Utveckling av ett projekt- och timhanteringssystem för Raseborgs Metall

Martin Molnár

Nico Pulkkinen

Examensarbete för Tradenom-examen

Utbildningsprogrammet i informationsbehandling

Raseborg 2011

EXAMENSARBETE

Författare: Martin Molnár & Nico Pulkkinen

Utbildningsprogram: Informationsbehandling, Raseborg

Handledare: Rolf Gammals

Titel: Utveckling av ett projekt- och timhanteringssystem för Raseborgs Metall

---

Datum: 23.11.2011 Sidantal: 64 Bilagor: 1

Sammanfattning

Uppgiften för examensarbetet var att utveckla ett skräddarsytt projekt- och timhanteringssystem som stöd för verksamheten på företaget Raseborgs Metall. Systemet hanterar uppföljning av projekt i fråga om materialåtgång, arbetstimmar och använd maskintid. Systemet stöder löneräkningen genom att anställda själva kan mata in sina timmar, som sedan kan granskas av löneräkningen enligt löneperiod.

Systemet har integrerats på företagets webbserver och kan användas från vilken dator som helst med internetuppkoppling och en tillräckligt ny webbläsare. Systemet är lätt att använda även för den del av användarna som inte har någon större erfarenhet av datorer. Gränssnittet fungerar både på svenska och finska. Programmet är i huvudsak utvecklat i skriptspråket PHP, men en del av funktionaliteten är skriven i Javascript.

I framtiden kan detta arbete användas av kunden som en mera avancerad användarmanual och som ett stöd för det dagliga arbetet med systemet. Även framtida utvecklare och Raseborgs Metalls samarbetspartners har nytta av arbetet, då programmets funktioner beskrivs i detalj.

Arbetets huvudsakliga fokus ligger på utvecklingen av programmets funktionalitet, men beskriver även objektorienterad programmering samt annan typ av kod vi använt oss av. Arbetet behandlar inte HTML- eller CSS-kod, då dessa inte är relevanta i fråga om funktionalitet.

Språk: Svenska Nyckelord: Objektorienterad, affärssystem, PHP, javascript, jQuery

---

OPINNÄYTETYÖ

Kirjoittanut: Martin Molnár & Nico Pulkkinen

Koulutusohjelma: Tietojenkäsittely, Raasepori

Opintojen ohjaaja: Rolf Gammals

Otsikko: Utveckling av ett projekt- och timhanteringssystem för Raseborgs Metall

---

Päivämäärä: 23.11.2011 Sivut: 64 Liitteet: 1

Tiivistelmä

Opinnäytetyön tehtävänä oli kehittää räätälöity toiminnanohjausjärjestelmä tukemaan Raaseporin Metalli Oy:n yritystoimintaa. Järjestelmän tarkoituksena on projektien, materiaalin käytön, kulutetun työajan sekä koneiden käytön seuranta. Järjestelmä tukee palkanlaskentaa, sillä työntekijät pystyvät itse lisäämään ja tallentamaan työtuntinsa jotka tarkastetaan palkanlaskennassa palkanmaksukauden mukaan.

Järjestelmä on yhteensopiva Raaseporin Metallin nykyisen palvelimen kanssa ja se toimii kaikissa tietokoneissa, joista löytyy Internet-yhteys sekä uudempi selain. Kokemattomampien tietokoneen käyttäjien vuoksi järjestelmä on tehty helppokäyttöiseksi. Käyttöliittymä on sekä suomen- että ruotsinkielinen. Ohjelma on pääasiallisesti kehitetty käyttämällä PHP-ohjelmointikieltä, mutta osa toiminnoista perustuu Javascript-kieleen.

Tulevaisuudessa asiakas voi käyttää opinnäytetyötä perusteellisena ohjekirjana ja päivittäisen työnteon tukena järjestelmän parissa. Myös järjestelmän tulevat kehittäjät voivat hyödyntää opinnäytetyötä, sillä kaikki toiminnot on käyty läpi yksityiskohtaisesti.

Opinnäytetyö keskittyy pääasiassa järjestelmän toimintojen kehittämiseen, mutta kuvailee myös käytettyä olio-ohjelmointitekniikkaa. Tavallista HTML- ja CSS-koodia ei käsitellä, sillä nämä eivät ole olennaisia ohjelman toimintoja ajatellen.

Kieli: Ruotsi Avainsanat: Olio-ohjelmointi, toiminnanohjausjärjestelmä, PHP, Javascript, jQuery

---

BACHELOR'S THESIS

Authors: Martin Molnár & Nico Pulkkinen
Degree Programme: Business Information Technology, Raseborg
Supervisor: Rolf Gammals
Title: The Development of an Enterprise Resource Planning System for Raseborgs Metall / Utveckling av ett projekt- och timhanteringssystem för Raseborgs Metall

---

Date: 23 November 2011 Pages: 64 Appendices: 1

Summary

The task for this thesis was to develop a tailor-made ERP (Enterprise Resource Planning) system that supports the day-to-day activities at the company Raseborgs Metall. The system facilitates monitoring of projects in terms of material consumption, hours worked and use of machines. The system also makes the follow-up of the employees' monthly hours worked much easier since every employee now enters his own hours into the system. The people managing salaries can easily access this information and filter it by date to get the hours for a certain period of time.

The system is integrated on the company's Web server and can be used from any computer with an Internet connection and a sufficiently powerful browser. The system is also easy to use, since some users do not have much experience of computers. The interface is available in both Swedish and Finnish. The program is developed using the programming language PHP, but it also incorporates Javascript for certain functions.

This thesis works as a more detailed user manual and as support for the daily activities concerning the use of the system. All functions have been described in detail and future developers may benefit from this thesis.

The main focus of the thesis is the development of the functionality of the system, but the object-oriented programming technique used is also described. A description of regular HTML and CSS code is not included, as they are not relevant for the functionality.

Language: Swedish Keywords: Object-oriented, ERP-system, PHP, Javascript, jQuery

---

Innehållsförteckning

1 Introduktion

Allt fler företag investerar i affärssystem för att underlätta det dagliga arbetet. Konkurrensen blir allt hårdare och kampen om kunderna allt intensivare. Det är väldigt viktigt att det administrativa arbetet i ett företag inte är ett hinder för verksamheten. Problemet för många är att systemen på marknaden inte är skräddarsydda för just deras verksamhet och man blir tvungen att anpassa sin verksamhet enligt systemet istället för tvärtom. Vill man skräddarsy ett system för sig handlar detta ofta om en avsevärd investering, då kostnaden för anskaffning och underhåll är hög.

Vårt uppdrag är att skapa ett lättanvänt och lättunderhållet projekt-, personal-, material- och arbetstidshanteringssystem åt Raseborgs Metall. Målsättningen är att kunna minimera den tid som går åt till administrativt arbete i samband med löneräkning, projekthantering och materialuppföljning. Systemet skall även underlätta uppföljningen av projekt och de resurser som har använts i dessa.

Systemet kommer inte enbart att underlätta uppföljningen och administrationen av det dagliga arbetet utan kommer också att sammanställa nödvändig information på ett och samma ställe. Det kommer exempelvis att fungera som ett register för kunder och anställda där man har tillgång till kontaktuppgifter och annan tillbörlig information.

För anställda kommer systemet även att ersätta de gamla timlistorna som lämnas in eftersom var och en bokför sina egna arbetstimmar i systemet.

För oss som jobbar med projektet är det ett ypperligt tillfälle att fördjupa våra kunskaper i PHP och få en inblick i hur en organisation fungerar och vad vi som IT-kunniga kan göra för att underlätta arbetet för företag i andra branscher än vår egen.

1.1 Uppdragsgivare

Uppdragsgivare till projektet är Tom Fagerstedt, verkställande direktör på Raseborgs Metall. Kunden kontaktade själv Yrkeshögskolan Novia och berättade om sina behov, varefter vi tog oss an projektet som examensarbete.

1.1.1 Om företaget

Raseborgs metall är ett aktiebolag som verkar inom metallbranschen. Bolaget är grundat år 1993 och specialiserar sig på planering, tillverkning och installation av

metallkonstruktioner. Till de tjänster bolaget erbjuder hör svetsning, optisk skärbränning, svarvning, fräsning, hydraulpressning och profilmangling. Företaget tillverkar både nya konstruktioner och utför reparationsarbeten på slitna delar. År 2007 hade företaget en omsättning på 1,1 miljoner euro och sysselsatte 13 personer. Samarbete har gjorts med bland annat IDO Kylpyhuone Oy, Lassila & Tikanoja, Oy Forcit Ab, Tammet Oy och Ovako Wire Oy Ab.

1.1.2 Bakgrund till uppdraget

På grund av mängden papper som ackumuleras vid projekthantering, löneräkning, materialuppföljning och övriga personalrelaterade ärenden bestämde man sig på Raseborgs Metall för att man måste investera i ett affärssystem. Affärssystem med de erforderliga funktionerna finns redan på marknaden, men eftersom implementeringen av sådana system är väldigt kostsam och de inte är skräddarsydda för en viss kunds speciella behov beslöt ledningen på Raseborgs Metall att kontakta Yrkeshögskolan Novia för att se om ett samarbete kring detta var möjligt. Eftersom vi var i behov av ett examensarbetesprojekt mottog vi uppdraget med glädje.

1.2 Syfte och målsättning

Vår avsikt var att utveckla ett skräddarsytt affärssystem med god användbarhet och användarvänlighet. Vår tidigare erfarenhet av affärssystem består av mångårig användning av Gigantti Oy:s affärssystem Elguide och IDO Bathroom Oy:s affärssystem IFS Applications. På basis av vår erfarenhet av dessa system har vi kunnat jämföra och utveckla vårt system till att motsvara kommersiella system i fråga om användarvänlighet och funktionalitet. Den största vikten i detta arbete ligger på själva utvecklandet, men även användarvänlighet och installation behandlas.

Målet med detta projekt var att inom utsatt tid färdigställa ett välfungerande och användarvänligt affärssystem skräddarsytt för Raseborgs Metalls behov. Systemet skall fungera i enlighet med de önskemål och krav kunden ställt. Koden skall även vara kommenterad för att underlätta eventuell vidareutveckling.

1.2.1 Målgrupp

Vår primära målgrupp för arbetet är personer intresserade av systemutveckling med PHP. Texten kan också vara av intresse för studerande som studerar systemutveckling samt

företaget Raseborgs Metall, för vars räkning detta system utvecklas. Målgruppen för själva systemet är såväl ledningen som arbetarna på Raseborgs Metall.

1.2.2 Avgränsning

Vi har inriktat oss på utvecklandet och problemlösningen inom programmeringen av systemet samt arbetet med att implementera de tekniska lösningarna i ett sådant gränssnitt där även personer med sämre datorkunnande kan använda systemet utan svårigheter. Vi presenterar inte programmets kod i sin helhet i själva avhandlingen utan inriktar oss på de viktigaste funktionerna och utvecklandet av dessa. Vi förklarar inte heller i detalj all den kod som hör till funktionerna vi skriver om utan presenterar utvalda delar som vi anser intressanta. Vi berättar om utvecklandet av funktionerna på en nivå där en del kan kännas främmande för dem som inte är insatta i PHP-programmering. För läsaren som inte är insatt i kodspråket ges ändå förklaringar och man behöver inte nödvändigtvis förstå själva koden för att texten skall vara intressant.

1.3 Definitioner

1.3.1 PHP

PHP står för Hypertext Preprocessor och är ett skriptspråk som används för att köra dynamiskt innehåll på webbservrar. Språket började utvecklas av Rasmus Lerdorf år 1994 då han sammanställde en mängd Perl-skript för att hantera sin hemsidas loggning. Eftersom intresset för dessa skript växte släppte han dessa i paket. Den första versionen som utkom kallade han ”Personal Home Page”. Ett år därefter skrev han en skriptmotor som avkodar HTML-formulär och som tillsammans med det redan utvecklade Personal Home Page bildade PHP/FI 2.0. Efter version PHP/FI 2.0 (Personal Home Page/Forms Interpreter) övergick man till att använda förkortningen PHP. PHP användes i ganska liten utsträckning ända fram till år 1999. Enligt Netcraft växte antalet domäner som använde PHP från 1 miljon till 5 miljoner på ett år. PHP 4 släpptes under sommaren 2000 och var en ordentlig uppgradering av den tidigare PHP 3. Detta kan ha varit en bidragande orsak till den snabba ökningen av domäner som använde PHP. PHP 5 släpptes sommaren 2004 och är den version som för tillfället används. Under åren har det kommit många uppdateringar till PHP som utökat dess funktionalitet, fixat buggar, säkerhetsbrister och ökat kompatibilitet med exempelvis Windows.

Skriptspråket anses vara relativt lätt att lära sig för nybörjare och man kan åstadkomma något funktionellt redan efter att ha studerat språket en kortare tid. För professionella utvecklare erbjuder PHP ett snabbt sätt att utveckla webbapplikationer. PHP kan integreras i HTML-dokument och utföra olika funktioner för att göra till exempel en hemsida dynamisk. Med en dynamisk hemsida menas en hemsida som interagerar med användaren.

