individuellt oberoende av andra applikationer inom samma projekt. I de flesta fallen innehåller ett projekt två applikationer: en för front-office och en för back-office, vilka delar samma databas. Front-office delen av projektet består av den delen som vanliga användaren har tillgång till, medan back-office delen omfattar den administrativa delen. Man kan också ha ett projekt som innehåller många mini-sajter, med varje webbplats som en egen applikation.

Varje applikation är en uppsättning av en eller flera moduler. En modul är oftast en sida eller en grupp av sidor med liknande syfte. En typisk applikation kunde tänkas ha modulerna hem, blogg, varukorg, konto osv.

Modulerna innehåller handlingar (actions), som representerar olika åtgärder som går att göra i en modul. Modulen *varukorg* kunde till exempel innehålla handlingarna *lägg till*,

visa, *uppdatera* och *töm*. Generellt beskrivs handlingarna med ett verb. Att handskas med handlingar är nästan som att arbeta med sidor i en klassisk webbapplikation, även om två olika handlingar kan leda till samma sida. (Potencier & Zaninotto s. 29)

Figur 3 presenterar hur koden är organiserad i ett blogg-projekt, i en projekt/applikation/modul/handling-struktur.



Figur 3. Exempel på projektuppbyggnad. (Potencier & Zaninotto 2010)

Alla webbapplikationsprojekt har i stort sätt samma typ av innehåll:

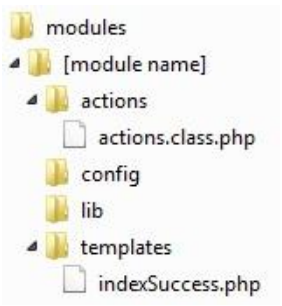
- En databas, såsom MySQL eller PostgreSQL
- Statiska filer (HTML, bilder, JavaScript filer, CSS filer etc.)
- Uppladdade filer
- PHP klasser och bibliotek
- Utomstående bibliotek (tredje parts skript)
- Loggfiler
- Konfigurationsfiler

Symfony erbjuder en vanlig filträdstruktur för att organisera allt detta innehåll på ett logiskt sätt, i överensstämmelse med den valda arkitekturen (designmönstret MVC och projekt/applikation/modul/handling-grupperingen). Denna trädstruktur skapas automatiskt vid initieringen av varje projekt, applikation eller modul. Naturligtvis går

trädstrukturen också att anpassa enligt behov. I Figur 4 kan man se hur filträdstrukturen för en standard Symfony applikation ser ut:

Figur 4. Filträdstrukturen i ett standard Symfony-projekt.

Varje applikation innehåller en eller flera moduler. Varje modul har sin egen underkatalog i *modules*-katalogen och namnet på denna katalog väljs när modulen skapas. I Figur 5 kan man se den typiska filträdstrukturen i en modul:



Figur 5. Typisk filträdstruktur i en modul.

2.4 Kontroller-lagret

Kontroller-lagret, som innehåller den kod som länkar affärslogiken med presentationen, är indelat i olika komponenter som används för olika ändamål:

- Frontkontrollern är en unik ingång till applikationen. Den laddar konfigurationen och bestämmer vilken handling som skall exekveras.
- Handlingarna innehåller den applikativa logiken. De granskar förfrågans integritet och utarbetar de uppgifter som behövs av presentationslagret.
- Sessionsobjekten, som innehåller data om användaren.
- Filter är delar av kod som körs för varje förfrågning, före eller efter handlingen. Till exempel, säkerhets- och valideringsfilter används ofta i webbapplikationer. Man kan utvidga ramverket genom att skapa egna filter.

2.4.1 Frontkontrollern

Alla webbförfrågningar hanteras av en enda frontkontroller, som är en unik ingång till hela programmet i en viss miljö.

När frontkontrollern får en förfrågning, använder den routing-systemet för att koppla ihop en handling och en modul med webbadressen som användaren vill navigera till. Följande URL: *http://localhost/index.php/mymodule/myAction* kallar till exempel på *index.php*-skriptet (frontkontrollern) och kommer att förstås som en kallelse på handlingen *myAction* i modulen *mymodule*. (Potencier & Zaninotto s. 74)

2.4.2 Handlingar (Actions)

Handlingarna är applikationens hjärta, för de innehåller applikationens logik. De kallar på modellen och definierar variabler för vyn. När man gör en webbförfrågning i Symfony, definierar URL:n en handling och förfrågningsparametrarna. (Potencier & Zaninotto s. 76)

2.4.3 Sessioner

Symfony hanterar automatiskt användarsessioner och kan hålla ständig data för användarna mellan förfrågningarna. Den använder inbyggda PHP-sessionshanteringsmekanismer och förbättrar dem genom att göra dem mer konfigurerbara och enklare att använda. (Potencier & Zaninotto s. 84)

2.4.4 Säkerhet vid hantering av handlingar

Möjligheten att utföra en handling kan begränsas till användare med vissa rättigheter. Med hjälp av de verktyg som Symfony erbjuder för detta ändamål kan man skapa säkra applikationer, där användarna måste verifieras innan vissa funktioner eller delar av applikationen blir åtkomliga. Att säkra en applikation kräver två steg: att deklarera säkerhetskraven för varje handling och att logga in användare med behörighet så att de kan få tillgång till dessa säkra handlingar.

I modulens konfigureringsfil *security.yml* kan man endera ställa in varje handling till säker skilt för sig eller så alla på en gång. (Potencier & Zaninotto s. 88)

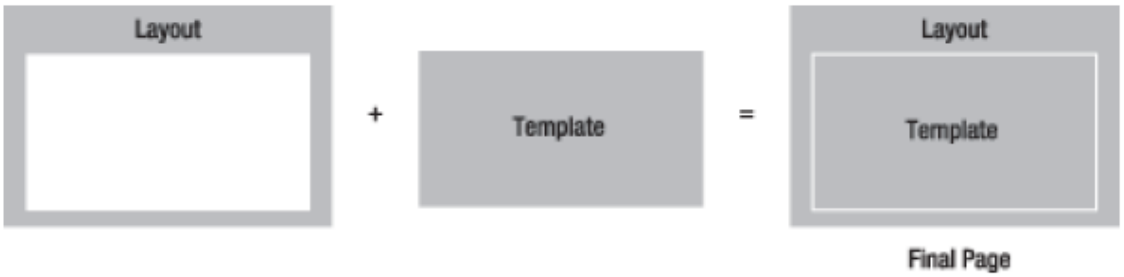
2.5 Vylagret

Vylagret ansvarar för att rendera utmatning av data som korreleras med en viss handling. I Symfony består vyn av flera delar, där varje del är utformad på ett sådant sätt att den lätt kan ändras. I detta kapitel behandlas hur sidorna är uppbyggda i Symfony och vilka dessa olika delar av vylagret är.

2.5.1 Siduppsättning

Om man skulle inspektera noggrannare vilken som helst template-fil i ett Symfony-projekt, så skulle man märka att den inte är en fullständig XHTML-fil. Detta innebär bland annat att *DOCTYPE*-definitionen saknas och av att <*html*>- och <*body*>-taggarna saknas från filerna. Orsaken till detta är att man hittar dem på ett annat ställe i applikationen, i en fil som kallas *layout.php*, vilken innehåller sidans layout.

Layout.php-filen kallas också ibland för den globala templaten och i den lagras HTML-kod som är gemensam för alla templates för att undvika kodrepetition i alla vyfiler. Template-filernas innehåll integreras i layouten eller så kan man också se det som om layouten används för att dekorera template-filerna. I Figur 6 demonstreras hur en template tillsammans med layouten bygger upp den slutliga sidan:



Figur 6. Templaten och layouten bygger tillsammans upp den slutliga sidan. (Potencier & Zaninotto 2011)

Den globala templaten kan helt och hållet anpassas för varje applikation. Man kan lägga till så mycket HTML-kod som behövs. Layouten används oftast för att lagra navigeringsmenyn, webbsidans logo och annan information som hålls oförändrad då man använder applikationen. Man kan även ha mer än en layout och bestämma vilken layout som skall användas för varje specifik handling. (Potencier & Zaninotto s. 101)

2.5.2 Kodfragment

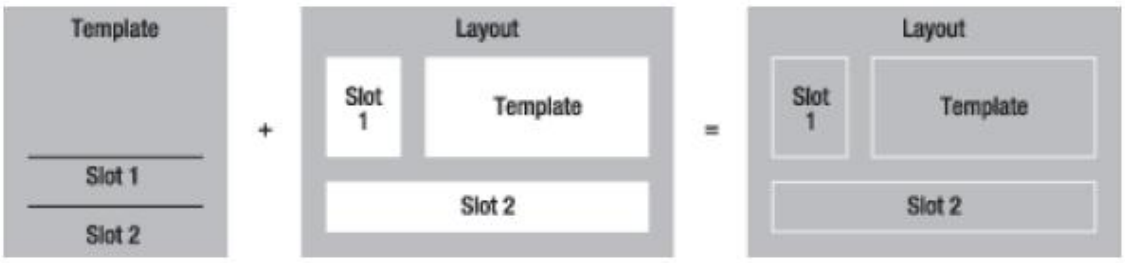
Man kan ofta ha behov av att använda en bit av HTML- eller PHP-kod på flera sidor. För att undvika repetition av dessa fragment erbjuder PHP-biblioteket *include( )*-funktionen, som kan användas för att inkludera en fil med HTML- eller PHP-kod inom en annan. Men bl.a. för att man skall kunna använda olika variabelnamn mellan kodfragmenten och de olika filerna som ska inkludera den samt för att varje kodfragment skulle kunna hanteras skilt för sig av cache-systemet erbjuder Symfony på tre alternativa typer av intelligenta kodfragment för att ersätta *include*:

- Partials.
- Komponenter (eng. components).
- Slots.

Om logiken är enkel, kommer man att endast vilja inkludera en template-fil som har tillgång till visst data, som man skickar till den. För detta ändamål skall man använda en *partial*. Såsom också templates-filerna, hittar man partials-filerna i *templates/*-katalogen, och de innehåller HTML-kod med inbäddad PHP-kod. Ett partial-filnamn börjar alltid med ett understreck (_), och det hjälper att skilja åt template-filer från partial-filer eftersom de befinner sig i samma katalog.

Om logiken är mer komplicerad (till exempel om man behöver komma åt datamodellen och/eller ändra på innehållet beroende på sessionen), kommer man att föredra att separera presentationen från logiken. För detta ändamål skall man använda sig av en *komponent*. På samma sätt som MVC-designmönstret påverkar handlingar och templates, kan man bli tvungen att dela upp en partial i en logik-del och i en presentationsdel. Detta görs då man använder *komponenter*. En *komponent* är som en handling, förutom att den är mycket snabbare. Den logiska delen lagras i en *module/actions/components.class.php*-fil medan presentationen lagras i en *partial*.