Syntaxen (skripspråkets ”grammatik”) liknar på många sätt programmeringsspråket C. Språket har influenser från många andra kodspråk förutom C, men även helt unika PHP-egenskaper.

PHP är ett skriptspråk med öppen källkod och är helt gratis att använda. Traditionellt används det tillsammans med en Apache-server installerad på en Unix/Linux-plattform, men det kan lika väl användas på en Windows-plattform tillsammans med en Access databas.

(Rantala 2005, s. 9-10; Jonsson 2001, s. 13-16)

1.3.2 MySQL

MySQL är en typ av databas. För att bättre kunna förstå nyttan med databaser förklaras i detta stycke också kort om databaser i allmänhet. Som exempel har vi ett kundregister med ett antal kunder som blivit för stort att hantera med penna och papper eller kalkylbladstabeller. Ägaren av denna information vill snabbt och enkelt kunna komma åt vissa data. Han vill även kunna sortera sina sökresultat enligt olika kriterier. Låt oss säga att informationen består av namn, adress, postnummer, telefonnummer, födelseår och kön.

För att kunna göra detta skapas en databas som innehåller data i en tabell med kolumner enligt följande:

Namn	Adress	Postnummer	Telefon	Födelseår	Kön

I dessa kolumner fylls sedan information för varje kund in. Efter att informationen är ifylld i databasen kan man söka efter kunder på basis av alla dessa kriterier. Man kan även sortera enligt alla dessa grupper så att man till exempel får den äldsta kunden högst upp och den yngsta lägst ner i tabellen. Med databasfrågor kan man sedan hämta ut enbart viss information ur databasen.

Ett företag kan sedan exempelvis precisera sin marknadsföring så att reklamen för hudkräm endast skickas ut till kvinnor, eller att en viss stads reklamblad enbart går ut till de kunder där postnumret överensstämmer med staden där erbjudandena gäller. Det finns många typer av databaser, men vi har valt att använda oss av MySQL-databas.

MySQL dök upp år 1996 och var då endast en enkel SQL-implementation. Utvecklingen började dock redan år 1979 då en svensk vid namn Michael Widenius skapade databassystemet UNIREG för företaget TcX. Då UNIREG inte mera stödde behoven för TcX utvecklade Widenius MySQL.

MySQL har byggts runt tre grundprinciper, pålitlighet, användarvänlighet och prestationsförmåga. Dessa principer kombinerat med en öppen källkod har över åren gjort MySQL till ett snabbt och kraftfullt system som är kostnadseffektivt och egenskapsmässigt välutvecklat.

Idag betraktas MySQL som världens populäraste databastyp. Det används till hemsidor, sökmotorer och digital datalagring i allmänhet världen över. MySQL används aktivt bland annat av företag och organisationer som Sony, Xerox, HP och NASA.

(Vaswani 2004, s. 1-12)

1.3.3 Javascript

Javascript är ett skriptspråk som används av majoriteten av moderna interaktiva webbsidor. Till skillnad från exempelvis PHP exekveras Javascript oftast inte på servern utan på användarens egen dator. Javascript-tolkningsmotorer finns idag inbyggda i alla moderna webbläsare. Javascript-funktionalitet är nuförtiden inte begränsad enbart till datorer, utan kan också tolkas av webbläsare i telefoner, spelkonsoler och handdatorer. Språket har utvecklats och lanseras i flera olika versioner, men idag är de enda relevanta versionerna 3 och 5. På grund av upphovsrättsliga skäl är det officiella namnet ECMAScript, men språket kallas i folkmun alltid för Javascript. Javascript utvecklades ursprungligen av företaget Netscape, som under 1990-talet var ägare till den populäraste webbläsaren Netscape Navigator.

(Flanagan 2011, s. 1-17)

I vårt program har vi använt Javascript till exempelvis tabellsortering. Kalenderfunktionen som underlättar ifyllningen av datuminformation är också Javascriptbaserad.

2 Val av programmeringsspråk

Under vår utbildning i Informationsbehandling på Yrkeshögskolan Novia har vi bekantat oss med programmeringsspråken C# och PHP. Vi hade ursprungligen tänkt oss att programmera systemet i C# eftersom vi hade mer utbildning i detta programmeringsspråk. Vi valde dock att programmera i PHP på grund av ett antal faktorer.

PHP är ett flexibelt skriptspråk som jämfört med många andra är relativt lätt att lära sig. Eftersom vi inte hade tillräckliga kunskaper i varken C# eller PHP valde vi PHP för att vi då lättare kunde komplettera de kunskaper vi hade. Detta var förstås inte den enda orsaken till vårt beslut. PHP är även billigare i drift för vår kund. För att driva ett webbaserat PHP-system krävs endast en vanlig linuxbaserad webbserver där LAMP är installerat. LAMP är ett programpaket för Linux som består av Apache, MySQL och antingen PHP, Perl eller Python. Dessa bildar en plattform helt baserad på programvara med öppen källkod som är gratis att använda. Jämfört med en Windowslösning, som krävts för ett program utvecklat i C#, är detta ett väldigt förmånligt alternativ.

Med de kunskaper vi hade vid projektets början kunde vi även dra slutsatsen att vi skulle ha större kontroll över ett PHP-baserat system då kunden vill ha ändringar i framtiden. Ifall vi inte kommer att fortsätta inom denna bransch då vi utexamineras från skolan eller inte har möjlighet att underhålla systemet åt kunden, har de även lättare att hitta PHP-kunniga i vår region som kan göra detta åt dem. I Raseborg finns ett flertal företag som arbetar med PHP, medan jag i skrivande stund inte vet ett enda regionalt företag som arbetar med C#. Efter diskussion med kunden kom vi fram till att deras befintliga webbserver kan användas ifall vi väljer att programmera i PHP och om de levererade tjänsterna uppgraderas till att innehålla en databas och mer utrymme för lagring av data.

2.1 Notepad++ istället för Netbeans som programmeringsverktyg

För att underlätta arbetet med att bygga systemet beslöt vi oss för att använda ett program som kallas Netbeans. Netbeans är en IDE (Integrated Development Environment), som tillåter en att hålla direktkontakt med FTP-servern och fungerar som textredigerare för

olika typer av kod. Programmet gör det även snabbare och enklare att navigera i mappstrukturen och att snabbt få fram den fil man söker efter

Till en början verkade detta vara ett program som tillgodosåg våra behov och underlättade vårt arbete. Dock fick vi efter ett tag problem med såväl inloggning som sparande av filer. Några av de problem som förekom var att man inte kunde logga in med sitt FTP-användarnamn trots att det fungerade under förra sessionen och den enda lösningen var att skapa ett nytt. Ett annat problem var att programmet utan att meddela någonting tappade kontakten med servern och inte sparade något av det man just skrivit. Efter ett tag insåg vi att vi sparar tid genom att jobba direkt via en vanlig FTP-klient och använda oss av textredigeringsprogrammet Notepad++ för att editera filerna istället för att använda Netbeans.

Vi har tidigare använt oss av Netbeans när vi jobbat på en annan server och på ett annat nätverk och programmet har fungerat utmärkt, så vi antar att problemen beror på antingen inkompatibilitet mellan servern och programmet eller instabilitet i Novias trådlösa nätverk.

3 Objektorienterad programmering

I vårt projekt har vi strävat efter att jobba enligt en programmeringsmodell som kallas objektorienterad programmering. Detta har för oss inneburit att vi måste vidareutveckla de tidigare kunskaper vi erhållit i vår undervisning. I skolan har vi studerat objektorienterad programmering men inte implementerat det i ett projekt av denna omfattning.

Objektorienterad programmering är en programmeringsstil där man använder sig av ”objekt” för att representera någonting i verkligheten. Ett objekt har tillstånd, beteende och identitet. Lars Mathiassen, Andreas Munk-Madsen, Peter Axel Nielsen och Jan Stage (2001, 18) beskriver objektet som *”En entitet med identitet, tillstånd och beteende.”*

Objektorienterat programmerande har några fördelar framför procedurell programmering. Det är i allmänhet lättare att underhålla ett system som är objektorienterat och programvaran anses av många vara pålitligare. I det långa loppet kan det också innebära kortare utvecklingstider eftersom man kan återanvända delar av koden från ett tidigare projekt. Enligt Englund och Olsson är dessa fördelar säljargument för objektorienterad programmering men behöver inte stämma för alla projekt.

Som tidigare nämnts beskrivs verkligheten i objektorienterad programmering med hjälp av objekt. Objektens egenskaper beskrivs i sin tur med hjälp av klasser. I ett kundregister finns ju många kunder, där varje kund har ett unikt tillstånd och beteende. Klassen samlar ihop en grupp objekt och ger en gemensam beskrivning för dessa. I boken Objektorienterad analys och design beskrivs klassen som *”En beskrivning av en samling objekt som delar struktur, beteendemönster och attribut.”*

Eftersom objektorienterat tänkande återspeglar verkligheten kan tankesättet beskrivas med ett konkret exempel:

Bilen gick sönder, men efter att mekanikern reparerat den fungerade den fint.

I denna mening har vi två objekt (bilen och mekanikern), ett utgående tillstånd för bilen (den är sönder), en händelse som involverar båda dessa (bilen repareras) och en ändring av bilens tillstånd (den fungerar igen). Enligt samma princip fungerar objekten i ett program då en viss händelse appliceras och objektet byter tillstånd.

(Mathiassen, Munk-Madsen, Nielsen, Stage 2001, s. 17-20; Englund & Olsson 2003)

Då vi under projektets gång strävat efter att tillämpa objektorientering har vi kommit till att detta har både för- och nackdelar. En stor fördel för oss är att många så kallade klasser, kan tillämpas över hela projektet. Table-klassen som definierar databasfrågorna som bestämmer vilken information som skall hämtas från databasen är ett bra exempel. I Table-klassen finns några olika funktioner. Vilken som kallas beror på var i programmet man befinner sig och vilken information man vill visa. I Table-klassen finns också raderingsfunktionen som tillämpas oberoende av vad som skall raderas från vilken tabell. Valideringen av formulär sköts också av en klass. I denna ena fil finns alltså valideringsfunktionaliteten för alla formulär i programmet.

Enligt vår mening är den största fördelen med objektorienterad programmering att man tvingas att göra allt dynamiskt. Ifall man behöver återanvända koden behövs mycket små ändringar för att den skall kunna tillämpas i en annan situation. I framtiden betyder det också att samma kod kan tillämpas i andra programmeringsprojekt ifall man har behov av den.

Det finns dock nackdelar jämfört med en mer traditionell programmeringsstil. För oss har det objektorienterade tänkandet ofta betytt att en liten funktion, som annars skulle kunnat

kodas i en och samma fil, måste delas upp i ett flertal olika filer som sedan kallar på varandra och skickar information till varandra.

3.1 Hur vi tillämpat objektorienterad programmering

Objektorienterad programmering går ut på att bygga upp objekt som beskriver verkliga ting. I vårt program har vi kunnat återanvända samma kod för alla våra objekt, med vissa objektspecifika variationer. Detta kapitel beskriver hur vi tillämpat objektorienterad programmering och varför vi i vissa fall avviker från den objektorienterade modellen.

Vår kund önskar kunna följa upp arbetstagarnas arbetstimmar och det material samt den maskintid de har lagt på ett projekt. Därför hanterar vi ett projekt som ett objekt i vårt program. Objektorienterad programmering innebär att vi som programmerare försökt bygga upp en klass som så bra som möjligt beskriver detta objekt. Vi kan inte gå igenom programmeringens grunder, men i detta kapitel förklarar vi hur vårt program interagerar med användaren då han/hon hanterar ett projekt.

Tillsammans med kunden har vi tagit reda på vad en användare bör kunna hantera och manipulera i ett projekt:

- Skapande av ett nytt projekt
- Uppdatering av projektinfo för existerande projekt
- Granska vilka arbetare som arbetar med projektet
- Granska vilket material som har använts i projektet
- Granska vilka maskiner som har använts i projektet
- Granska ifall ett projekt med ett visst projektnamn redan existerar eller ej
- Granska ifall ett projekt med en viss ID är aktiverat eller ej
- Få fram en listning på alla projekt
- Få fram en listning på alla projekt som möter sökkriterier

Vi nämnde tidigare att ett objekt beskriver verkligheten. I detta fall har kunden ett projekt i verkligheten och detta projekt har ett projektnamn samt andra egenskaper. Kunden kan med vårt program spara informationen för projektet i en databas och med hjälp av klassen vi byggt upp manipulera denna information i ett senare skede. Tumregeln inom all

programmering är att undvika skandinaviska tecken som t.ex. å ä och ö. Därför har vi strävat efter att på kodnivå skriva allting på engelska och ett projekt representeras därför i programmet av klassen Project.

Vi har till en viss del avvikit från den traditionella modellen för objektorienterad programmering. Där man i vanliga fall använder sig av funktioner inom klassen för att t.ex. spara ett projektnamn skilt för sig, har vi skapat ett formulär där all information kan sparas med en enda funktion. Detta var i vårt fall en lyckad lösning som resulterade i mindre kod och gav oss möjligheten att återanvända ett och samma formulär för både skapande av ett nytt projekt samt redigering av ett existerande. Kod 1 visar hur våra klasser är uppbyggda och hur en funktion i klassen kan se ut. Koden skapar alltså en klass med namnet Project och innehåller allting som behövs för att kunna komma åt och manipulera projektinformation. I Kod 1 har vi en funktion vid namnet addEditForm(). Denna funktion visar ett inmatningsformulär åt användaren i det grafiska gränssnittet.

Kod 1. Skapandet av en klass samt inmatningsformulär

```
class Project extends Table
{
	function addEditForm($pid = NULL, $path)
	{
		if ($pid) $this->info = $this->getInfo($pid);
		$output .= '<form action="' . $path . '" method="POST"
		name="projectadd">';
		$output .= '<table>';
		$output .= '<tr><input type="text" name="projektnamn"
		value="' . $this->info['projektnamn'] . '" /></td></tr>';
		$output .= '</table>';
		$output .= '</form>';
		$output .= '</div>';
		return $output;
	}
}
```

Användaren ser aldrig en klassfil i gränssnittet, utan han/hon interagerar med klassen via andra filer som vi kallar för actions-filer. En actions-fil granskar vad användaren har valt att visa på en sida och anropar sedan funktioner från rätt klasser för att få fram den information som behövs. Vi kommer att förklara användningen av dessa olika typer av filer senare i arbetet.

4 Filsystem och programstruktur

Då man programmerar objektorienterat blir filstrukturen lite annorlunda än vid procedurell programmering. I detta stycke beskrivs hur filsystemet är upplagt. I Figur 2 i slutet av kapitel 4 visas en översikt på de tre första nivåerna av filsystemet.

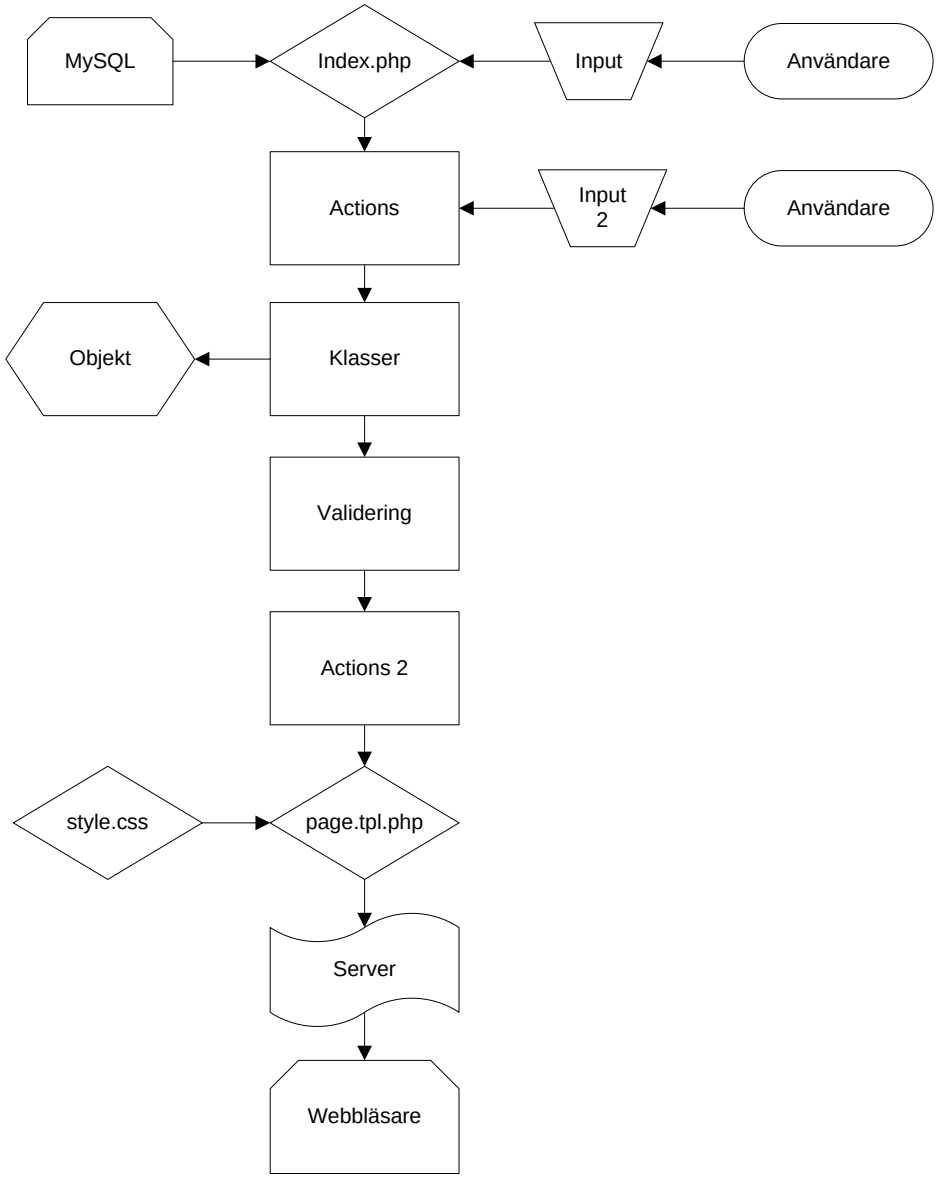
4.1 Skriptspråksfiler

Eftersom vi strävar efter att använda oss av objektorienterad programmering är programmeringsfilerna indelade i mappar med namnen ”actions” och ”classes”. I rotmappen ligger bootstrap.php, index.php och settings.php. Tidigare i arbetet redogjorde vi för hur klasser beskriver en samling objekt med samma struktur och beteende. Med actions-filerna i sin tur kan man dra paralleller till reparationsprocessen som bilen utsattes för i kapitel 3. Dessa actions-filer är de som utför operationer som gör att objekten ändrar tillstånd. Så om klasserna beskriver objekten beskriver actions den operation som utförs när man manipulerar objekten.

I includes-mappen finns filer med egen funktionalitet och som utför olika uppgifter. Dessa filer ligger separata från varandra och kallas på vid olika tillfällen för att utföra specifika uppdrag. Ett exempel på vad en includes-fil kan göra är function-action.inc som spjälker upp sökvägen i adressfältet. Detta görs för att programmet skall kunna inkludera rätt filer vid rätt tillfälle. I denna mapp ligger även Javascript-filer som sköter om bland annat sortering av tabeller och kalenderfunktionen.

4.2 Filstruktur

I rotmappen ligger tre filer som är nödvändiga för programmets funktionalitet. Filen bootstrap.php inkluderar filerna i classes- och includes-mapparna i programmet så att man i index.php kan inkludera enbart denna fil. I error_log skrivs alla felmeddelanden ut och sparas så att man kan gå igenom dessa vid behov. I settings.php hämtar programmet informationen som används vid inloggningen till databasen. För att kunna visa och manipulera programmets objekt finns klass- och actionsfiler som tillsammans bygger upp funktionaliteten. För att få en lite bättre bild över hur programmet går tillväga då en användare ger ett kommando visar Figur 1 ett flödesschema över hur de olika filerna interagerar med varandra.



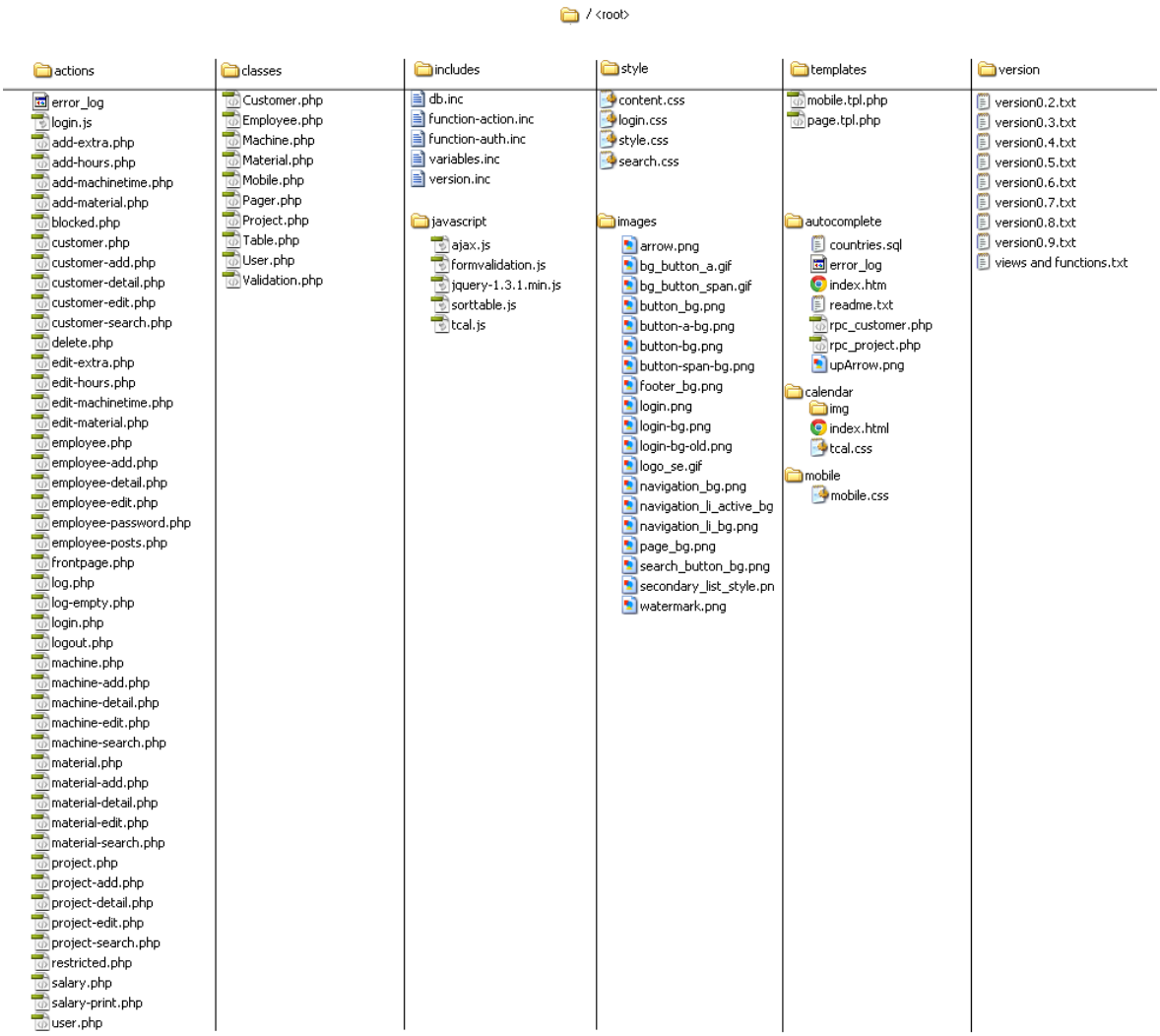
Figur 1. Flödesschema över ett användarkommandos utförandeförlopp.

I flödesschemat ovan beskrivs alltså vilka filer programmet använder sig av då användaren gett ett kommando. Ett kommando kan vara en knapptryckning, till exempel då användaren vill redigera viss information i databasen med hjälp av programmet. Märk att detta inte är någon definitiv väg för hur alla funktioners filer interagerar med varandra utan endast beskriver den logiska vägen för en stor del av uppgifterna programmet utför. Den första delen av flödesschemat är alltså användaren själv och det han eller hon matar in. Vi ser att index.php reagerar på den knapptryckning som har gjorts med att kalla en actionsfil för att utföra en funktion. I det här fallet är funktionen att visa ett formulär för editering av en anställds uppgifter. Actionsfilen hämtar sedan in klassfilen för att den skall kunna

manipulera objektet. Klassfilen i detta exempel är den som definierar de anställda och deras egenskaper. Då användaren gjort sina ändringar och tryckt på Spara skickas informationen vidare till valideringsklassen. Valideringen granskar den inmatade informationen och ifall den är felaktig eller otillräcklig skickas man tillbaka till formuläret för inmatning av informationen tillsammans med ett meddelande som berättar vad man gjort fel. Ifall informationen klarade valideringen sparas ändringarna i databasen och användaren skickas vidare till detaljvyn för den anställda man just manipulerat. Detaljvyn byggs upp av en actionsfil, lika som formuläret för inmatning av information som vi använde tidigare.

Vi har även valt att inkludera filen page.tpl.php i detta flödesschema. Tidigare nämndes att actionsfilerna byggde upp formuläret för inmatning av den anställdes uppgifter och tabellen som informationen sedan visades i. Page.tpl.php i sin tur bygger upp hela sidan och utan den skulle inte något av de andra formulären eller tabellerna visas. Detta gäller för alla sidor i programmet. Tillsammans med style.css bygger page.tpl.php upp den struktur och det utseende som webbläsaren sedan återger i skärmen åt programmets användare. I

Figur 2 visas hur mappstrukturen är uppbyggd och var de olika filerna som används ligger.