Om fragmentet är menat att ersätta en viss del av layouten, där det redan kan finnas standardinnehåll, skall man använda sig av en *slot*. I grund och botten är en slot en platshållare som kan läggas i vilket som helst av vyelementen (i layouten, i en template eller i en partial). Ifyllningen av platshållaren går sedan till lika enkelt som att ge en variabel ett nytt värde. Ifyllningskoden lagras globalt i svaret på förfrågan, så den kan definieras var som helst (i layouten, i en template eller i en partial). Om en layout annars enbart hade en plats för innehåll, dit templaten kommer, erbjuder slots möjligheten att definiera flera sådana platser. (Potencier & Zaninotto s. 103-108) Figur 7 representerar hur sidan kan uppbyggas och layouten kan ifyllas av en template och två slots:



Figur 7. Exempel på användningen av flera slots för att bygga upp en sida. (Potencier & Zaninotto 2011)

2.6 Modell-lagret

Affärslogiken i en webbapplikation är väldigt beroende av applikationens datamodell. Modell-komponenten i Symfony baserar sig på någon av de objekt-relationella kartläggningsmodellerna Propel eller Doctrine. I ett Symfony-projekt kommer man åt data i databasen enbart genom objekt, man är aldrig i direkt kontakt med databasen. På detta sätt upprätthålls en hög abstraktionsnivå och portabilitet av applikationen.

2.6.1 Databaschemat i Symfony

För att skapa dataobjekt-modellen, som Symfony skall använda sig av, måste man först översätta databasens relationsmodell till en objekt-datamodell. ORM-lagret behöver en beskrivning av relationsmodellen för att kunna skapa kartläggningen och denna beskrivning kallas för *schema*. I ett schema definieras tabellerna, deras samband med varandra och deras kolumnegenskaper. Schemafilen heter *schema.yml*, är skriven i YAML-format och finns i projektkatalogen */config/doctrine*. (Potencier & Zaninotto s. 122-123)

2.6.2 Modellklasser

Schemat används för att bygga upp modellklasser av ORM-lagret. Dessa klasser kan skapas automatiskt via kommandoradsgränssnittet med kommandot *doctrine:build-model*. Kommandot startar upp en analys av schemat och genereringen av en basdata-modellfil och två egentliga datamodell-filer per databastabell, i projektkatalogen */lib/model/doctrine*. (Potencier & Zaninotto s. 124)

2.6.3 Modellen på två ställen

Nackdelen med att använda en ORM är att datastrukturen måste definieras på två ställen: en gång för databasen och en gång för objekt-modellen. Lyckligtvis erbjuder Symfony ett verktyg som gör att man kan generera den ena på basen av den andra, och därmed undvika dubbelt arbete och minimera chansen för misstag.

Om man börjar bygga upp sin applikation med att skriva *schema.yml* filen, kan man skapa en SQL förfrågan som skapar databastabellerna direkt från YAML-datamodellen med kommandot: *php symfony doctrine:build-sql*.

Om man å andra sidan börjar bygga upp sin applikation med att skapa databasen först, kan man automatiskt skapa ett schema på basen av databasens tabeller med kommandot: *php symfony doctrine:build-schema*. (Potencier & Zaninotto s. 140)

2.7 Länkar och routing-systemet

2.7.1 Vad är routing?

Routing är en mekanism som omskriver webbadresser (URL-adresser) för att göra dem mer användbara och lättare att förstå. Webbadresser kan fungera endera som serverinstruktioner eller som en del av gränssnittet. I det första och mer traditionella fallet bär de enbart information från bläddraren till servern för att aktivera en viss handling önskad av användaren.

Huvudidén med routing är att se webbadresser som en del av gränssnittet. Då kan applikationen utforma webbadresserna så att de kan ge information åt användaren, och användaren kan använda webbadresserna för att komma åt olika delar av applikationen. Detta är möjligt i Symfony-applikationer eftersom den webbadress som visas för användaren inte är kopplad serverinstruktionen, som behövs för att utföra användarens begäran. Istället är webbadressen kopplad till den begärda delen av applikationen och den kan se ut hur som helst. (Potencier & Zaninotto s. 142-143)

2.7.2 Hur fungerar det?

Symfony separerar den externa webbadressen och dess interna URI (Uniform Resource Identifier) och korrespondensen mellan dessa görs av routing-systemet. Det betyder att en webbadress visas för användaren medan en annan tolkas av webbservern. Routing-systemet använder sig av en speciell konfigureringsfil, *routing.yml*, där man kan definiera routing-regler.

Varje begäran som sänds till en Symfony-applikation analyseras först av routing-systemet. Routing-systemet söker sedan efter en motsvarighet bland olika mönster som är definierade i routing-reglerna. (Potencier & Zaninotto s. 144-145)

2.7.3 Hjälpfunktioner för länkar

På grund av routing-systemet borde man alltid använda sig av hjälpfunktioner för länkar istället för <a>-taggar i applikationerna.

Den första av hjälpfunktionerna för länkar är *link_to( )*. Funktionen returnerar en XHTML-kompatibel hyperlänk och den förväntar sig två parametrar: elementet som skall kunna klickas på och den interna URI:n av resursen som den pekar till. Om man istället för en hyperlänk vill visa en knapp skall man istället använda sig av hjälpfunktionen *button_to( )*. (Potencier & Zaninotto s. 147-148) Figur 8 visar exempel på hur man kan kalla på dessa funktioner och vad den motsvarande webbadressen resulterar i:

```
// Hyperlink on a string
<?php echo link_to('my article', 'article/read?title=Finance_in_France') ?>
 => <a href="/routed/url/to/Finance_in_France">my article</a>

// Hyperlink on an image
<?php echo link_to(image_tag('read.gif'), 'article/
read?title=Finance_in_France') ?>
 => <a href="/routed/url/to/Finance_in_France"><img
src="http://www.symfony-project.org/images/read.gif" /></a>

// Button tag
<?php echo button_to('my article', 'article/read?title=Finance_in_France')
?>
 => <input value="my article"
type="button"onclick="document.location.href='/routed/url/to/
Finance_in_France';" />
```

Figur 8. Exempel på användning av hjälpfunktioner för länkar. (Potencier & Zaninotto 2010)

2.8 Formulär

Med formulär avses alla sidor eller delar av sidor där användaren kommer åt att mata in nytt data till databasen eller att ändra på existerande data i databasen.

2.8.1 Att visa ett formulär

I Symfony är ett formulär ett objekt som definieras i handlingen och förs över till templaten. För att kunna visa ett formulär måste först alla dess fält definieras. Symfony använder begreppet ***ò****widgets****ò*** för dessa fält. Man kan skapa ett nytt formulärobjekt av klassen *sfForm* i handlingen genom att kalla på *$this->form = new sfForm( )* och ställa in formulärets fält genom att kalla på *$this->form->setWidgets( )* med de önskade widget-objekten som argument. För att rendera formulärets alla fält med sina titlar räcker det att man kallar på *echo $form* från template-filen. (Potencier & Zaninotto s. 159-160)

2.8.2 Formulär-widgets

Det finns olika widgets tillgängliga för olika typer av inmatning. Varje typ av inmatningsfält representeras av ett eget slags widget-objekt. Till exempel om man vill rendera ett typiskt textinmatningsfält skall man använda sig av ett objekt av widget-klassen sfWidgetFormInput och om man vill rendera ett fält för inmatning av lösenord, där den inmatade texten ersätts av stjärnor, skall man använda sig av ett objekt av widget-klassen *sfWidgetFormInputPassword*.

2.8.3 Hantering och validering av formulär

I praktiken handlar formulärhantering om mycket mer än att enbart ta emot inmatade värden av användaren. För hantering av de flesta formulär måste applikationen gå igenom fyra steg:

- Kontrollera att data överensstämmer med en uppsättning av fördefinierade regler (obligatoriska fält, formatet av e-postadresser, etc.)
- Eventuellt omforma en del av inmatad data för att göra det förståeligt (borttagning av mellanslag, konvertering av datum till PHP format, etc.)
- Om data är ogiltigt, måste formuläret visas igen med felmeddelanden där de är aktuella.
- Om data är korrekt, måste data hanteras och användaren dirigeras vidare till någon annan handling.

Symfony erbjuder ett automatiskt sätt att validera inmatad data mot en uppsättning av fördefinierade regler. Först måste man definiera en uppsättning av regler för varje inmatningsfält med hjälp av *$this->form->setValidators( )*. Sedan när formuläret hanteras måste inmatade datat bindas ihop med formuläret med hjälp av *$this->form->bind( )* och till sist måste man begära att formuläret kollar att data är korrekt genom att kontrollera returvärdet av *$this->form->isValid( )*. Funktionen *isValid( )* fungerar också som en säkerhetskontroll. (Potencier & Zaninotto s. 170-172)

2.9 Internationalisering

Symfony erbjuder flera egenskaper för att underlätta internationalisering (I18n) och lokalisering (L10n) av webbapplikationer. Bland de viktigaste och lättaste att använda av dessa är hjälpfunktionen för översättning *__( )*. Hjälpfunktionen används i template-filerna och allt den kräver är att man skriver ut fraserna som skall kunna översättas med hjälp av den. Till exempel om man skriva òVälkommen till applikationenò och vill kunna visa texten på ett annat språk då användaren byter språk, måste man skriva ut texten på följande sätt: *<?php echo __(ôVälkommen till applikationenô) ?>*.

Varje gång funktionen *__( )* kallas, letar Symfony efter en översättning på argumentet från översättningsfilen som motsvarar användarens kultur. Översättningsfilerna är skrivna i XLIFF-format (XML Localization Interchange File Format) och sparade i applikationens *i18n/*-katalog. För att exempelvis skapa en översättningsfil för språket engelska måste man via kommandoradsgränssnittet kalla på kommandot *php symfony i18n:extract --auto-save frontend en*. (Potencier & Zaninotto s. 219-228)

2.10 Admin-generatorn

Webbapplikationer grundar sig ofta på data som är lagrat i en databas och på ett gränssnitt för att komma åt data. Med hjälp av Symfony kan man automatisera den repetitiva uppgiften att skapa moduler för att manipulera data, som grundar sig på Doctrine-objekt. Om objektmodellen är ordentligt definierad, kan Symfony även generera en hel webbplats administration automatiskt med hjälp av admin-generatorn.

2.10.1 Kodgenerering baserad på modellen

I en webbapplikation kan operationer för dataåtkomst kategoriseras som en av följande:

- Skapande av data
- Hämtande av data
- Uppdatering av data
- Borttagning av data

Dessa operationer är så vanliga att man använder en särskild förkortning för dem: CRUD, som kommer från de engelska termerna Create, Retrieve, Update och Delete. Många webbsidor kan reduceras till en av dem. Exempelvis är upplistningen över de senaste inläggen i en forum-applikation ett exempel på hämtande av data, medan sidan för att svara på ett inlägg är ett exempel på skapande av data.

De grundläggande handlingarna och templates som realiserar CRUD-operationer för en särskild tabell skapas upprepade gånger i webbapplikationer. I Symfony-projekt innehåller modell-lagret tillräckligt med information för att automatiskt kunna generera färdig kod för att möjliggöra CRUD-operationer för alla databasens tabeller. (Potencier & Zaninotto s. 231)

2.10.2 Administrationsmoduler

Symfony kan automatiskt generera fullständiga moduler som baserar sig på modellklass-definitionerna i *schema.yml*-filen. Man kan skapa en hel webbplats-administration genom att enbart använda sig av genererade administrationsmoduler. Man kan till och med skapa en webbplats som endast består av administrationsmoduler.