Figur 2. En översikt av programmets fil- och mappstruktur.

4.3 Struktur och design

I styles-mappen ligger CSS-filerna som styr systemets utseende. Här ligger även en mapp med grafik och bilder som används. Under ”templates” hittar man själva ryggraden till systemet. I den mappen finns page.tpl.php som specificerar i vilken ordning allt innehåll skrivs ut. CSS-filerna och page.tpl.php-filen i templates bestämmer alltså programmets struktur och design och skapar tillsammans det gränssnitt som användaren ser när han använder programmet.

4.4 Programmets uppbyggnad

Programmet består av sex huvudkategorier:

- Projekt
- Kunder
- Material
- Anställda
- Maskiner
- Timuppföljning

Dessa kategorier presenteras på egna sidor och byggs upp med hjälp av klassfiler som hämtar in information från tabeller i databasen. Timuppföljning har i sig ingen egen tabell utan hämtar informationen via en SQL-vy som skapats just för detta ändamål. Att kalla på en vy gör det lättare att få ut informationen då vi använder samma funktion för att plocka ut data ur flera databastabeller.

I huvudmenyn finns förutom länkarna för var och en av huvudkategorierna även länkar för att lägga till arbetstimmar, maskintid, material och övrig löneinformation. De tre förstnämnda är alla länkade till projekttabellen i databasen, eftersom man skall kunna få ut information om hur många arbetstimmar, maskintimmar och material som gått åt till ett projekt. Övrig löneinformation är enbart kopplad till den anställda och hämtas ut i samband med listningen av arbetstimmar. En arbetares arbetstimmar för den nuvarande månaden hämtas ut och presenteras på dennes profilsida.

5 Databasstruktur

Bakom gränssnittet och PHP-koden som visar informationen för användaren ligger såklart en databas som lagrar all information. Databasen består dels av tabeller som lagrar innehållet och dels av Views. Views är virtuella tabeller som kombinerar kolumner från olika existerande databastabeller. Views används för att man i programmet inte skall behöva skriva ut långa databasfrågor, utan istället kunna anropa vyn man skapat på förhand.

Ett exempel på användning av views är frågan:

```
SELECT * FROM `v_projekt_detalj` WHERE `pid` = $pid LIMIT 1";
```

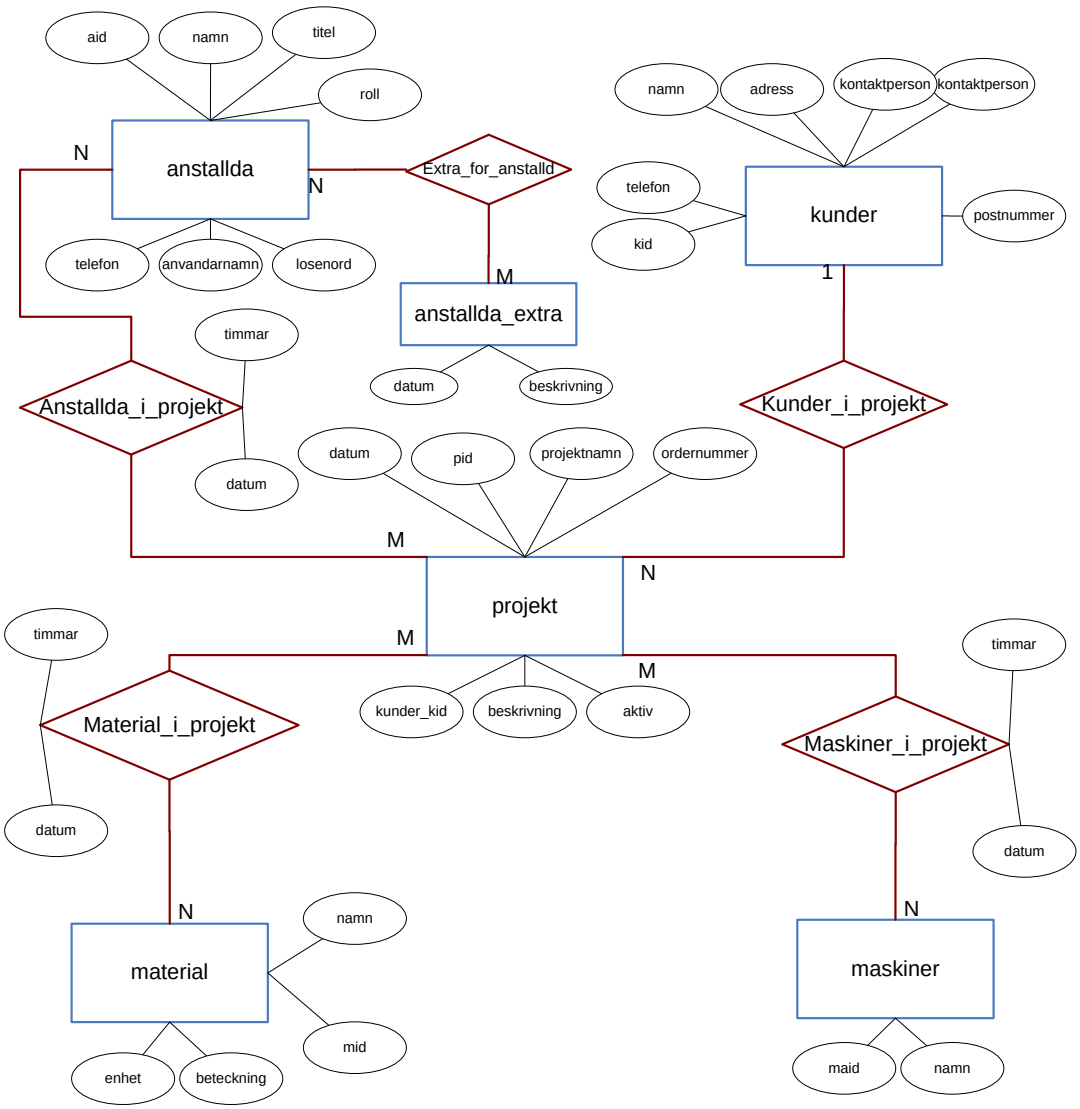
I denna databasfråga väljer man att hämta all information från vyn v_projekt_detalj. Ifall man inte skulle ha använt sig av vyn skulle frågan ha sett ut enligt kodexemplet nedan:

Kodexempel 1. Databasfråga

```
SELECT p.pid, p.projektnamn, p.ordernummer, p.beskrivning, k.kid, k.epost, k.kontaktperson, k.telefon FROM projekt p AND kunder k WHERE p.pid = $pid
```

Vyn v_project_detail har alltså skapats i databashanteraren genom att köra denna databasfråga och spara den som en vy.

Ett ER-diagram över databasen kan ses i Figur 3.



Figur 3. ER-diagram över programmets databas.

Tabellerna anstallda, kunder, maskiner, material och projekt innehåller alltså information om respektive sak. Tabellerna anstallda_i_projekt, maskiner_i_projekt och material_i_projekt kopplar ihop de olika tabellerna så att man kan länka en viss anställd, maskin eller material till ett projekt. Man kan sedan i programmet fylla i hur mycket material som gått åt, hur mycket maskintid som använts eller hur många timmar en viss anställd har jobbat. Kopplingen ”kunder_i_projekt” är ingen egen tabell utan beskriver enbart kopplingen mellan projekt och kunder. Projekt har istället en egen kolumn för kundID eftersom ett projekt enbart kan ha en kund.

5.1 Arbete med databasen

Vår databas byggde vi ursprungligen upp i ett program kallat MySQL Workbench. Där skapade vi färdigt de kopplingar som behövdes tabellerna emellan. Det enda vi inte skapade i Workbench var relationstabellerna anstallda_i_projekt, maskiner_i_projekt, material_i_projekt och anstallda_extra. Dessa kopplar ihop två tabeller då en användare gör ett inlägg som arbetstimmar, använd maskintid eller material som gått åt. Dessa tabeller behövs för att man skall kunna mata in flera värden för ett projekt och per anställd. Undantaget är kunder i projekt, eftersom ett projekt enbart kan ha en kund. I detta fall behövs ingen relationstabell eftersom kunden kopplas till projektet med en främmande nyckel i projekttabellen. I relationstabellerna har varje inlägg ett unikt ID. De övriga kolumnerna innehåller foreign keys som kopplar ihop inlägget med rätt projekt, anställd och material/maskin samt egna kolumner för tabellspecifik information. Tabellen timmar_i_projekt innehåller foreign keys för projekt och anställda samt information om inlagda timmar och datum.

5.2 Cascade-radering

Då relationerna mellan tabellerna i databasen var gjorda återstod ännu frågan om hur raderingen skulle skötas. Om man raderar ett projekt ur projekttabellen i databasen bör även alla kopplingar till projektet raderas från de övriga tabellerna. Ifall detta inte görs återstår en massa rader i relationstabellerna som inte har någon koppling och som inte fyller någon som helst mission. I en databas vill man ju inte lagra skräpdata som bara tar utrymme och gör databasens övergripande prestanda sämre.

För att ta ett praktiskt exempel på hur detta skulle påverka programmet tänker vi oss att man vill radera ett projekt. Då projektet raderas ska även projektets data från relationstabellerna raderas från databasen. Ifall programmet inte raderar den inmatade informationen från relationstabellerna blir det kvar inlagda timmar, material och maskintid som inte är kopplade till något projekt. På lång sikt betyder detta att databasens filstorlek blir större än den skulle behöva vara och att sökningar i de tabeller som innehåller överbliven data tar längre att utföra.

Raderingen utförs genom att en administratör raderar ett projekt via programmet. Då kör ett skript en databasfråga som på basis av projektets ID raderar projektet från projekttabellen. Databasen har ställts in för att granska de tabeller där en främmande nyckel är kopplad till projektet. Ifall en främmande nyckel med samma värde hittas, raderas även alla data på den raden. Detta betyder att alla inlagda timmar, all maskintid och använt material som är länkade till det projekt man raderar tas bort ur databasen. Tekniken som används för detta kallas ”ON DELETE CASCADE” och finns inbyggt i databashanteraren.

6 Användarhantering och roller

Användarna kan indelas i två grupper, administratörer och användare. Administratören skall ha tillgång till all information som systemet har att erbjuda medan användarna endast skall kunna mata in sin egen information, d.v.s. arbetstimmar, materialåtgång, maskintid och övrig löneinformation. I användargränssnittet ser administratörerna alla tabeller, kan lägga till och ta bort innehåll och redigera både sitt och andras innehåll. För användarnas del syns enbart det som de behöver för att fylla i de saker de har tillgång till. Då man skapar en användare kan man välja om man vill skapa en administratör eller vanlig användare. På basis av detta sparas värdet ”admin” eller ”anstalld” i en kolumn i anstallda-tabellen. Detta värde används senare då programmet autentiserar användaren.

6.1 User-klassen

I början av varje fil som laddas skapas en ny instans av User-klassen. I filen User.php finns funktioner som är relaterade till kontroll av användarens id, namn, roll och rättigheter. I detta kapitel beskrivs hur programmet hanterar åtskiljningen av de olika rollerna med olika användarrättigheter.

Då man loggar in körs en kontroll av uppgifterna man matat in mot databasen och variablerna $role, $uid och $name tilldelas sina värden på basis av det användarkonto man loggar in med. Efter detta startas sessionen och sessionsvariabler för roll, användarens ID och namn skapas. Dessa sessionsvariabler kan sedan användas av programmet för att köra kontroller då det behövs. I Kod 2 visas hur sessionen startas och sessionsvariablerna skapas. Dessa sessionsvariabler töms först när en användare loggar ut eller när sessionen avslutas eftersom de ska vara tillgängliga under hela den tid användaren är inloggad. En session förstörs automatiskt efter 24 minuter ifall man inte rört systemet under denna tid och användaren måste logga in på nytt.

Kod 2. Startandet av session då man loggar in

```
session_start();
$_SESSION['username'] = htmlspecialchars($username);
$_SESSION['role'] = $role;
$_SESSION['uid'] = $uid;
$_SESSION['name'] = $name;
```

Den första kontrollen som körs finns i filen page.tpl.php och kollar ifall den inloggade är en administratör eller vanlig användare med hjälp av funktionerna $user->isAdmin(),$user->isUser() och checkUserRole(). På basis av detta skrivs de olika knapparna och länkarna ut i gränssnittet. En administratör har, som tidigare påpekats, flera alternativ i gränssnittet än en vanlig användare.

I Kod 3 visas funktionen checkUserRole(). Här används de sessionsvariabler som skapades i Kod 2 för att kontrollera användarens roll och returnerar antingen värdet admin eller anstalld. Denna funktion anropas i funktionerna isAdmin() och isUser() som nämndes i föregående stycke.