Administrationsmodulerna tolkar modellen genom en konfigurationsfil som kallas *generator.yml*, som kan ändras för att utvidga alla genererade komponenter och modulens utseende och känsla. Man kan åsidosätta genererade handlingar och templates, för att kunna integrera egen funktionalitet i modulerna, genom att skapa dem inom modulen med samma namn men med annorlunda innehåll än i de genererade.

Administrationsmodulerna måste skapas skilt för varje modell. En modul som baserar sig på Doctrine-modellen kan genereras genom att man använder sig av kommandoradsgränssnittet och *doctrine:generate-admin*-kommandot.

Om man efter skapandet av en administrationsmodul ser på innehållet i modulkatalogen ser man inte någon genererad kod, utan katalogen är tom. Detta beror på att koden är skapad först då det finns behov för det och om man vill se hur koden ser ut för den genererade modulen måste man först samverka med den i bläddraren och sedan kontrollera innehållet i projektets *cache/*-katalog. (Potencier & Zaninotto 2010 s. 233-235)

3 UTVECKLINGSVERKTYG

3.1 NetBeans

För programmering av programvarans PHP-, HTML-, CSS- och AJAX-kod användes utvecklingsmiljön NetBeans. NetBeans IDE (Integrated Development Environment) är en prisbelönt integrerad utvecklingsmiljö för Windows, Mac, Linux och Solaris. NetBeans-projektet består av en öppen källkods IDE och en applikationsplattform som gör det möjligt för utvecklare att snabbt skapa webb-, enterprise-, skrivbords- och mobilapplikationer med Java-plattformen, samt med PHP, JavaScript och Ajax, Groovy och Grails, och C/C++. (NetBeans 2011)

NetBeans-projektet stöds av en aktiv utvecklings-community och erbjuder omfattande dokumentation och utbildningsresurser samt ett brett urval av tredje parters plugins. I detta arbete användes versionerna 6.8 och 7.0 av NetBeans och Symfony plug-in.

3.2 Versionshantering

Versionshantering innebär att tidigare versioner av en sida, dokument, källkodsfiler eller program kan återskapas, och ändringar gjorda i dessa tidigare versioner kan spåras. Möjlighet till parallell utveckling, exempelvis rättning av äldre versioner parallellt med vidareutveckling av nya, är också väsentlig. Detta är i synnerhet användbart när många

personer arbetar med samma sak. Vanligen används ett speciellt program, exempelvis CVS (Concurrent Versions System), men moderna utvecklingsverktyg har ofta inbyggt stöd för versionshantering. (Görling 2009 s. 117-122)

I början av projektet användes TortoiseSVN för versionshantering, men i ett senare skede bestämde uppdragsgivaren att versionshanteringssystemet Git skulle användas i stället.

3.2.1 TortoiseSVN

TortoiseSVN är ett lättanvänt SCM (Software Configuration Management)-källkodskontrollprogram för Microsoft Windows och fungerar som en fristående klient. Klienten realiseras som en utvidgning av Windows shell, vilket gör den sömlöst integrerad i Windows Explorer.

Eftersom TortoiseSVN inte är en integration för ett specifikt IDE, kan man använda den med vilket utvecklingsverktyg som helst. (TortoiseSVN 2011)

3.2.2 Git ï Fast Version Control System

Git är ett distribuerat versionshanteringssystem som är gratis och har öppen källkod. Git är planerat för att hantera allt från små till mycket stora projekt med snabbhet och effektivitet.

Varje Git-klon (eng. *clone*) är ett fullständigt arkiv med komplett historik och full funktionalitet för versionspårning, som inte är beroende av nätverksanslutning eller anslutning till en central server. Förgrening och sammanslagning går snabbt och lätt att göra med hjälp av Git. (Git 2011)

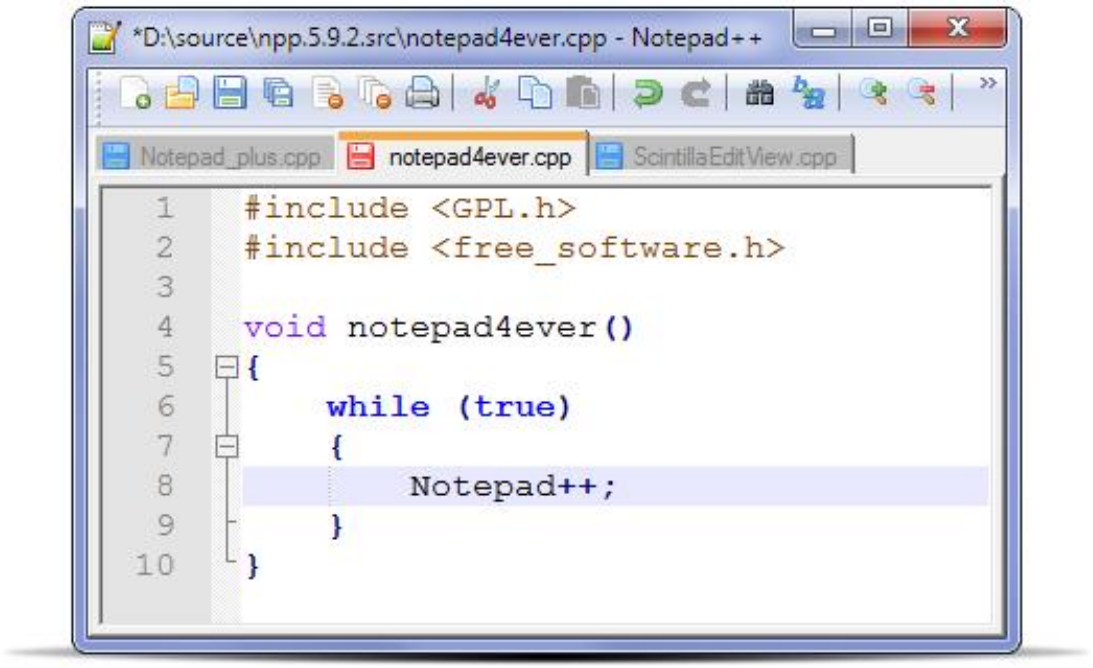
3.3 Notepad++

Notepad++ är ett gratis textredigeringsprogram med öppen källkod, som stöder flera olika programmeringsspråk och är en utmärk ersättare av Windows-programmet Anteckningar. Programmet körs i Microsoft Windows-miljön. I utvecklingsskedet

användes Notepad++ för mindre ändringar och för att se på innehållet av filer snabbt, utan att behöva starta upp utvecklingsmiljön NetBeans.

Notepad++ baserar sig på den kraftfulla redigeringsverktygskomponenten Scintilla, är skriven i C++ och använder sig av Win32 API och STL (Standard Template Library), vilket leder till en högre exekveringshastighet och till mindre programstorlek. Utvecklarna av Notepad++ strävar till att genom att optimera så många rutiner som möjligt, utan att minska på användbarhet, försöka minska på världens koldioxidutsläpp. Deras idé går ut på att när man använder mindre CPU-kraft, kan datorn varva ner och minska på strömförbrukningen, vilket resulterar i en grönare miljö. (Notepad++ 2011)

Figur 9 visar hur textredigeringsprogrammet Notepad++ ser ut.



Figur 9. Textredigeringsprogrammet Notepad++. (Notepad++ 2011)

3.4 Firebug

Firebug är ett webbaserat utvecklingsverktyg som möjliggör felsökning, redigering och övervakning av CSS, HTML, DOM och JavaScript på vilken webbsida som helst och erbjuder även andra verktyg för webbutveckling. Verktyget har också en JavaScript-

konsol för att logga fel och kontrollera värden, samt en "Net"-funktion, som övervakar den tid i millisekunder det tar att köra skript och ladda bilder på sidan.

Firebug är gratis, har öppen källkod och är licensierad under BSD-licensen. Firebug skrevs ursprungligen i januari 2006 av Joe Hewitt, en av de ursprungliga skaparna av Firefox-bläddraren. Firebug har en arbetsgrupp som övervakar utvecklingen av den öppna källkoden och utvidgandet av verktyget. Firebug har två stora realiseringar, ett tilläggsprogram för Mozilla Firefox och Firebug Lite, som kan integreras i webbsidan. Från och med november 2010 har cirka 3 miljoner användare installerat tilläggsprogrammet Firebug. (Firebug 2012)

Förutom felsökning på webbplatser, är Firebug ett användbart verktyg för testning av webbsäkerhet och av webbplatsers prestanda.

4 UTFÖRANDET AV PRAKTISKA DELEN

Den praktiska delen av examensarbetet gick ut på att planera och utveckla en utvidgad databasdesign och Inköpsverktyget för uppdragsgivarens delvis existerande nya webbapplikation. I detta kapitel förklaras hur de nya egenskaperna planerades, hur datamodellen är uppbyggd och vilka de skapade centrala egenskaperna är. För att erbjuda en bra överblick av funktionaliteten och för att inte avslöja alltför detaljerad information om programvaran, är exempelkoden av datasäkerhetsskäl aningen förenklad och variablers och funktioners namn är förändrade.

4.1 Konceptuell modell

Den centrala funktionaliteten i Inköpsverktyget kan beskrivas med hjälp av fyra centrala begrepp:

- Beställningar
- Leverantörer
- Mottagande av beställningar och varor
- Beställningsförslag

4.1.1 Beställningar

Inköpsbeställningar omfattar beställning av varor från en specifik leverantör. För att underlätta läsandet kommer man hädanefter att hänvisa till inköpsbeställningar som beställningar. En beställning kan ha fyra olika status: sparad, beställd, delvis mottagen eller helt och hållet mottagen. Varorna som hör till en beställning beställs av en leverantör genom att sända beställningen till leverantören via e-post. Varje beställningsrad i databasen innehåller också detaljerad information om beställningen, såsom tid för beställningen, vem som skapat beställningen, önskad tid för leverans och beställningsnummer. Alla varor som ingår i en beställning har samband med beställningen och informationen om hur många exemplar av varje vara beställs är den viktigaste informationen gällande varje varurad i databasen vid beställningsskedet. Varje beställning är dessutom bunden till en affärsenhet inom en detaljhandelskedja/affärskedja.

4.1.2 Leverantörer

Varje lagervara, dvs. varje fysiska vara som kan ha ett lagersaldo, bör vara bunden till en viss leverantör av vilken man kan beställa den ifrågavarande varan för återförsäljning. En vara kan också vara bunden till flera olika leverantörer samtidigt, men av dessa är en förvald. Varje leverantörrad i databasen innehåller basdata om leverantören, såsom till exempel namn, adress, e-postadress, kontaktpersonens uppgifter och en flagga som bestämmer om det är tillåtet att göra beställningar av leverantören.

4.1.3 Mottagande av beställningar och varor

En vara kan mottas endera oberoende av eller som en del av en beställning. Om en vara tas emot som en del av en beställning måste man kunna associera beställningen, varan, den mottagna mängden, tiden för mottagning och vem som tog emot varan, med mottagningsraden. Om varan tas emot oberoende av en beställning, behöver raden inte innehålla någon information om beställningar.

4.1.4 Beställningsförslag

Varje lagervara bör ha ett aktuellt lagersaldo, ett maksimalt lagersaldo och en larmgräns i databasen. Om det aktuella lagersaldot underskrider larmgränsen bör den ifrågavarande varan hamna med på ett leverantörspecifikt beställningsförslag. Mängden som kommer att beställas av den ifrågavarande varan utgörs av skillnaden mellan det aktuella lagersaldot och det maksimala lagersaldot. Beställningsförslag skapas av en skild funktion som måste triggas av användaren.

4.2 Utvecklingsprocessen

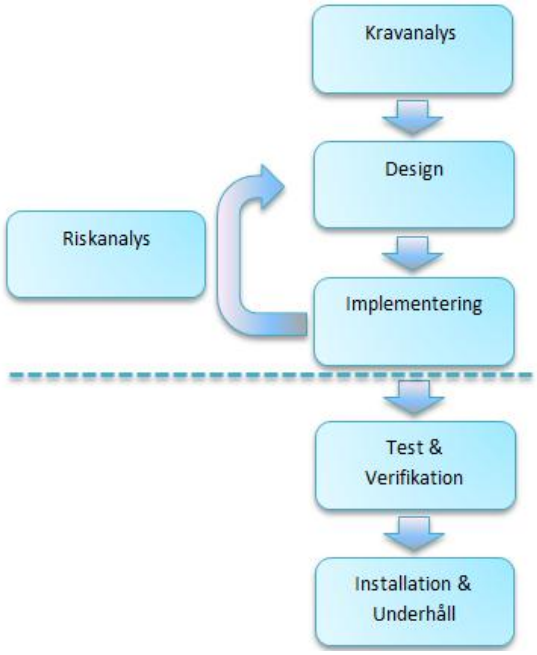
Vattenfallsmodellen är en av de äldre modellerna för utveckling av programvara. Modellen etablerades kring 1970 och användes länge som huvudmall vid systemutveckling. Den är sekventiell och innehåller fem separata steg:

- **Kravanalys och specifikation** - I det första steget studeras och specificeras kraven.
- **Design -** I det andra steget planeras applikationens grundläggande arkitektur utifrån specifikationen.
- **Implementering -** I det tredje steget skrivs själva programkoden utifrån designen.
- **Test och verifikation -** I det fjärde steget testas att programmet motsvarar de krav som ställts i början av projektet.
- **Installation/underhåll -** I det femte och slutliga steget överlämnas produkten till kunden och övergår i en förvaltnings-/underhållsfas.

Ett steg utförs i taget och mellan varje steg genomförs granskningsmöten för att kontrollera om det är dags att gå vidare till nästa steg. (Görling 2009 s. 59-60)

Den iterativa modellen eller spiralmodellen är den andra huvudtypen av utvecklingsmodeller. Den grundar sig på att utvecklingsarbetet går runt i en spiralform där alla stegen från Vattenfallsmodellen passeras flera gånger under projektets utveckling istället för att gå en gång sekventiellt genom processen. (Görling 2009 s. 62)

Modellen som användes för att utveckla detta projekt var en blandning av Vattenfallsmodellen och den iterativa modellen. Först uppställdes en kravspecifikation, som godkändes av uppdragsgivaren och efter det påbörjades planeringen av databasen. Efter att datamodellen började få form påbörjades planeringen av programmet, utgående från datamodellen och kravspecifikationen. Följande steg var implementering och efter att mindre helheter var realiserade och testade gjordes en riskanalys och vid behov ändrades designen. Själva kodandet inleddes med att först skriva pseudokod, som beskrev vad programkoden skulle åstadkomma, och efter det skrevs den egentliga programkoden. Figur 10 visar en grafisk presentation av utvecklingsmodellen som användes.



Figur 10. Grafisk presentation av utvecklingsmodellen.

De två sista stegen i modellen kommer att förverkligas först i ett senare skede och därför finns linjen i Figur 10. Testandet kommer att utföras genom att skriva user-stories på basen av kravspecifikationen och motsvarande funktionella tester för dem.

4.2.1 Kravspecifikationen

För att komma igång med kravspecifikationen hade vi först ett möte tillsammans med uppdragsgivaren där vi tillsammans skapade riktlinjer för huvudegenskaperna i Inköpsverktyget. Huvudkravet för projektet var att när Inköpsverktyget är färdigt, skall man kunna utföra liknande tjänster som i uppdragsgivarens nuvarande program, men

databasdesignen skulle förbättras. Det ibrukvarande programmet studerades grundligt för att få ett grepp om helheten i dess tjänster.

Efter att ha kommit fram till några riktlinjer om egenskaperna gick vi tillsammans med uppdragsgivaren igenom vad som dittills hade implementerats i RCMS och hur de nya egenskaperna skulle förhålla sig till befintlig funktionalitet. Val av ramverk och plattform gjordes av uppdragsgivaren och en stor del av RCMS redan var implementerad då jag började planera Inköpsverktyget.

Kravspecifikationen består först av allt av en introduktion, som kortfattat förklarar vad Inköpsverktyget är och vad det kan användas för. Efter det följer en detaljerad beskrivning av de fyra helheterna som tas upp i kapitel 4.1: beställningar, leverantörer, mottagande av beställningar och varor samt beställningsförslag. Förutom detta innehåller kravspecifikationen en beskrivning av datamodellen, hur partier av varor skall behandlas i samband med beställningar och en uppspjälkt tidsestimering. Beställningsförslag och behandling av partier av varor ingår i kravspecifikationen men kommer att förverkligas först i ett senare skede av uppdragsgivarens andra resurser.

När tidsestimeringen för projektet började kartläggas var det viktigt att komma fram till en så realistisk estimering som möjligt. Gällande tidsestimering skriver Görling (2009 s.99) om en vanlig tumregel som kallas *The doubling rule*, vilken innebär att man alltid bör dubbla de estimeringar som görs. Denna regel följdes inte ordagrant, men den erbjöd vissa behändiga riktlinjer.

4.3 Datamodellen

Att få planerat och uppbyggt en fungerande och användbar datamodell, som kan utvidgas vid behov var en de viktigaste delarna i detta projekt. Hela systemets effektivitet, pålitlighet och utvidgningsbarhet är beroende av den.

Datamodellen är en utvidgning av en existerande databas och grundar sig på ARTS (The Association for Retail Technology Standards). ARTS är en del av National Retail Federation och en internationell medlemsorganisation inriktad på att minska på

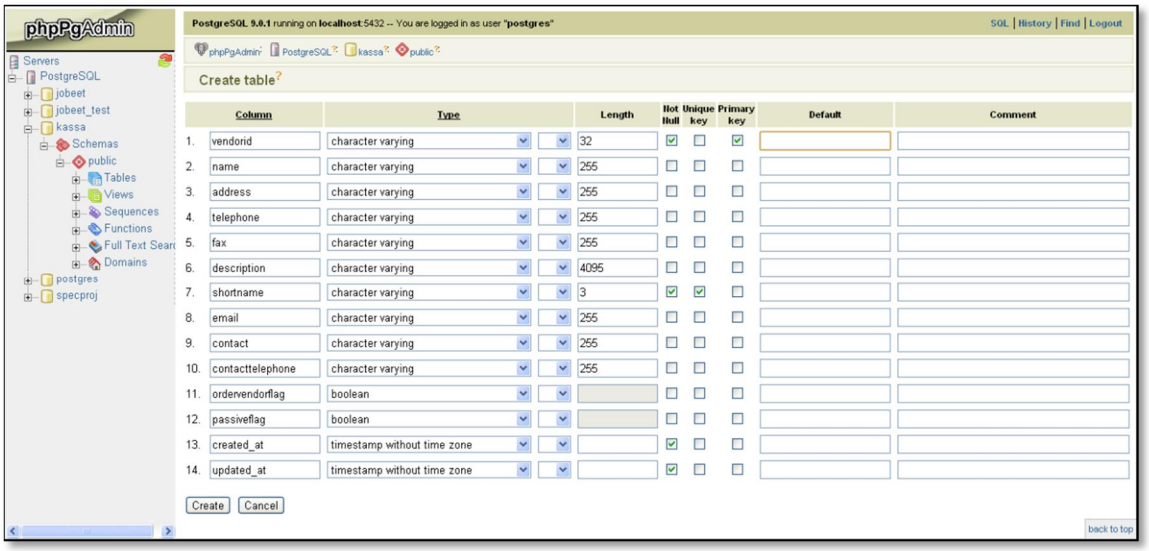
teknikkostnaderna genom standarder. Sedan 1993 har ARTS levererat exklusiva applikationsstandarder, -produkter och -program för detaljhandeln, vilka är avsedda att främja innovation och öka effektiviteten hos återförsäljare. Medlemskap i ARTS är öppet för alla medlemmar av det internationella tekniksamfundet: återförsäljare från alla industrisegment, applikationsutvecklare och hårdvarutillverkare (National Retail Federation 2011).

Allt data som sparas i databasen, förblir också i databasen. Ingenting raderas från databasen, utan data som man inte längre vill att användaren skall komma åt, passiveras med hjälp av en flagga i databasen. Alla rader i databasens alla tabeller har en *passive*-flagga, som kan ha värdet 1 eller 0. Orsaken till denna lösning är att man senare skall kunna ha tillgång till alla händelser i databasen för eventuell felsökning och/eller redovisning.

Som en regel för projektet har också fastslagits att inga tabeller får använda sig av automatiskt inkrementerande identifikationsnumror, utan varje rad bör identifieras av ett universellt unikt identifikationsnummer UUID (Universally Unique Identifier). En UUID är en16-bytes lång nummerserie som representeras av 32 hexadecimala siffror. Detta innebär att det finns 3×10^{38} olika teoretiska kombinationer, så sannolikheten att det förekommer två rader med samma identifikationsnummer i en databas är väldigt liten. Men en UUID är inte garanterat unik, utan snarare praktiskt taget unik. (Wikipedia 2011c)

4.3.1 Tabeller

Till den existerande databasen skapades sju nya tabeller manuellt med hjälp av phpPgAdmin, som är ett bläddrarbaserat administrativt gränssnitt för PostgreSQL-databaser. I Figur 11 presenteras gränssnittet för skapande av nya tabeller:



Figur 11. Skapandet av en ny tabell med phpPgAdmin.

Den existerande tabellen *item*, som innehåller data om varorna, användes för många nya databassamband. De nya tabellerna är:

- **PURCHASEORDER**, som innehåller data om beställningarna och har samband med tabellerna *purchaseorderlineitem*, *vendor*, *businessunit* och *currency*.
- **PURCHASEORDERLINEITEM**, som innehåller data om varje beställd vara och har samband med tabellerna *purchaseorder*, *item* och *receiveditem*.
- **RECEIVEDITEM**, som innehåller data om mottagna varor och måste vara en egen tabell eftersom varor kan också tas emot oberoende av beställningar eller så kan en vara mottas i flera omgångar och vid varje omgång måste tidpunkt, mottagaren och den mottagna mängden registreras. Tabellen har samband med tabellerna *item*, *purchaseorderlineitem* och *operator*.
- **VENDOR**, som innehåller data om leverantörer och har samband med tabellerna *purchaseorder* och *vendoritem*.
- **VENDORITEM**, som innehåller data om varor som är kopplade till leverantörer. Tabellen fungerar som sammanbindningstabell mellan tabellerna *vendor* och *item*.
- **ITEMPURCHASEPRICE**, som innehåller data om inköpspriset av varor och är har samband med *item*-tabellen och det kommer i ett senare skede också att läggas till samband med *vendor*-tabellen för att varor skall kunna köpas in av olika leverantörer till olika pris.