Kod 3. Kontroll av användarens roll

```
function checkUserRole()
    {
        if ($_SESSION['role'] == 'admin')
        {
            $username = $_SESSION['username'];
            $password = $_SESSION['password'];
            $result   =   mysql_query("SELECT   *   FROM   anstallda   WHERE
anvandarnamn = '$username' AND losenord = '$password'");
            if (mysql_num_rows($result))
            {
                while ($row = mysql_fetch_array($result))
                {
                    return $row['roll'];
                }
```

```
            }
        }
        else
        {
            return 'anstalld';
        }
    }
```

Kod 4 visar funktionen isAdmin(). Funktionen isUser() är uppbyggd på samma sätt men admin är ersatt av user. Här körs alltså funktionen checkUserRole() från Kod 3 inne i funktionen.

Kod 4. Kontroll ifall användaren är administratör

```
function isAdmin()
{
      if ($this->checkUserRole() == 'admin') return 1;
      else return 0;
}
```

Då dessa funktioner körts kan man bara använda en enkel villkorssats som if ($user->isAdmin()) eller if ($user->isUser()) för att mata ut olika innehåll för administratörer och användare.

En annan funktion som är relaterad till kontroll av användarens roll är $user->authenticate(). Då det i programmet finns sidor som endast administratörer skall komma åt är det viktigt att användare inte på något sätt kan komma åt dem. Fastän de anställda inte har länkar till dessa sidor i programmet kan man komma åt dem genom att skriva in direktlänken i adressfältet i webbläsaren. För att lösa detta finns det i början av varje fil som enbart administratörer har åtkomst till, en kontroll av användarens rättigheter. I Kod 5 ser man funktionen som kallas.

Kod 5. Kontroll av användarens roll och omdirigering ifall ej administratör

```
function authenticateUser()
{
      if   (isset($_SESSION['username'])   &&   $this->checkUserRole()   ==
      'admin'  ||  isset($_SESSION['username'])  &&  $this->checkUserRole()
      == 'anstalld')
      {
            $username = $_SESSION['username'];
            $password = $_SESSION['password'];
            $result = mysql_query("SELECT * FROM anstallda WHERE
            anvandarnamn = '$username' AND losenord = '$password'");
            if (mysql_num_rows($result)) return true;
            else header('Location: /rasmet/actions/login.php');
      }
```

```
        else if (isset($_SESSION['username']) && $this->checkUserRole() !=
        'admin' || isset($_SESSION['username']) && $this->checkUserRole()
        != 'anstalld')
        {
              $username = $_SESSION['username'];
              $password = $_SESSION['password'];
              $result = mysql_query("SELECT * FROM anstallda WHERE
              anvandarnamn = '$username' AND losenord = '$password'");
              if(mysql_num_rows($result)) eader('Location: ?a=restricted');
              else header('Location: /rasmet/actions/login.php');
        }
        else header('Location: /rasmet/actions/login.php');
}
```

Här kollas om man är administratör eller inte. Ifall man är administratör händer inget förutom att sidan laddas, men är man en vanlig användare omdirigeras man till en sida som berättar att man inte har behörighet att se innehållet på sidan man försökte nå. Denna funktion körs genom att skriva $user->authenticate(); då man vill använda den. Det är viktigt att funktionen körs först, eftersom man inte vill att något skall hända eller något innehåll visas förrän det har kontrollerats att användaren har rätt till informationen. En annan orsak till att authenticate() måste köras först är att den modifierar headerinformation och det inte fungerar ifall något skrivs ut tidigare i koden. Funktionen $user->authenticate() körs alltså i början av varje fil som enbart administratörer har tillgång till.

7 Inloggning

Vårt inloggningsformulär består av ett vanligt HTML-formulär som är strukturerat med en tabell och CSS. Då man kommer till inloggningssidan finns en javascriptfunktion som placerar markören i användarnamnsfältet med koden <body onLoad="document.forms.login.username.focus()">.

Figur 4. Inloggningsformulär.

Själva inloggningsformuläret byggs upp i actionsfilen login-form.php. Inloggningens utseende bestäms av en egen CSS-fil vid namnet login.css.

7.1 Säkerhetsfunktioner

Vi ville skapa en säker inloggningsfunktion. Därför görs ett antal kontroller på den information man matar in i inloggningsrutorna. Exemplet nedan beskriver en inloggningfunktion utan säkerhetsunktioner och hur någon kan utnyttja detta för att bryta sig in i systemet.

I kodexemplet kontrolleras inte lösenordet som användaren matat in. Användaren kan fritt skriva in vad som helst. Ifall användaren matar in user som användarnamn och ' OR "=' som lösenord skulle koden för inskickandet av informationen se ut enligt Kodexempel 2.

Kodexempel 2

```
$_POST['username'] = 'user';
$_POST['password'] = "' OR ''='";
```

Kodexempel 3 kontrollerar om användarnamnet finns i databasen genom att se om databasfrågan returnerar något svar.

Kodexempel 3

```
$query = "SELECT * FROM users WHERE user='{$_POST['username']}' AND
password='{$_POST['password']}'";
mysql_query($query);
```

Om man skrev in ett användarnamn som existerade och angav lösenordet ' OR "=' skulle databasfrågan som sedan skickas in alltså se ut som Kodexempel 4.

Kodexempel 4

```
SELECT * FROM users WHERE user='user' AND password='' OR ''=''
```

Ifall man är bekant med MySQL-kommandon kan man se att detta innebär att vilken användare som helst kan logga in utan ett lösenord.

(W3Schools.com, 1999-2011; PHP.net, 2001-2011)

7.1.1 Kryptering och borttagning av specialtecken

För att göra inloggningen säker kontrollerar vi först den inmatade informationen med mysql_real_escape_string() som lägger till tecknet ”\” framför \x00, \n, \r, \, ', " och \x1a. Sedan tar funktionen strip_tags() bort alla HTML- och PHP-taggar från det man matat in. Vi använder oss även av preg_replace() funktionen för att ersätta vissa ord som vi själva specifierar. Hur koden är uppbyggd kan ses i Kod 6 nedan. Koden i exemplet berör endast användarnamn, men samma kontroller görs även på lösenordet.

Kod 6. Kod som förhindrar kodinjektion

```
SELECT * FROM users WHERE user='user' AND password='' OR ''=''
$username = mysql_real_escape_string($_POST['username']);
$username = strip_tags($username);
$replace = array("/from/i", "/select/i", "/delete/i",
"/update/i","/union/i","/insert/i","/drop/i","/http/i","/--/i");
$username = preg_replace($replace, "", $username);
```

För att förhindra att någon skall kunna komma åt lösenorden i databasen görs en tredubbel kryptering som använder två olika krypteringsalgoritmer, sha1 och md5. Kod 7 nedan gör det lättare förstå hur krypteringen fungerar.

Kod 7. Saltning och kryptering av lösenord

```
$salt = sha1(md5($_POST['password']));
$password = md5($_POST['password'] . $salt);
```

I Kod 7 skapas först ett så kallat salt. Saltet skapas här genom att först kryptera lösenordet med md5-krypteringsalgoritmen samt att sedan kryptera det redan krypterade lösenordet med en sha1-algoritm. Efter detta md5-krypteras lösenordet och det nyligen skapade saltet och bildar tillsammans en 128-bitars sträng. Detta händer så klart både då man skapar ett nytt lösenord och då man loggar in, eftersom det inskrivna lösenordet måste kunna jämföras med det sparade lösenordet i databasen.

Lösenordskrypteringen är alltså en säkerhetsåtgärd för att försäkra sig om att inga lösenord står i klartext ifall någon skulle få tillgång till Raseborgs Metalls databas. För att få reda på lösenordet som ligger bakom den krypterade strängen måste man alltså godtyckligt prova ett lösenord och köra det igenom samma tredubbla kryptering och sedan jämföra det med de sparade lösenorden i databasen.

Nedan fortsätter vår login-funktion efter att man skickat in sina inloggningsuppgifter.

Kod 8. Start av session och omdirigering efter inloggning

```
if (mysql_num_rows($result))
{
	session_start();
	$_SESSION['username'] = htmlspecialchars($username);
	$_SESSION['role'] = $role;
	$_SESSION['uid'] = $uid;
	header('Location: ../index.php');
}
else
{
	$_POST['credentials'] = 'incorrect';
}
```

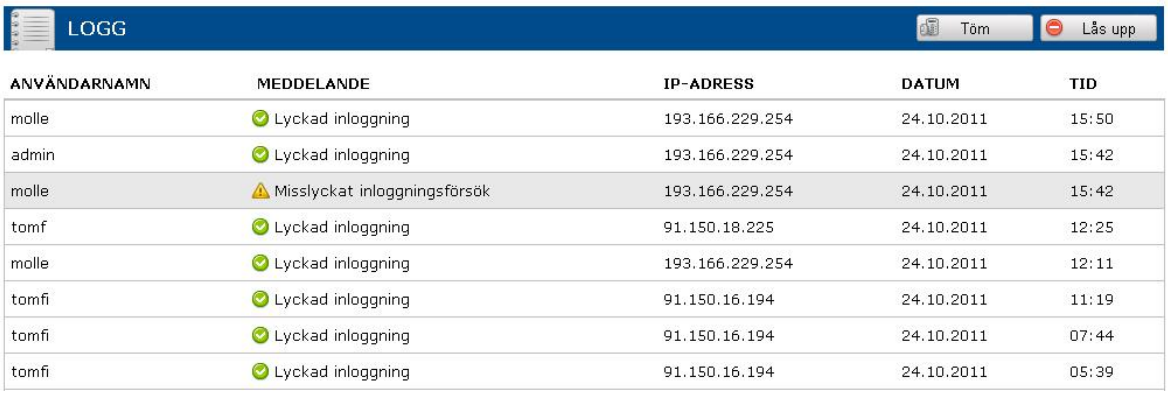
Den första if-satsen kollar om motsvarighet till den angivna inloggningsinformationen hittas från användartabellen i databasen. Om informationen är korrekt kommer man in i systemet, annars får man fylla i informationen på nytt. Om en användare skriver in fel användarnamn eller lösenord visar inloggningsformuläret ett felmeddelande som berättar att användaren matat in felaktig information.

7.1.2 Begränsning av inloggningsförsök

På grund av att det finns automatiserade robotar som kan försöka bryta sig in i systemet har vi valt att begränsa inloggningsförsök till fem försök per timme. För att göra detta kan man använda sig av ett antal olika metoder. Några av dessa är cookies, sessioner och databasloggning. Vi började med att undersöka vilken av metoderna som skulle vara bäst lämpad för vårt system. Med hjälp av en session kan man relativt enkelt blockera en robot eller användare efter ett visst antal försök. Denna metod är dock inte vattentät eftersom det räcker med att stänga webbläsaren för att få försöka på nytt. Cookies sparar en fil på användarens dator och denna fil granskas varje gång användaren försöker logga in. På detta sätt kommer systemet ihåg användaren fast webbläsaren har stängts och öppnats på nytt. Dock kan användaren eller roboten radera denna fil för att få ytterligare inloggningsförsök. Efter att ha undersökt de olika metoderna som står till buds kom vi fram till att vi måste logga varje inloggningsförsök i en databas och begränsa inloggningsförsöken för varje ip-adress. På detta sätt kan användaren inte kringgå granskningen.

7.2 Loggning av inloggningsförsök

Då en användare loggar in eller misslyckas med inloggningen sparas uppgifterna i en tabell i databasen. Från databasen kan programmet sedan hämta ut information som administratörerna i programmet kan komma åt. Detta görs för att administratörerna skall kunna granska inloggningsförsök och vara medvetna om ifall någon försöker bryta sig in i systemet. Figur 5 visar listningen av inloggningsförsök.



ANVÄNDARNAMN	MEDDELANDE	IP-ADRESS	DATUM	TID
molle	Lyckad inloggning	193.166.229.254	24.10.2011	15:50
admin	Lyckad inloggning	193.166.229.254	24.10.2011	15:42
molle	Misslyckat inloggningsförsök	193.166.229.254	24.10.2011	15:42
tomf	Lyckad inloggning	91.150.18.225	24.10.2011	12:25
molle	Lyckad inloggning	193.166.229.254	24.10.2011	12:11
tomfi	Lyckad inloggning	91.150.16.194	24.10.2011	11:19
tomfi	Lyckad inloggning	91.150.16.194	24.10.2011	07:44
tomfi	Lyckad inloggning	91.150.16.194	24.10.2011	05:39

Figur 5. Översikt over inloggningsförsök

I listningen kan administratören se användarnamnet, IP-adressen från vilken dator inloggningsförsöket gjordes samt datum och tid. Ifall man i denna lista har ett flertal

upprepade inloggningsförsök som misslyckats är det troligtvis inte en användare som försökt logga in utan någon som försöker bryta sig in i systemet. För att förhindra attacker av detta slag har vi utvecklat säkerhetsfunktionen som beskrevs i föregående kapitel.

För att bygga upp listan används tre separata filer, log.php från actions, Log.php och Table.php från classes.