- **PURCHASEORDERPROPOSAL**, som innehåller data om beställningsförslag och har samband med tabellerna *item* och *businessunit*.

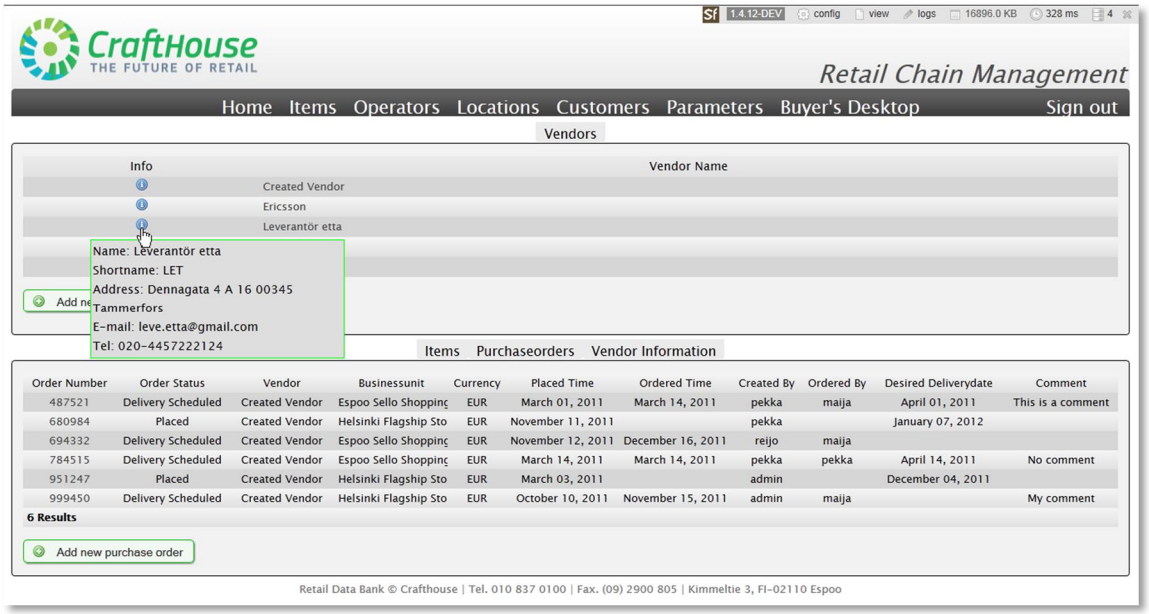
Se Bilaga för en grafisk representation av de nya databaserna och deras kopplingar.

4.3.2 Schema

Eftersom databastabellerna skapades direkt till databasen, kunde en motsvarande schema-fil skapas genom att köra kommandot: *php symfony doctrine:build-schema* via kommandoradsgränssnittet. För att detta skulle vara möjligt måste man också ha databasen konfigurerad rätt i *databases.yml* filen. Jag fick en färdigt konfigurerad konfigureringsfil för databasen, men man kan få filens innehåll att motsvara sin egna databas genom att enbart köra kommandot: *php symfony configure:database "pgsql:host=localhost;dbname=mydatabase" root mYsEcret* med korrekta värden.

4.4 Inköpsverktyget - programvaran

Inköpsverktyget i RCMS är uppbyggt i sin helhet genom att utnyttja Symfony-ramverkets admin-generator. Grundvyn i Inköpsverktyget byggs upp av en lista på leverantörer varav någon leverantör alltid är vald. Tillsammans med den valda leverantören visas endera dess varor, beställningar eller basinformation. Utseendemässigt har tanken varit att val mellan dessa görs genom att byta flik på sidan. I Figur 12 kan man se hur grundvyn av Inköpsverktyget ser ut i bläddraren.

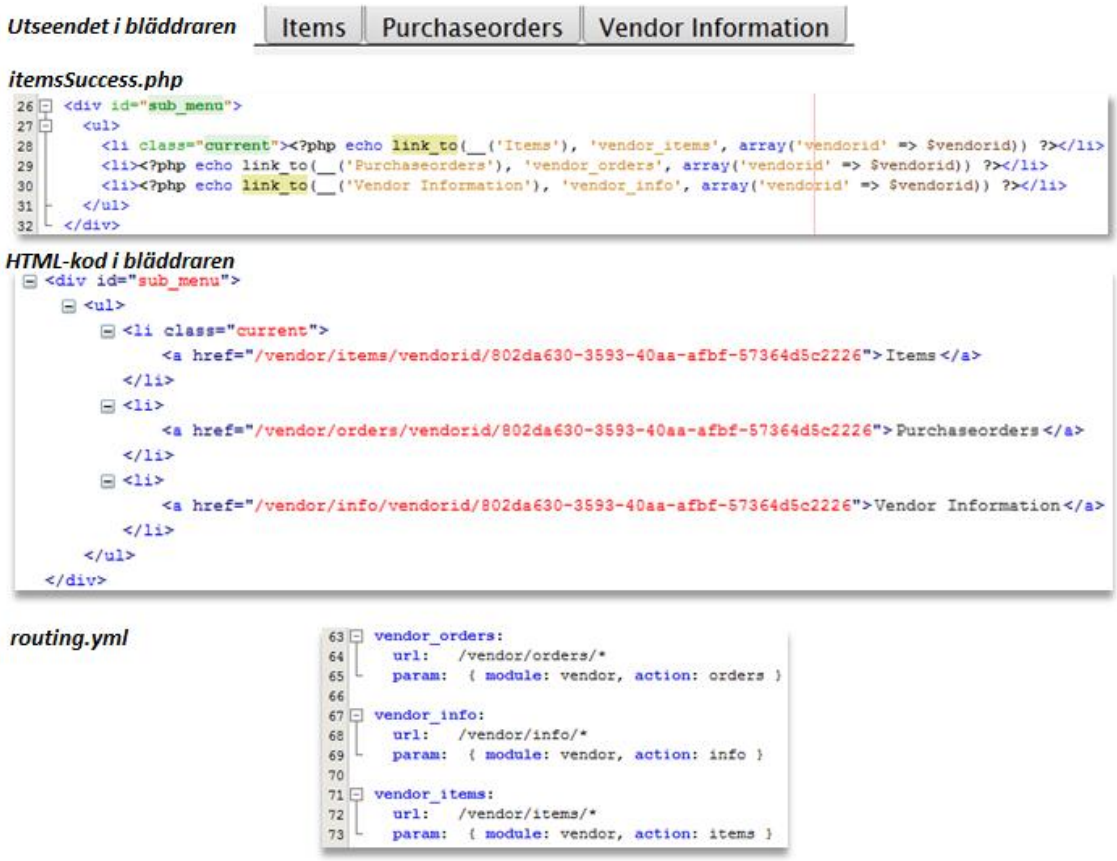


Figur 12. Grundvyn av Inköpsverktyget i RCMS med den valda leverantörens beställningar listade.

För att snabbt kunna se en översikt av leverantörens grundinformation har det realiserats en tooltip-ruta, som kommer fram då man lägger musen över informationssymbolen. Funktionaliteten för detta är realiserad med JavaScript.

4.4.1 Routing

I Inköpsverktyget används inga absoluta hyperlänkar för navigering mellan sidorna. Alla länkar som i HTML-kod ser ut att peka till en viss adress är egentligen utformade av ramverket efter att man med PHP-kod har kallat på ramverkets hjälpfunktioner med önskad metod och handling som parameter. I bläddrarens adressfält visas sedan en adress medan ramverket egentligen använder värdena i *routing.yml* för att se till vilken modul och handling den skall vidarebefordra användaren. I Figur 13 kan man se sammanbandet mellan utseendet i bläddraren, den blandade HTML- och PHP-koden, hur bläddraren har skrivit om PHP-koden till ren HTML-kod och hur motsvarande länkar representeras i routing-tabellen:



Figur 13. Exempel på routing från inköpsverktygets undermeny.

4.4.2 Genvägar inom applikationen

För att förbättra programvarans användbarhet har genvägar inom applikationen tagits i bruk. Låt oss säga att man exempelvis vill göra en beställning på en vara men man vill innan det ändra namnet på varan. För att inte först behöva navigera till sidan med alla varor, sedan söka fram den ifrågavarande varan och till sist öppna varans redigeringssida och där göra ändringar har istället alla listningar av varor gjorts på det sättet att där varans namn skrivs ut, fungerar utskriften egentligen som en direkt länk till varans redigeringssida. Med listning menas att data från databasen endast visas, utan att användaren direkt kommer åt att lägga till eller att göra ändringar i det.

Länkarna skapas i vyfilerna med hjälp av ramverkets hjälpfunktion *link_to( )*. Genom att ge ett motsvarande värde för rutten till redigeringssidan i routing-tabellen och varans *itemid* som argument för funktionen, kan ramverket vidarebefordra användaren till rätt

sida. I Figur 14 kan man se hur funktionen används och hur argumentvärdena utformas samt hur PHP-koden ser ut då den har omformats till HTML-kod av ramverket och webbservern.

```
<?php echo link_to($item->getItem()->getName(), 'item_edit', array('itemid' => $item->getItemid())) ?>
```

```
<a href="/items/85a85c73-7f97-44bc-b793-7b66679e7c48/edit">Itemname</a>
```

Figur 14. Länk till varans redigeringssida i PHP-kod och motsvarande HTML-kod.

4.4.3 Beredskap för flerspråkighet

Alla statiska texter i Inköpsverktygets template-filer är utskrivna genom att utnyttja ramverkets __( )-hjälpfunktion. När man kör kommandot för att skapa översättnings-filer via kommandoradsgränssnittet kommer alla strängar, som är utskrivna på detta sätt, att automatiskt bilda ett eget block i översättningsfilerna. På detta sätt är det i ett senare skede väldigt lätt att ge applikationen stöd för flera språk.

4.5 Användargränssnittet

För varje tabell i databasen skapades en egen modul i applikationen. Modulerna skapades med kommandoradsverktyget och *vendor*-modulen skapades exempelvis med kommandot i Figur 15:

```
C:\windows\system32\cmd.exe
C:\rcms\jani_branch>php symfony --theme="customTheme" doctrine:generate-admin kassapp Vendor --module=vendor_
```

Figur 15. Exempel på kommando för skapande av modul.

Genom att använda *doctrine:generate-admin*-kommandot skapar ramverket automatiskt fullt fungerande grundfunktionalitet för den angivna modellklassen, så att man kan lista, lägga till, editera och ta bort värden från databastabellen. Parametern *--theme=**ò**customTheme**ò*** berättar för ramverket att det skall använda ett fördefinierat tema för att bestämma utseendet för de genererade vyfilerna.

Genom att skapa egna funktioner, med samma namn som de genererade, i filen */apps/kassapp/modules/modulNamn/actions/actions.class.php* och egna vyfiler i katalogen */apps/kassapp/modules/modulNamn/templates/* kan man ersätta och/eller

utvidga den automatiska funktionaliteten skapad av ramverket, med en mer avancerad och självutvecklad funktionalitet för modulerna.