Kod 9. Skapande av instans för logg

```
$log = new Log();
$log->setViewName('login');
$pager->setRowsPerPage(20);
```

I Kod 9 från log.php skapas en ny instans vid namn Log. Här bestäms också setViewName() som får värdet login. $pager bestämmer hur många rader som visas per sida och använder sig av pagineringsfunktionen.

För att få ut informationen ur databasen används funktionen getRows() i Table-klassen. I denna funktion finns en villkorssats som kollar om det handlar om loggning av inloggningsförsök och kör en viss databasfråga ifall villkoret uppfylls. Kod 10 beskriver getRows() som används vid hämtning av inloggningsförsök

Kod 10. Funktionen getRows()

```
function getRows($view, $offset, $rowsPerPage)
{
	if ($view == 'login') $query = "SELECT * FROM $view ORDER BY datum
	DESC LIMIT $offset, $rowsPerPage";
	else $query = "SELECT * FROM $view LIMIT $offset, $rowsPerPage";
	$result = mysql_query($query);
	if ($result)
	{
		while ($row = mysql_fetch_array($result))
		{
			$output[] = $row;
		}
	}
	return $output;
}
```

Variabeln $view har alltså fått värdet login, offset fås från Pager-klassen och rowsPerPage sätts till 20 i Kod 9. Offset bestämmer vilka sökresultat som visas på vilken sida då man har fler än 20 stycken. De första 20 visas alltså på sida ett och de 20 följande på sida två. Hur offset och rowsPerPage fungerar förklaras mer detaljerat i kapitlet om pagineringsfunktionen.

För att funktionalitet från Table-klassen ska kunna återanvändas används den inbyggda PHP-funktionen ”extends”. Kodexemplet nedan visar hur en fil som använder extends-funktionen börjar.

Kodexempel 5: Klassen Log kan använda sig av Tableklassens funktioner

```
class Log extends Table
```

Därefter används funktionen getRowsTable() som skapar en array av den hämtade informationen med koden i Kod 11.

Kod 11. Skapande av array med getRows

```
$array = $this->getRows($this->getViewName(), $offset, $rowsPerPage);
```

Denna information skrivs sedan ut i en tabell och formateras på basis av om inloggningsförsöket lyckades eller misslyckades.

8 Användargränssnitt och inmatning

Själva användargränssnittet för programmet är uppbyggt med HTML och CSS. Vi strävar efter att skapa ett visuellt intryck av ett program och inte en hemsida, trots att systemets användargränssnitt är helt webbaserat. I detta arbete behandlas endast användargränssnittet för administratörer eftersom det även innehåller all den information som vanliga användare ser.

Det viktigaste kriteriet vad gäller användargränssnittet för kundens del är att det är lätt att förstå och att inmatningen av information är så logisk som möjligt. Systemet skall även fungera på två språk.

8.1 Navigering

Navigeringen i programmet sker med hjälp av en menyrad där man hittar de olika kategorierna av information. I denna menyrad finns också direktlänkar för att mata in arbetstimmar och lägga till material och maskintid som använts i ett projekt. I Figur 6 nedan ser man en översikt av menyn då man är inloggad som administratör.

Figur 6. Menyrad för administratörskonto.

8.2 Skapande och redigering

Då man vill redigera innehåll av någonting trycker man helt enkelt på objektets namn. I Figur 7 nedan kan man se Projektöversikten. Då det finns många projekt i översikten kommer projekten att pagineras. Detta innebär att endast 20 projekt visas per sida.

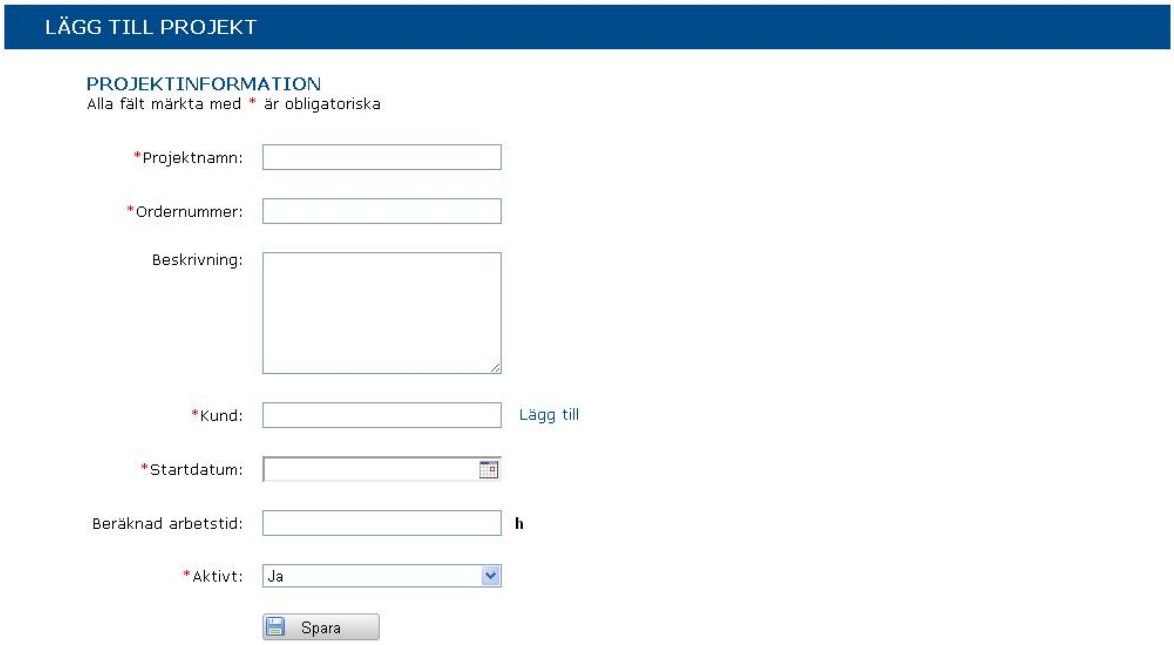
Figur 7. Projektöversikt med ett resultat

Notera att projektets namn är utmatat som en länk. Trycker man på denna får man upp en detaljvy som innehåller all info kopplat till projektet. För att lättare hitta det man är ute efter delas informationen upp i olika kategorier i detaljvyn.

Figur 8. Detaljvy av projekt.

I figuren ovan syns en del av detaljvyn för projektet. Under detta finns i programmet även rutor för beställare, materialåtgång och maskintid som använts i projektet. Vill man redigera projektinformationen trycker man på knappen ”Redigera projekt”.

Redigering och skapande av nytt projekt använder samma formulär så vi har valt att endast visa formuläret för skapandet av ett nytt projekt. Om man redigerar ett projekt ändrar man på informationen i fälten istället för att fylla i information i tomma fält. I Figur 9 ses formuläret för skapande av ett nytt projekt.



Figur 9. Skapande av project.

Då man skapar ett nytt projekt finns det några funktioner som är speciellt intressanta. För det första har vi så klart själva funktionen som sköter om att inmatade data sparas i databasen, men även funktionen för att söka upp och välja rätt kund och granskningen av inmatad information är värda att se på.

Koden i actions-filen för skapande av projekt är ganska simpel och består enbart av kontroll av användarens roll, skapande av en instans för det nya projektet och utskrift av formuläret för inmatning av data. Själva formuläret byggs upp i projektklassen och tas in i actions filen genom $output .= $project->addEditForm();. Innehållet i $output tas sedan in i index.php och läggs till i variabeln $content. $content skrivs sedan ut i filen page.tpl.php. Funktionen addEditForm är en rätt så lång funktion för uppbyggnaden av

informationsinmatningen både då man skapar nya projekt och då man redigerar ett gammalt. Formuläret som skrivs ut byggs upp i filen Project.php, som är en klassfil. I denna fil granskas om det är frågan om ett befintligt projekt eller skapandet av ett nytt och specificeras vad som skrivs ut på basis av om projektet finns eller inte. Då man skapar eller redigerar projekt körs även en validering av informationen. Denna funktion kommer att förklaras i sin helhet senare i arbetet.

Då man kommer till sidan skrivs formuläret för inmatning av information ut. När man matat in informationen i formuläret och tryckt på Spara-knappen skickas informationen vidare till valideringsfilen. Ifall informationen går igenom valideringen körs funktionen addProject(). Då det handlar om editering av ett befintligt projekt körs istället funktionen updateProject(). Själva sparandet till databasen sköts av just funktionerna addProject() och editProject(). I Kod 12 visas den del av funktionen som sparar den inmatade informationen i databastabellen:

Kod 12: Sparande av information I databasen

```
function addProject($projektnamn, $ordernummer, $kunder_kid,
$beskrivning, $startdatum, $aktivt)
{
	$result = mysql_query("INSERT INTO `projekt` (`projektnamn`,
	`ordernummer`, `kunder_kid`, `beskrivning`, `datum`, `aktiv`)
	VALUES ('$projektnamn', '$ordernummer', $kunder_kid,
	'$beskrivning', '$startdatum', '$aktivt')");
}
```

9 Initieringsfiler

Då programmet körs igång inkluderas vissa filer som hjälper att bygga upp funktionaliteten. Eftersom dessa filer är kritiska vid uppstartandet av programmet kallas de till initieringsfiler. I detta kapitel kommer de viktigaste initieringsfilerna och deras funktionalitet att beskrivas.

9.1 Index.php

Filen index.php är den första fil som kallas på då man loggat in. Försöker man komma åt index-filen utan att vara inloggad blir man omdirigerad till inloggningssidan med hjälp av villkorssatsen i Kod 13.

Kod 13. Kontroll ifall session med användarnamn finns

```
if (!isset($_SESSION['username'])) header('Location: actions/login.php');
```

I indexfilen inkluderas också två andra filer, settings.php och bootstrap.php. Filen settings.php innehåller inloggningsinformation till databasen och bootstrap.php inkluderar filer i classes- och includesmappen. Filerna i includes-mappen innehåller funktioner som kan användas av programmet. När man inkluderar filer betyder detta att man kan anropa och utnyttja funktioner deklarerade i dem.

Funktionen split_action_path() hämtas från filen function-action.inc i includes-mappen. Med hjälp av den ser index.php till att rätt actions-fil inkluderas beroende på var i programmet man befinner sig och vad man gör. Fastän koden kan verka aningen svår att förstå är funktionaliteten bakom den relativt enkel.

Då man är inloggad körs split_action_path() varje gång en sida laddas. Ifall adressbalken innehåller en adress med ”?a=” betyder det att $_GET[’a’] innehåller allting som kommer efter lika med-tecknet, ifall detta inte bryts med ett &-tecken.

Funktionen split_action_path() granskar värdet för $_GET[’a’] och delar det vid varje snedstreck (”/”). En lista (array) vid namn $a skapas och den får värdena från adressfältet där a[0] består av den som står efter ”?a=”, a[1] av det som står efter det första snedstrecket osv. Adressen www.raseborgsmetall.fi/?a=project/edit/8 skulle alltså delas på följande sätt:

a[0] = project
a[1] = edit
a[2] = 8

Kod 14 visar funktionen split_action_paths() uppbyggnad.

Kod 14. Funktionen split_action_path

```
function split_action_path()
{
	$a = $_GET['a'];
	$a = split('/',$a);
	return $a;
}
```

I index.php granskas vilken fil som skall inkluderas baserat på vad arrayen ”$a” innehåller. Vår array i exemplet innehåller alltså ”project”, ”edit” och ”8”. If-satsen i Kod 15 granskar värdena för arrayen $a. Om ”$a[1]” innehåller ”add” eller”edit” och ”$a[2]” inte är tom körs koden innanför krullparenteserna. I detta fall tilldelar den variabeln ”$file” värdet ”./actions/projekt-edit.php”. Detta är alltså sökvägen till den fil som ska inkluderas i detta skede. Efter if-satsen i slutet av Kod 15 körs sedan en require_once-funktion med den sökväg som hämtats ut med hjälp av funktionen split_action_path. Denna inkluderar den rätta filen för just denna situation.

Kod 15. Kontroll av vilken fil som skall inkluderas

```
if ($a)
{
	$a = split_action_path();
	if ($a[1] == 'add' || ($a[1] == 'edit' && $a[2] != ''))
	{
		$file = './actions/' . $a[0] . '-' . $a[1] . '.php';
	}
	if (file_exists($file))
	{
		require_once $file;
	}
}
```

Funktionen require_once('templates/page.tpl.php'); anropas efter att if-satsen har körts igenom för att granska ifall filen page.tpl.php har blivit inkluderad. Om inte, inkluderar den även den filen.

9.2 Bootstrap.php

I föregående stycke nämndes att filen bootstrap.php inkluderar filer i classes- och includesmapparna. I detta kapitel förklaras hur detta åstadkoms.

Det första som sker är att en array blir tilldelad alla filnamn i respektive mapp. Sedan gås filerna i mapparna igenom och inkluderas ifall de är av rätt filtyp. Som ett exempel inkluderas enbart filer som slutar på .php från mappen classes och enbart filer som slutar på .inc från mappen includes. Ifall det finns övriga filer i mapparna ignoreras dessa av funktionen. I Kod 16 kan man se hur funktionen för inkluderandet av includes-filer är uppbyggd.

Kod 16. Inkludering av includes-filer

```
$includes_dir = 'includes/';
$handler = opendir($includes_dir);
while ($file = readdir($handler))
{
```

```
        if ($file != '.' && $file != '..')
        {
                if (substr($file, -4) == '.inc')
                {
                        require_once $includes_dir . $file;
                }
        }
}
```

9.3 Settings.php

Det har tidigare nämnts att settings.php innehåller inloggningsinformationen till databasen. I detta kapitel gås kort igenom hur den är uppbyggd. Filen består av enbart 4 rader där variabler tilldelas information om användarnamn, lösenord, databasnamn och databasadress. I Kod 17 ser man hur detta åstadkoms. Informationen har här ersatts av x på grund av säkerhetsskäl.

Kod 17. Settings.php

```
$db_user = 'xxxxxxxxxxx';
$db_pass = 'xxxxxxxx';
$db_name = 'xxxxxxxxxxx';
$db_url = 'localhost';
```

Denna information används av filen db.inc för att kontakta databasen. Filen db.inc är belägen i includes-mappen.

10 Meddelandefunktionen

Då en användare har glömt att mata in information i ett obligatoriskt fält eller då programmet vill meddela eller varna om något ser användaren en meddelanderuta som beroende på meddelandets natur är antingen röd eller grön. För meddelanden då något lyckats är rutan grön och då något misslyckats eller programmet vill varna för något är den röd. Hela meddelandesystemet sköts med hjälp av tre funktioner, message(), messageClass() och getMessage().

För att det skall bli lättare att förstå kommer funktionen att beskrivas i logisk ordning. Vi vet att filen som hanterar meddelanden inkluderas och kan anropas i praktiskt taget vilken fil som helst. Funktionen anropas genom att skriva in koden i Kodexempel 6.

Kodexempel 6

```
message('Text att visa i meddelandet ', 0);
```

Funktionen som anropas heter alltså message(). I parentesen som följer skrivs meddelandet samt en nolla eller etta in. Siffran som följer meddelandet används för att bestämma vilken färg meddelanderutan skall ha. I programmet används oftast en if-sats för att kolla upp ifall vissa villkor uppfylls eller inte och på basis av detta skrivs ett meddelande ut.

Själva funktionen message() använder sig av variablerna $message och $tone. För att programmet inte skall tömma variablernas innehåll då man gör en omdirigering (PHP Redirect) använder vi oss av en session för att spara värdena. För dem som är obekanta med PHP görs en omdirigering enligt Kodexempel 7 nedan.

Kodexempel 7

```
header( 'Location: http://www.exempeladress.com/ny_sida.html' ) ;
```

I Kod 18 ser man koden för funktionen message(). Här tilldelas variablerna $message och $tone informationen som angetts innanför parenteserna i Kodexempel 6. Efter detta skapas en session vid namn message som får värdet av variabeln $message. En villkorssats kollar därefter upp om $tone har värdet 1 eller 0 och på basis av detta skapas en session vid namn tone som får värdet som sparats i $tone.

Kod 18. Funktionen message

```
function message($message, $tone)
{
	$_SESSION['message'] = $message;
	if ($tone == 1)  $_SESSION['tone'] = 1; else $_SESSION['tone'] = 0;
}
```

Funktionen messageClass() kontrollerar variabeln $tone och printar antingen message-positive eller message-negative. Detta används för att bestämma vilken klass div-elementet som omger meddelandet i programmet får. I Kod 19 visas hur messageClass är uppbyggt.

Kod 19. Funktionen messageClass

```
function messageClass($tone)
{
	if ($tone == 1) print 'message-positive'; else print 'message-
	negative';
}
```

Hittills finns ingen funktion där själva meddelandet skrivs ut, utan vi har enbart sparat värden och bestämt vilken klass meddelandets omgivande div-element får. Funktionen getMessage i Kod 20 visar hur meddelandet skrivs ut då funktionen anropas.

Kod 20. Funktionen getMessage

```
function getMessage()
{
	print $_SESSION['message'];
}
```

Nu har alla de funktioner som används vid utskrift av ett meddelande gåtts igenom. Dessa bygger upp funktionaliteten men ännu har inget meddelande skrivits ut i programmet. Kod 21 visar hur meddelandet skrivs ut då villkoren uppfylls.

Kod 21. Utskrift av meddelande

```
if (isset($_SESSION['message']) && isset($_SESSION['tone'])) { ?>
<div class="<?php if (isset($_SESSION['tone'])) print
messageClass($_SESSION['tone']); else print 'message'; ?>">
<?php if (isset($_SESSION['message']))
{
	getMessage();
	unset ($_SESSION['message']);
	unset ($_SESSION['tone']);
}
```

För att meddelanderutan skall visas måste det finnas en sessionsvariabel med meddelande och en med tone. När dessa finns skrivs ett div-element ut som får klassen message-positive eller message-negative. Efter detta anropas funktionen getMessage som skriver ut meddelandet i div-elementet. För att detta meddelande inte skall hänga med då man byter sida i programmet töms sessionsvariablerna message och tone alltid efter att funktionerna körts igenom.

11 Table-klassen

Table är klassen som sköter om tabellvisningen i programmet. Eftersom tabeller visas i en mängd olika former och innehållande olika information beroende på situationen, behöver detta specificeras någonstans. En av funktionerna sköter hämtning av tabelldata och är giltig för alla tabeller innan några filter har applicerats. En annan funktion tar hand om hämtning av informationen i tabellerna efter att ett sökfilter har applicerats. Denna används

av alla andra klasser förutom projektklassen och när information angående lön för arbetarna ska visas. Projekt, arbetstimmar och extrainfo har sina egna funktioner för applicerandet av sökfilter eftersom man måste kunna filtrera dessa sökresultat också enligt datum.

11.1 Hämtning av tabellinformation

I Kod 22 visas funktionen getRows som anropas då man vill hämta ut information ur en databastabell. Variabeln $view får sitt värde tidigare i actions- och klassfilerna för respektive objektklass. På basis av denna variabel vet funktionen vilken information som skall visas. I variabeln $query sparas databasfrågan. Sedan kollas om databasfrågan returnerar något värde. Ifall ett värde returneras körs while-satsen. I while-satsen sparas sedan informationen i en array för att sedan returneras rad för rad i textform på skärmen av return $output.

Kod 22. Funktionen getRows

```
function getRows($view)
{
	$query = "SELECT * FROM $view";
	$result = mysql_query($query);
	if ($result)
	{
		while ($row = mysql_fetch_array($result))
		{
			$output[] = $row;
		}
	}
	return $output;
}
```

11.2 Hämtning av information då ett sökfilter appliceras

För att man skall kunna söka efter projekt, kunder, anställda, material och maskiner har vi utvecklat en sökfunktion till sidorna där de listas. För kunder, anställda, material och maskiner består sökfältet av en enkel sökruta där man matar in text, medan man i projektkategorins sökfunktion även har möjlighet att specificera ett start- och slutdatum. Vi kommer i detta kapitel att gå igenom sökfunktionen för de olika klasserna. Figur 10 visar hur sökfältet för projektkategorin visas för användare.

Figur 10. Sökformuläret för projektklassen

Sökfunktionens formulär byggs upp i respektive klassers actions-filer. För projektklassen betyder detta alltså att formulärkoden ligger i filen project.php i actions-mappen. När man matat in kriterier för sökningen och trycker på Sök tilldelas variabeln $keyword den text man matat in i fritextsökningsfältet. Variablerna skickas sedan till funktionen getSearchTable() i Project-klassen som skickar vidare informationen till funktionen getSearchRows() i Table-klassen. I den funktionen byggs databasfrågan upp på basis av den inmatade informationen. Här skapas också en array av informationen som hämtats ur databasen. Arrayen kan sedan utnyttjas av getSearchTable() i Project-klassfilen för att bygga upp visningen av informationen på sidan. För projektsökfunktionen sparas även startdatumet i variabeln $start och slutdatumet i $finish.

Kod 23. Funktionen getSearchRows

```
function getSearchRows($table, $keyword)
{
	$res = mysql_query("DESCRIBE $table");
	$num_rows = mysql_num_rows($res);
	$i = 0;
	while($row = mysql_fetch_array($res))
	{
		$last_row = $num_rows - 1;
		if ($i < $last_row)
		{
			$columns .= $row[0] . " LIKE '%" . $keyword . "%' OR ";
		}
		$i = $i + 1;
		if ($i == $num_rows)
		{
			$columns .= $row[0] . " LIKE '%" . $keyword . "%'";
		}
	}
	$query = "SELECT * FROM $table WHERE $columns";
	$result = mysql_query($query);
	if ($result)
	{
		while ($row = mysql_fetch_array($result))
```

```
                {
                        $output[] = $row;
                }
        }
        else
        {
                $output = 'Inga resultat';
        }
        return $output;
}
```

Funktionen getSearchRows() i Kod 23 används när man kör en sökning som involverar någon av de andra huvudobjektklasserna förutom projekt. Variabeln $res sparar information om tabellen och $num_rows räknar sedan med hjälp av den inbyggda funktionen mysql_num_rows() ut hur många kolumner tabellen innehåller och sparar vissa värden för varje kolumn. Efter detta deklareras variabeln $i som skall hålla reda på vilken rad i tabellen som gås igenom.

While-satsen körs så länge som det finns rader i $res-arrayen och villkorssatserna inne i den bestämmer att så länge vi inte är på sista raden körs den första if-satsen och adderar 1 till värdet på variabeln $i. När vi kommer in på sista raden körs den andra if-satsen. Detta görs för att förhindra att strängen i $columns-variabeln slutar på OR. Om denna skulle slutat på OR skulle strängen inte kunnat användas i databasfrågan som följer.

Orsaken till att vi enbart använder det första värdet i arrayen i villkorssatserna beror på hur DESCRIBE-funktionen returnerar data. Värdena den tar ut ur databasen är följande: Field, Type, Null, Key, Default och Extra för varje kolumn i tabellen. Vi vill alltså komma åt värdet som ligger i Field för varje kolumn. Detta värde är sparat i arrayens första post. Punkten i $columns .= gör så att varje gång while-satsen körs igenom läggs resultatet till i strängen i $columns istället för att ersätta det gamla. Innehållet i variabeln $columns blir således:

kolumnnamn1 LIKE %inmatat sökord% OR kolumnnamn2 LIKE %inmatat sökord% OR kolumnnamn3 LIKE %inmatat sökord% OR kolumnnamn4 LIKE %inmatat sökord%

Strängen ovan motsvarar fyra genomkörningar av while-satsen. Hur många gånger det upprepas beror på hur många kolumner tabellen innehåller. Denna sträng läggs sedan in i databasfrågan som körs därefter. Denna funktion söker alltså igenom alla kolumner i en viss tabell och matchar innehållet mot sökordet. Funktionen returnerar sedan resultatet i variabeln $output.

För projektklassen körs en liknande funktion vid namn getSearchRowsProject. Den ser i övrigt likadan ut men skapandet av databasfrågan är mycket mer komplex. Här finns alltså tre sökkriterier, sökord, startdatum och slutdatum. Sökfunktionen måste fungera även då bara ett eller två av sökkriterierna angivits. Kod 24 beskriver villkorssatserna i funktionen getSearchRowsProject().

Kod 24. Villkorssatser i getSearchRowsProject

```
if ($start == '' && $finish == '')
{
	$columns .= $row[0] . " LIKE '%" . $keyword . "%' OR ";
}
else if ($start != '' && $finish == '')
{
	$columns .= $row[0] . " LIKE '%" . $keyword . "%' AND startdatum >=
	'" . $start . "' OR ";
}
else if ($start == '' && $finish != '')
{
	$columns .= $row[0] . " LIKE '%" . $keyword . "%' AND startdatum <=
	'" . $finish . "' OR ";
}
else
{
	$columns .= $row[0] . " LIKE '%" . $keyword . "%' AND startdatum
	BETWEEN '" . $start . "' AND '" . $finish . "' OR ";
}
if ($i == $last_row)
{
	if ($start == '' && $finish == '')
	{
		$columns .= $row[0] . " LIKE '%" . $keyword . "%'";
	}
	else if ($start != '' && $finish == '')
	{
		$columns .= $row[0] . " LIKE '%" . $keyword . "%' AND
		startdatum >= '" . $start . "'";
	}
	else if ($start == '' && $finish != '')
	{
		$columns .= $row[0] . " LIKE '%" . $keyword . "%' AND
		startdatum <= '" . $finish . "'";
	}
	else
	{
		$columns .= $row[0] . " LIKE '%" . $keyword . "%' AND
		startdatum BETWEEN '" . $start . "' AND '" . $finish . "'";
	}
}
```

Innan Kod 24 börjar ser funktionen likadan ut som getSearchRows() i Kod 23. Därefter avviker getSearchRowsProject() från getSearchRows() angående hur databasfrågan skapas. Här kollas först om start- och slutdatumfälten lämnats tomma. Då byggs databasfrågan

enbart upp med sökordet som kriterium. Om detta inte är fallet kollas om bara ett av datumfälten har blivit ifyllda. Om detta stämmer byggs databasfrågan upp med sökordet och ett av datumen. Det är i det här fallet ingen skillnad ifall sökordsfältet lämnats tomt, eftersom frågan då enbart filtrerar efter datum. Ifall startdatum för sökningen var ifyllt söker funktionen alla projekt som startat efter detta datum. Om bara slutdatum fyllts i filtrerar funktionen bort alla projekt som startats efter det inmatade datumet. Då användaren fyllt i alla fält kontrolleras alla projekt vars startdatum infaller mellan de angivna start- och slutdatumen och som innehåller sökordet som matats in.