4.5.1 En modul inom en annan

Beställningarna och varorna som hör till beställningarna finns i egna tabeller i databasen och därmed också i egna moduler. Modulerna i fråga är *purchaseorder* och *purchaseorderlineitem*.

Då man skall lägga till varor till en beställning anropas *purchaseorderlineitem*-modulens *executeNew()*-funktion med beställningens id som parameter, så att den valda varan skall kunna kopplas med beställningen i fråga. För att spara id:n i en lokal variabel kallas funktionen *$request->getParameter('orderid')* och efter det skapas ett nytt objekt i klassen Purchaseorderlineitem med *$purchaseorderlineitem = new Purchaseorderlineitem();*. För att koppla objektet med beställningen kallas *$purchaseorderlineitem->setPurchaseorderid($request->getParameter('orderid'))*.

Ett objekt i formulärklassen för tilläggning av varor med en förutbestämd beställnings-id skapas av *$this->form = new PurchaseorderlineitemForm($purchaseorderlineitem);* genom att ge det tidigare skapade objektet som parameter. Den egentliga utskriften av formuläret sker i vyfilen genom att kalla på *<?php echo $form ?>*.

```
16    public function executeNew(sfWebRequest $request)
17    {
18      //Check if parameter is given
19      if ($this->hasRequestParameter('orderid'))
20      {
21        // Save the parameter into a variable
22        $this->purchaseorderid = $request->getParameter('orderid');
23
24        // Create an instance of Purchaseorderlineitem
25        $purchaseorderlineitem = new Purchaseorderlineitem();
26
27        // Set the purchaseorder id for the instance
28        $purchaseorderlineitem->setPurchaseorderid($this->purchaseorderid);
29
30        $this->form = new PurchaseorderlineitemForm($purchaseorderlineitem);
31      }else
32      {
33        // If parameter is not given, redirect to error page
34        $this->forward404();
35      }
36    }
```

Figur 16. Funktionen executeNew() i apps/kassapp/modules/purchaseorderlineitem/actions/actions.class.php

För att på beställningssidan lista upp alla varor som hör till den ifrågavarande beställningen måste man skapa nya accessmetoder till tabellerna. De nya metoderna heter *getItems()* i båda fallen och de skapas i filerna */lib/model/doctrine/Purchaseorder.class.php* och */lib/model/doctrine/PurchaseorderlineitemTable.class.php*. Den första metoden skapar en databasförfrågan som returnerar beställningar för en viss id och kallar sedan på den andra metoden med förfrågan som parameter. Den andra metoden i sin tur returnerar alla resultat från tabellen, som motsvarar beställningen i parametern. I Figur 17 kan man se metodernas uppbyggnad:

Purchaseorder.class.php

```
20    public function getItems()
21    {
22      $q = Doctrine_Query::create()
23        ->from('Purchaseorderlineitem p')
24        ->where('p.purchaseorderid = ?', $this->getPurchaseorderid());
25
26      return Doctrine_Core::getTable('Purchaseorderlineitem')->getItems($q);
27    }
```

PurchaseorderlineitemTable.class.php

```
15    public function getItems(Doctrine_Query $q = null)
16    {
17      if (is_null($q))
18      {
19        $q = Doctrine_Query::create()
20          ->from('Purchaseorderlineitem p');
21      }
22      $q->addOrderBy('p.rownumber DESC');
23
24      return $q->execute();
25    }
```

Figur 17. getItems()-funktionerna.

Den övre funktionen kallas från beställningsmodulens formulärfil */apps/kassapp/modules/purchaseorder/templates/_form.php* och man kan till exempel skriva ut varje varas namn och den beställda mängden enligt koden i Figur 18:

```
19  <?php if (($count = $purchaseorder->getItems()->Count()) > 0): ?>
20  <h2><?php echo __('Ordered items') ?></h2>
21  <table>
22    <th><?php echo __('Itme name') ?></th>
23    <th><?php echo __('Ordered units') ?></th>
24    <?php foreach ($purchaseorder->getItems() as $i => $item): ?>
25    <tr>
26      <td><?php echo $item->getItem()->getName()?></td>
27      <td><?php echo $item->getOrderedunits()?></td>
28    </tr>
29    <?php endforeach; ?>
30  </table>
31  <?php endif; ?>
```

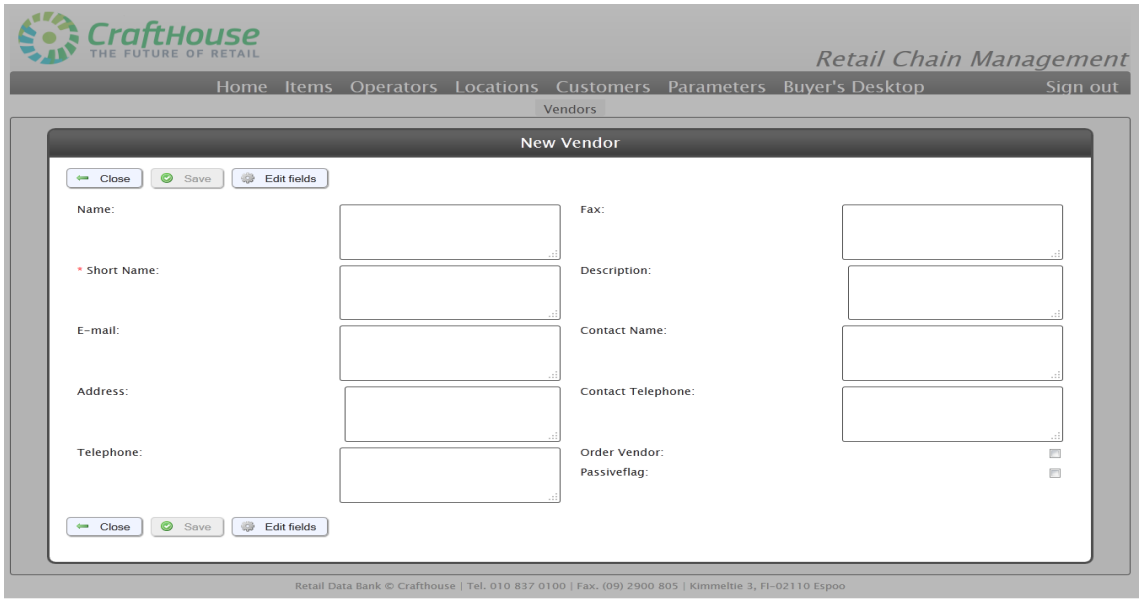
Figur 18. Exempel på utskrift av beställningens varor.

Om *$count* är noll betyder det att beställningen inte innehåller varor och då visas inte heller någonting.

4.5.2 Formulär

För att kunna visa ett formulär genom att enbart kalla på *<?php echo $form ?>* från vyfilen måste man först initialisera formulärobjektet i kontrollern och dessutom definiera bland annat vilka fält som skall visas, standardvärden och valideringen i modellen. Exempel på skapande av formulärobjektet visas i Figur 16 och i detta kapitel behandlas modellen för formulär. Man kommer åt formulärobjektens modellfiler från filerna i projektkatalogen */lib/form/doctrine/*. Som exempel kommer formuläret för skapande av ny leverantör att användas.

Om man inte skulle göra förändringar till modellen, utan enbart använda formuläret som genererades i samband med skapandet av modulen skulle man råka ut för problem. Formuläret skulle visa databasens alla fält förutom id-fältet, och dessutom skulle id-fältet förbli utan värde och sparandet skulle misslyckas eftersom kolumnen är definierad att inte få innehålla NULL-värden. Följande figur representerar utseendet av formuläret i bläddraren med modellfilen oförändrad:



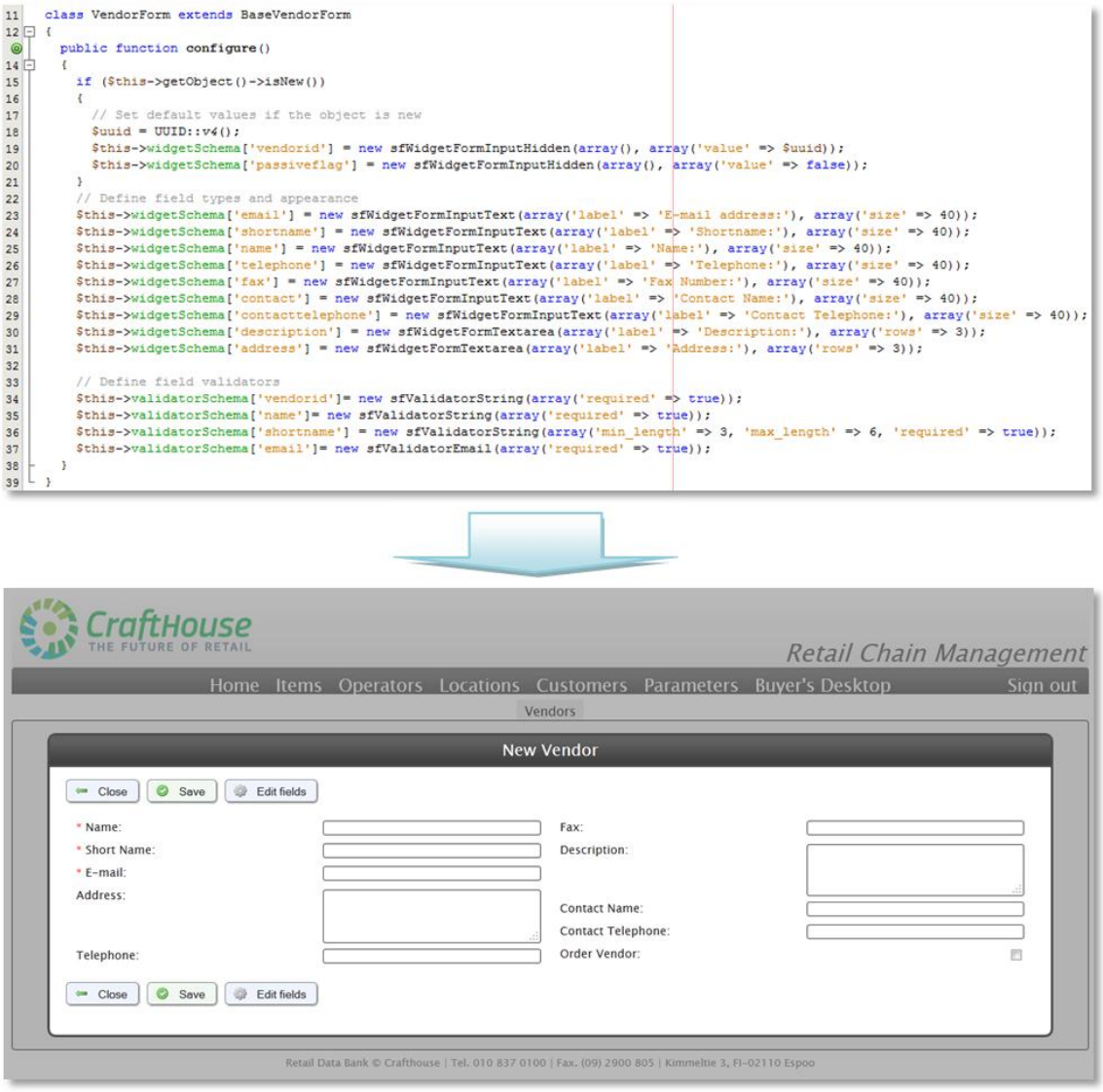
Figur 19. Formulär för skapande av leverantör med formulärets modellfil oförändrad.

Det specificerades att vid skapande av en ny leverantör måste *vendorid*-fältet få en UUID, *shortname*-fältet måste vara mellan tre och sex tecken långt och *email*-fältet måste följa e-postadress formatering. Dessutom specificerades att *vendorid*- och

passiveflag-fälten inte skulle synas och att *vendorid*-, *name*-, *shortname*- och *email*-fälten är obligatoriska att fylla i.

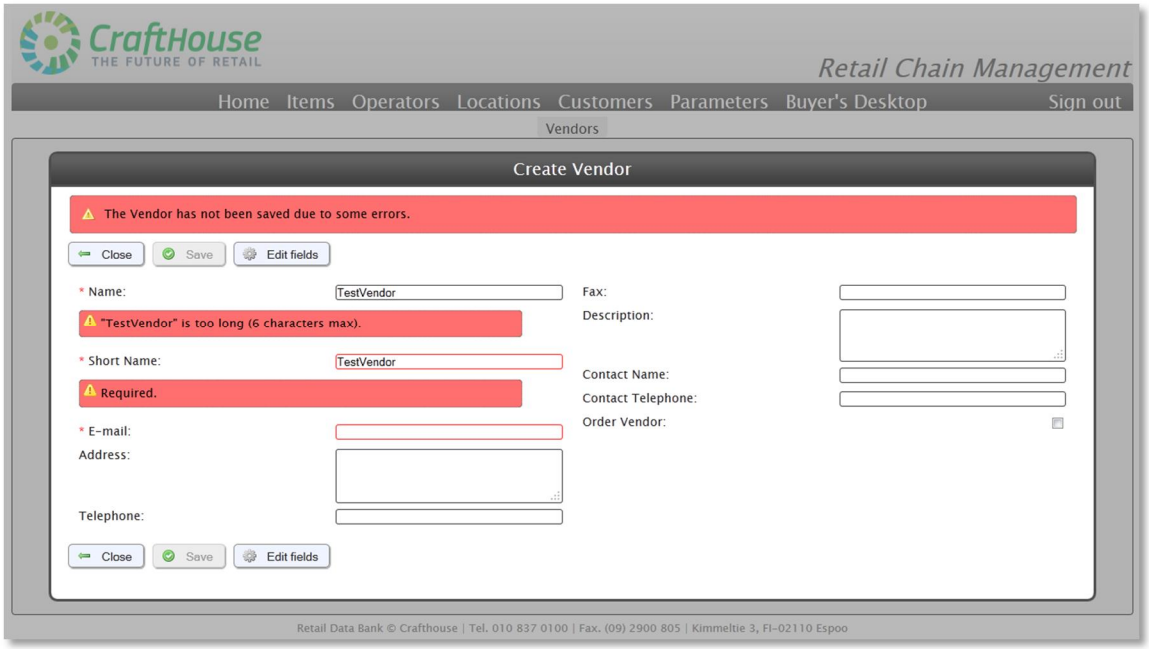
Modellfilen för *vendor*-modulens formulär är */lib/form/doctrine/VendorForm.class.php* och funktionen *configure()* måste ändras på för att kunna uppfylla specifikationerna. Först kollas om objektet är nytt och i så fall läggs som standardvärde för *vendorid* en ny UUID, som fås genom att kalla på funktionen *UUID::v4()*, och som standardvärde för *passiveflag* ett *FALSE*-värde. Efter det definieras hurudana fält skall representera de andra kolumnerna, vad det skall stå framför varje fält och hur breda fälten skall vara. Till sist definieras hur de fälten som behöver valideras, skall valideras.

I Figur 20 kan man se hur *configure()*-funktionen ser ut i *VendorForm.class.php* och hur det resulterande formuläret ser ut i bläddraren:



Figur 20. Exempel av formulär efter att ändra på dess modellfil.

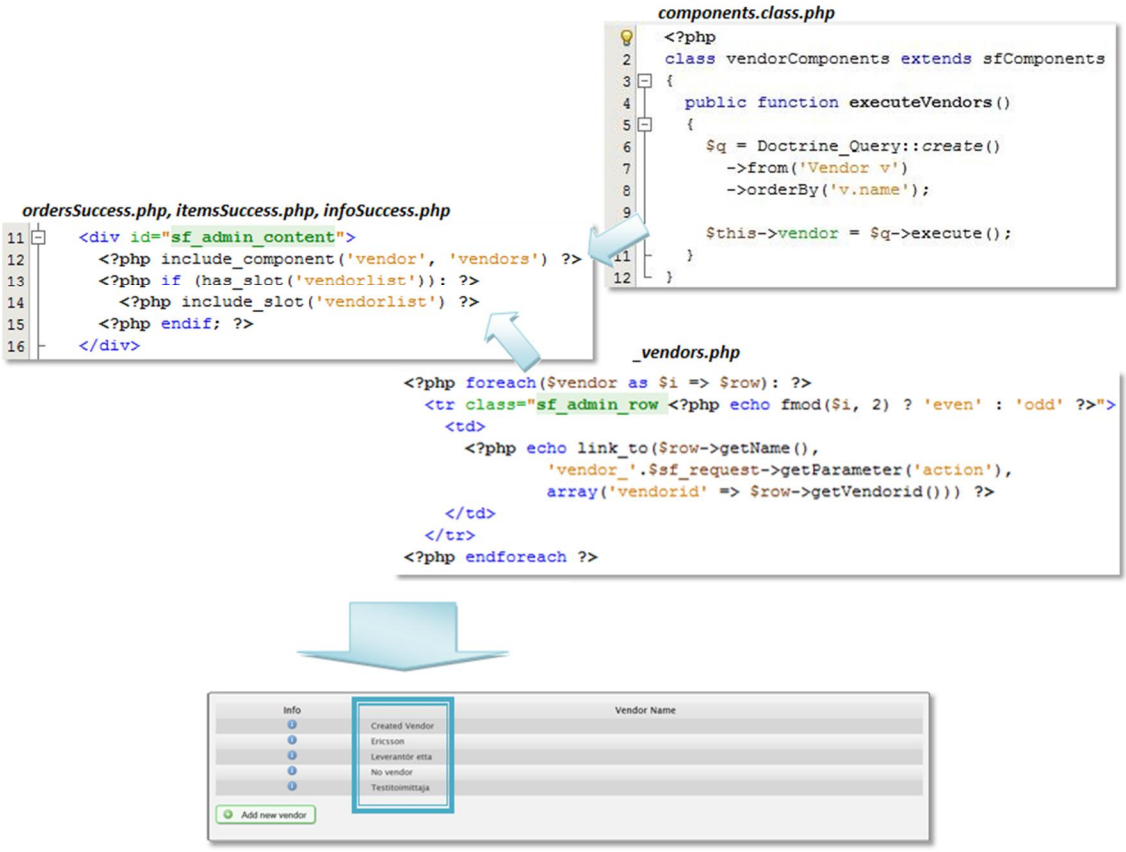
Om man fyller i felaktig data i formuläret eller lämnar ett obligatoriskt fält ofyllt berättar applikationen vilka fält är felaktiga och av vilken orsak. Denna funktionalitet görs av Symfony-ramverkets formulärvalidering på basen av de fyra sista raderna i *configure()*-funktionen i Figur 20. Skapandet av en ny leverantör är omöjligt innan valideringskraven för alla fält är uppfyllda.



Figur 21. Exempel på hur bristfällig data i formulär indikeras för användaren.

4.5.3 Listning av leverantörer

Själva listningen av leverantörer är gjord i tre steg. Först görs i modulens *component.class.php*-fil en databasförfrågan, som ber efter alla rader från leverantörtabellen. Utskriften av varje leverantörs namn görs sedan i ett separat kodfragment, där alla rader itereras igenom och varje leverantörnamn skrivs ut så att varannan rad får en annan bakgrundsfärg än den föregående raden. För att applikationen skall veta att den skall använda funktionen i *components.class.php* och visa kodfragmentet måste detta definieras i vyfilen. Figur 22 visar hur allt hänger samman och hur HTML- och PHP-kod har kombinerats för att nå önskvärda resultat:

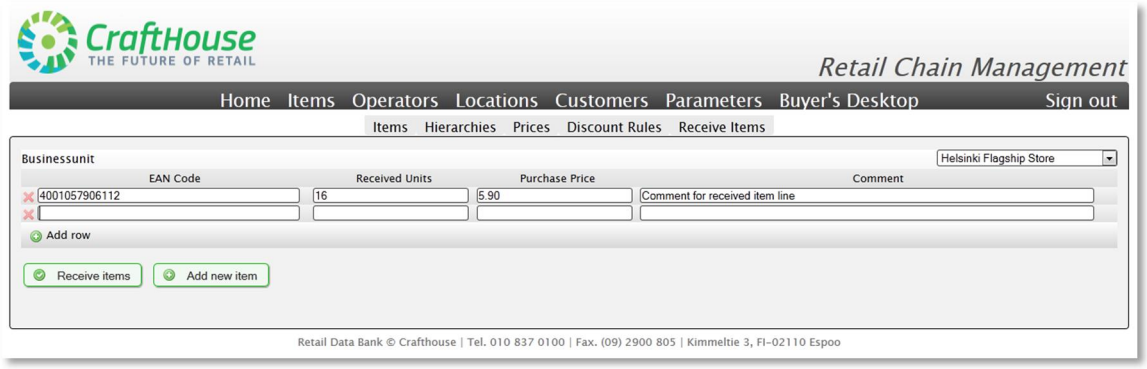


Figur 22. Kodexempel på listning av leverantörer.

Valet att använda en *komponent* och en *slot* för leverantörlistningen gjordes för att då man navigerar mellan den valda leverantörens varor, beställningar och grundinformation är det egentligen tre olika templates som visas. Detta var det enklaste sättet att visa samma information utan att behöva definiera den på flera ställen. Samtidigt erbjuder det en möjlighet att återanvända koden på ytterligare ställen.

4.5.4 Dynamiska rader för mottagande av varor

Då man vill motta varor som inte är en del av en beställning, kan man göra det via en egen sida. För att minimera stegen som användaren måste ta och för att effektivera processen har sidan en funktionalitet, som tillåter användaren att dynamiskt mata in informationen för så många varor som denne vill motta och så kan alla rader registreras på en gång. Utan denna funktionalitet, skulle användaren tvingas mata in informationen för en vara, registrera mottagningen och sedan ännu återvända till den ursprungliga sidan för att motta nästa vara och sedan igen repetera proceduren.



Figur 23. Sidan för mottagande av varor som inte är en del av en beställning.

Figur 23 visar hur sidan ser ut i bläddraren. Man kan lägga till rader till tabellen genom att klicka på den nedersta raden av tabellen med texten òAdd rowò och man kan ta bort vilken som helst osparad rad genom att klicka på det röda krysset vid vänstra kanten av raden.

Hur nya rader läggs till förverkligas med AJAX. I modulens vyfil *IndexSuccess.php* har en AJAX-funktion *addNewCode( )* tillagts, liksom en händelsehanterare som kallar på funktionen då man klickar på den nedersta raden i tabellen med *id=**òaddCodeò***. Följande figur presenterar hur AJAX-koden ser ut:

```
75          function addNewCode()
76          {
77            $.ajax(
78            {
79              type: "POST",
80              url: "<?php echo url_for('receiveditem/AddCode') ?>",
81              data: ({ newcodescount: newcodescount }),
82              success: function(data)
83              {
84                $('#code_table').find('tbody').append(data);
85                $('#code_table').find('tbody').find('tr:last').fadeIn(700);
86                setInputFieldSizes();
87                if (newcodescount > 0)
88                {
89                  $('input[id$=code]').focus();
90                }
91                newcodescount++;
92              }
93            });
94          }
95
96          // Bind the "Add new code" button to its function
97          $('#addCode').live('click', function()
98          {
99            addNewCode();
100         });
```

Figur 24. AJAX-funktion och händelsehanterare för att lägga till ny rad.

Funktionen *addCode( )* kallar i sin tur på modulens *executeAddCode( )*-funktion i *actions.class.php* som returnerar renderat innehåll av vyfilen *_addNewCode.php.* Det renderade innehållet av *_addNewCode.php* är egentligen PHP-koden för en rad i tabellen. Figur 25 visar *addCode( )*-funktionens uppbyggnad och *_addNewCode.php*-filens innehåll.

executeAddCode( )-funktionen

```
174    public function executeAddCode(sfWebRequest $request) {
175      $this->forward404unless($request->isXmlHttpRequest());
176
177      $number = intval($request->getParameter('newcodescount'));
178      $this->form = new ReceiveditemForm();
179      $this->form->addNewCodeFields($number);
180
181      return $this->renderPartial('addNewCode',
182              array('form' => $this->form, 'number' => $number));
183    }
```

_addNewCode.php

```
2   <tr>
3     <td> <?php echo image_tag('icons/textfield_delete',
4             'class=delete_row title="Clear not yet saved code row"') ?>
5          <?php echo $form['newcodes'][$number]['receiveditemid'] ?></td>
6     <td> <?php echo $form['newcodes'][$number]['code'] ?> </td>
7     <td> <?php echo $form['newcodes'][$number]['receivedunits'] ?> </td>
8     <td> <?php echo $form['newcodes'][$number]['purchaseprice'] ?> </td>
9     <td> <?php echo $form['newcodes'][$number]['comment'] ?> </td>
10  </tr>
```

Figur 25. Funktionen addCode( ) och vyfilen _addNewCode.php.

Att ta bort en rad görs också med en AJAX-händelsehanterare, som aktiveras då man klickar på ett element i tabellen med *class=**òdelete_rowò***. I den ifrågavarande tabellen är det röda krysset i vänstra kanten av raden i Figur 23 det enda elementet som hör till den klassen. Figur 23 visar hur händelsehanteraren ser ut i *IndexSuccess.php:*

```
108          // Remove new rows which have not been saved yet
109          $('.delete_row').live('click', function() {
110            $(this).closest('tr').fadeOut('slow', function() { $(this).remove(); });
111          });
```

Figur 26. AJAX händelsehanterare för borttagning av rader från tabellen.

4.6 Datasäkerhet

Förutom den inbyggda datasäkerheten i ramverket Symfony, såsom till exempel sessionshanteringen, inbyggt skydd mot XSS-anfall (Cross-Site Scripting) och inbyggt skydd mot CSRF-anfall (Cross-Site Request Forgery) har tilläggsmodulen *sfDoctrineGuardPlugin* använts i detta projekt. SfDoctrineGuardPlugin är en plugin som erbjuder autentiserings- och auktoriseringsfunktionalitet som är lätt att realisera.

För tillfället måste en användare endast autentisera sig själv, det vill säga logga in, för att komma åt funktionaliteten i applikationen. I ett senare skede kommer det att läggas till olika rättighetsnivåer för både enskilda användare och användargrupper för alla handlingar inom applikationen.

5 AVSLUTNING OCH DISKUSSION

Resultatet av arbetet blev en fullständig kravspecifikation samt en effektiv och utvidgningsbar databasuppbyggnad för Inköpsverktyget, som en del av projektet RCMS. Själva praktiska realiseringen av Inköpsverktyget har utvecklats så långt att man kan använda verktyget, men det kräver en hel del vidareutveckling och omfattande testning innan det är färdigt för produktionsanvändning.

5.1 Vidareutveckling av Inköpsverktyget

Även om Inköpsverktyget går att använda i sitt nuvarande skick, finns det många möjligheter för vidareutveckling. För att uppfylla kravspecifikationen borde för det första funktionaliteten för beställningsförslag förverkligas. Andra saker som kunde realiseras är till exempel möjligheten att göra samlade beställningar för flera enheter inom en detaljhandelskedja, så att det per enhet går att bestämma hur mycket av varje vara skall beställas av leverantören. För att underlätta användningen av Inköpsverktyget i större detaljhandelskedjor, med mycket information i databasen, skulle filtrering av informationen som visas för användaren vara väldigt nyttig att realisera. Det skulle också vara smart att skapa omfattande testprocedurer för all funktionalitet i Inköpsverktyget, för vilket man även kunde använda ramverket Symfony.

5.2 Ramverket Symfony

Ett av målen med examensarbetet var att lära sig utveckla webbapplikationer med hjälp av PHP-ramverket Symfony. Det är ett komplett ramverk med mångsidig automatiserad funktionalitet som kräver att man lär sig hur det fungerar och hur det är uppbyggt. Om man anstränger sig blir man garanterat belönad till sist. Utvecklaren av ramverket, Fabien Potencier, har skrivit väldigt heltäckande dokumentation och dessutom finns det på ramverkets hemsida olika tutorials som hjälper till att praktiskt komma igång med ramverket.

5.2.1 Inlärning

Ramverket Symfony har en väldigt brant inlärningskurva; speciellt gällande avancerad funktionalitet. Med hjälp av tutorials på ramverkets hemsida kunde man relativt snabbt få igång ett projekt och utveckla basfunktionalitet, men det tog ganska länge innan jag lärde mig vilka filer befann sig var och till vilka filer man skulle göra ändringar får att nå önskade resultat.

Till en början var det svårt att tänka på det sättet som MVC-designmönstret kräver och det var också krävande att se databastabellerna som användbara objekt, för att kunna utnyttja den objekt-relationella kartläggningsmodellen.

5.2.2 Fördelar

Ramverket Symfony medför många fördelar som underlättar livet för utvecklare, genom att göra saker i stället för dem eller genom att förenkla komplicerade uppgifter. Då man använder ramverket är det inte lätt att fuska med designmönstret MVC och därmed förblir koden också välstrukturerad och vidareutveckling underlättas.

Felsökningen i ramverket är också en stor fördel. Då man använder applikationer i utvecklingsomgivningen redovisas varje fel och meddelande, alla databasoperationer loggas och även exekveringstiden mäts automatiskt av ramverket, så man kan få ut all information som man behöver om en sida på några sekunder.

5.2.3 Nackdelar

Ramverket Symfony innebär också vissa nackdelar. Som en av de största nackdelarna skulle jag nämna dess branta inlärningskurva. En annan nackdel är det väldigt heltäckande och därmed väldigt komplicerade konfigureringssystemet, som innebär att man måste administrera väldigt många filer.

Det var väldigt svårt att hitta information om ramverket från andra källor än från ramverkets egen dokumentation. All teori är på sätt och vis partisk och det är därför svårt att säga vad som egentligen är bra med ramverket, och om vad som kunde vara bättre att utföra med ett annat liknande ramverk.

5.3 Slutsats

Jag har lärt mig väldigt mycket om att utveckla webbapplikationer och om ramverket Symfony. Speciellt överraskande var hur mycket webbapplikationer och utvecklandet av dem skiljer sig från traditionella desktop-applikationer.

Vad jag ännu skulle studera vidare om ramverket är användningen av externa plugins. Det finns massor av externa plugins tillgängliga och jag tror att det vore värt att lära sig använda dem på ett klokt sett.

Om jag nu skulle börja utveckla en ny webbapplikation skulle jag definitivt använda ramverket Symfony och jag skulle denna gång vara mycket snabbare och effektivare i utvecklandet. Jag skulle börja testa direkt från början, för att minska på arbetsmängden under och i slutet av utvecklingsskedet.

För någon annan som tänker på att börja utveckla webbapplikationer med hjälp av ramverket Symfony skulle jag rekommendera att först studera teorin grundligt, innan man börjar själva utvecklingen för ramverket är så mångsidigt att man sparar på arbetsmängden om man vet vad man gör innan man börjar göra det.

KÄLLOR

Görling, Stefan. 2009, *Att arbeta med IT-projekt*, Studentlitteratur AB, Lund, 308 s.

Jarmoğowicz J.; Napieralski A. & Zabierowski W. 2008, *Presentation of Improvements for PHP Programmers, Based on Symfony Framework. Creation of Example Portal and Description of Used Technology*, Modern Problems of Radio Engineering, Telecommunications and Computer Science, 2008 Proceedings of International Conference TCSET 2008, årg. 2008, 663 s.

Potencier, Fabien; Zaninotto, François. 2010, *A Gentle Introduction to Symfony 1.4*, Sensio Labs, 397 s.

Firebug 2012, What is Firebug. Tillgänglig http://getfirebug.com/whatisfirebug Hämtad 17.1.2012.

Git 2011, Home ï Git, The fast version control system. Tillgänglig: http://git-scm.com Hämtad 21.09.2011.

National Retail Federation 2011, The Association For Retail Technology Standards. Tillgänglig: http://www.nrf-arts.org/about Hämtad 03.11.2011.

NetBeans 2011, NetBeans IDE 7.0.1 Release Information. Tillgänglig: http://netbeans.org/community/releases/70 Hämtad 21.09.2011.

Notepad++ 2011, About. Tillgänglig: http://notepad-plus-plus.org Hämtad 21.09.2011.

TortoiseSVN 2011, About TortoiseSVN. Tillgänglig: http://tortoisesvn.net Hämtad 21.09.2011.

YAML 2011, The Official YAML Web Site. Tillgänglig: http://www.yaml.org Hämtad 30.08.2011.

Wikipedia 2011a, Objektorientering ï Wikipedia, Den fria encyklopedin. Tillgänglig: http://sv.wikipedia.org/wiki/Objektorientering Hämtad 24.08.2011.

Wikipedia 2011b, YAML ï Wikipedia, Den fria encyklopedin. Tillgänglig: http://sv.wikipedia.org/wiki/YAML Hämtad 30.08.2011.

Wikipedia 2011c, UUID ï Wikipedia, Den fria encyklopedin. Tillgänglig: http://en.wikipedia.org/wiki/Universally_unique_identifier Hämtad 14.11.2011.

BILAGA: SAMBAND MELLAN DATABASTABELLER I INKÖPSVERKTYGET