Om villkoret i satsen `if ($i == $last_row)` i Kod 24 uppfylls betyder det att funktionen går igenom den sista raden i arrayen. Koden som följer är egentligen en upprepning av den kod vi redan gick igenom i föregående stycke förutom att den sträng som läggs till i variabeln $columns inte slutar på OR. Som tidigare nämnts kan en databasfråga inte sluta på OR, så detta är ett nödvändigt steg för att försäkra att databasfrågan blir korrekt.

12 Validering

När en användare matar in information i ett formulär ska det finnas vissa fält man måste skriva information i för att man skall kunna sparas. Vi tar som exempel att man skapar ett nytt projekt. I Figur 11 visas hur ett fält ser ut då det är obligatoriskt att mata in information i det.

Figur 11. Ett fält som är obligatoriskt att fylla i.

De fält som krävs för att kunna spara är utmärkta med en röd stjärna. För att man inte skall kunna undvika att mata in information i dessa fält måste det alltså finnas en funktion som kontrollerar om man matat in något eller inte. För skapandet eller editerandet av ett projekt heter funktionen ”validateProject”. För användarna är det även viktigt att de data man tidigare matat in sparas så att man inte är tvungen att till exempel skriva en lång beskrivning på nytt ifall man glömt något av de obligatoriska fälten och tryckt spara.

Detta åstadkoms genom en villkorssats som kollar de olika fälten var för sig. Eftersom funktionen ser lika ut för varje fält visas i Kod 25 enbart den inledande villkorssatsen och kontrollfunktionen för projektets namn.

Kod 25. Kontroll av formulärets fält

```
if ($_POST['projektnamn'] == '' || $_POST['ordernummer'] == '' ||
$_POST['kunder_kid'] == '' || $_POST['startdatum'] == '')
{
      if ($_POST['projektnamn'] == '')
      {
            $projektnamn = '';
            $ordernummer = $_POST['ordernummer'];
            $beskrivning = $_POST['beskrivning'];
            $kunder_namn = $_POST['kunder_namn'];
            $kunder_kid = $_POST['kunder_kid'];
            $startdatum = $_POST['startdatum'];
            $aktivt = $_POST['aktivt'];
            $item .= ' <b>- Projektnamn</b><br /> ';
      }
}
```

Här kontrolleras först ifall något av de olika fälten saknar värde då man tryckt på spara-knappen. Ifall detta stämmer går vi vidare in och kollar om det är fältet för projektnamn som saknar värde. Ifall vi skulle ha inkluderat hela funktionen skulle programmet efter detta kodstycke ha upprepat kontrollen för ordernummer, kunder_kid och startdatum med en likadan villkorssats. Det som händer därefter är att värden från alla fält sparas i variabler för att kunna användas senare då vi ska komplettera informationen i det tomma fältet. Till sist sparas textsträngen ” - **Projektnamn**” i variabeln $item. Detta görs för att man i felmeddelandet skall kunna skriva ut vilken rad som saknar värde.

Kod 26 nedan körs innanför den inledande villkorssatsen. Den kör alltså funktionen addEditFormError() som finns i klassfilen Project.php. Koden ger de värden som sparats tidigare till formuläret i addEditFormError() så att man inte tvingas skriva in alla värden på nytt ifall man glömt något. Denna funktion kallas både när man skapar nya projekt och man editerar ett befintligt. Programmet åtskiljer skapandet och editerandet med en villkorssats som kollar om det finns ett projekt-id. På basis av detta används antingen actions-filen project-add eller project-edit.

Kod 26. Skickandet av ifylld information till addEditFormError

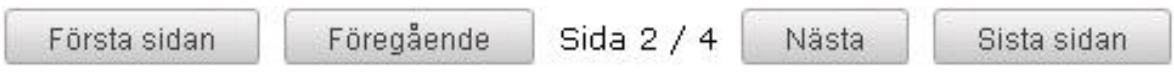
```
if ($pid != '')
{
```

```
        $output     .=     $project->addEditFormError($pid,     $projektnamn,
        $ordernummer, $beskrivning, $kunder_namn, $kunder_kid, $startdatum,
        $aktivt, '?a=project/edit/' . $pid);
        message('Fyll i följande fält:<br /> ' . $item, 0);
}
else
{
        $output     .=     $project->addEditFormError($pid,     $projektnamn,
        $ordernummer, $beskrivning, $kunder_namn, $kunder_kid, $startdatum,
        $aktivt, '?a=project/add');
        message('Fyll i följande fält:<br /> ' . $item, 0);
}
```

Funktionen addEditFormError() skriver sedan ut formuläret för inmatning på nytt, kollar om det finns värden för de olika fälten och skriver in dem i fälten där de hör hemma. Efter det kan man komplettera med den information som uteblev första gången man fyllde i formuläret.

13 Paginering

Då programmet kommer att hantera stora mängder data ville vi utveckla en funktion för att visa data i hanterbara mängder för användaren. Vi vill alltså inte visa alla sökresultat på samma sida ifall det handlar om en större mängd. När vi talar om sökresultat behöver det inte nödvändigtvis handla om en sökning som användaren gjort. En sökning från databasen görs automatiskt exempelvis när användaren trycker på Projektknappen i menybalken. Då söker programmet ut alla projekt från databasen och visar dessa åt användaren. För att en hanterbar mängd information skall visas på skärmen har vi utvecklat en pagineringsfunktion som delar upp sökresultaten på olika sidor. Vi kom överens med kunden att 20 resultat per sida är en bra mängd. Ifall man är på den första sidan visar pagineringsfunktionen hur många sidor sökningen gav, en knapp för att gå till nästa sida och en knapp för att hoppa direkt till sista sidan. Om man inte är på första sidan visas även en knapp för att komma till föregående sida och första sidan. Befinner man sig på sista sidan faller knapparna för nästa sida och sista sidan bort. Hur det ser ut för användaren kan ses i Figur 12 nedan.



Figur 12. Pagineringsfunktionens utseende I gränssnittet.

Pagineringens funktionalitet byggs upp i klassfilen Pager.php. Den första funktionen i Pager.php bestämmer hur många resultat som skall visas per sida. I Kod 27 kan man se hur funktionen är uppbyggd. Hur många rader som visas bestäms alltså av variabeln $rows. $rows bestäms i sin tur skilt i actionsfilen för respektive situation då paginering används. Det betyder att man kan använda olika värden för $rows vid olika tillfällen och därigenom variera hur många resultat som visas per sida beroende på vilken typ av objekt det handlar om.

Kod 27. Funktionen setRowsPerPage

```
function setRowsPerPage($rows)
{
	$this->rowsPerPage = $rows;
}
```

För att initiera en pager och bestämma vilket värde $rows får används två rader kod i actionsfilen. I kod x skapas alltså först en ny instans av pager() och sedan tilldelas setRowsPerPage() sitt värde. Märk att istället för 20 kan man här använda vilket positivt värde som helst. Ifall man vill se flera sökresultat per sida kan man öka på värdet närhelst det behagar.

Kod 28. Skapande av instans för Pager samt bestämmande av resultat per sida

```
$pager = new Pager();
$pager->setRowsPerPage(20);
```

För att programmet ska veta vilka resultat den ska visa på vilken sida finns en funktion som räknar ut en så kallad offset. Ifall vi befinner oss på den tredje sidan och har bestämt att det ska visa 20 resultat per sida betyder det att funktionen ska lämna bort de 40 första resultaten och visa enbart resultaten från 41 till 60. Denna uträkning sköts av en funktion som heter getOffset. Funktionen använder sig av numret på sidan man befinner sig på för att räkna ut vilka resultat som visas. Sidans nummer sparas i adressbalken, så om man befinner sig på den andra sidan kommer adressfältet att bestå av www.adress.fi/?a=klass&page=2. Då adressfältet innehåller page=2 kan man använda GET[’page’]-funktionen för att hämta ut värdet för page. Värdet för page används sedan i uträkningen i funktionen getOffset.

Kod 29. Funktionen getOffset

```
function getOffset($page)
```

```
{
	if ($page == '')
	{
		$pageNum = 1;
	}
	else
	{
		$pageNum = $page;
	}
	$offset = ($pageNum - 1) * $this->rowsPerPage;
	return $offset;
}
```

I Kod 29 kan man se hur funktionen getOffset är uppbyggd. Här tilldelas först variabeln $pageNum värdet av $page. Ifall man befinner sig på första sidan för sökresultaten kommer page inte att ha fått något värde. Därför bestäms i koden att ifall värdet saknas får sidnumret värdet 1. Det som är intressant här är uträkningen av variabeln $offset. Den bestäms på basis av den aktuella sidans nummer minus ett multiplicerat med antalet resultat som angivits i $rowsPerPage. Uträkningarna ser alltså ut enligt följande:

Sida	1:	1-1*20	=	0
Sida	2:			2-1*20=20
Sida	3:			3-1*20=40

sida 4: 4-1*20=60

Det här innebär alltså att på första sidan skippas inga resultat, på andra sidan skippas de 20 första, på tredje sidan skippas de 40 första osv. Att någonting ska skippas bestäms dock inte i denna funktion, här räknas enbart värdena för respektive hopp ut.

När information hämtas från databasen sker detta i Tableklassen. Pagerklassen hämtar samma data från databasen som Tableklassen, men skriver inte ut datan på skärmen utan räknar istället antalet rader. Sedan kalkyleras hur många sidor sidbytaren skall innehåla på basis av hur många resultat databasfrågan returnerar. I Kod 30 är $numrows antalet rader som databasfrågan returnerat och $rowsPerPage, som vi tidigare specificerat i objektets actionsfil, anger antalet resultat som skall visas per sida. Den inbyggda PHP-funktionen ceil() avrundar resultatet uppåt till närmaste heltal ifall summan av uträkningen är ett decimaltal. Detta betyder att ifall vi har fem resultat och vi bestämt att 2 resultat visas per sida kommer sidbytaren att visa tre sidor. Uträkningen för detta blir 5 dividerat med 2 som blir 2,5, varefter ceil(2,5) returnerar 3. I praktiken betyder detta att den sista sidan kommer att innehålla endast ett resultat.

Kod 30. Uträkning av hur många sidor sidbytaren ska innehålla

```
$maxPage = ceil($numrows/$rowsPerPage);
```

Kod 31. Inledning av funktionen getPager

```
function getPager($self, $id, $curId, $table, $type)
```

Funktionen getPager() i Pagerklassen bygger upp själva sidbytaren. I Kod 31 visas inte den funktionalitet som byggs upp utan enbart vilka parametrar funktionen använder sig av. Variabeln $self innehåller alltid sökvägen till den sida varifrån man anropat funktionen. Sökvägen används för att bygga upp sidbytarens knappar. $id, $curId och $type används enbart vid listning av en anställds inlägg, alltså arbetstid, använt material, maskintid och dagtraktamentesinformation. Variabeln $id anger den kolumn som skall genomsökas medan $curId anger det värde som skall sökas efter i kolumnen. $type innehåller den typ av inlägg som man söker efter och används enbart för att fältet där man väljer typ av information i inte skall återgå till standardläge. Om man sökt efter exempelvis materialinlägg är detta förvalt även efter att sidan laddats om. Sådan funktionalitet existerar enbart för att förbättra programmets användarvänlighet. Variabeln $table anger tabellen i databasen från vilken informationen skall hämtas. Med hjälp av dessa parametrar kan Pager räkna ut rätt antal sidor då programmet listar inlägg.

I Kod 32 anropas funktionerna getRowsTable() och getPager(). Dessa funktioner körs som standard då man inte angett några sökkriterier. Som exempel använder vi sidan för översikt av projekt. Då getPager() anropas används enbart variablerna $self och $table eftersom de övriga inte behövs för denna sökning. Det enda som behövs för att sidan skall fungera korrekt är att programmet vet varifrån funktionen anropas och vilken tabell som skall genomsökas. I Kod 32 anges sökvägen ?a=projekt som $self och tabellen v_projekt som $table.

Kod 32. Anrop av getRowsTable() och getPager

```
$project->getRowsTable($pager->getOffset($_GET['page']), $pager-
>getRowsPerPage());
$pager->getPager('?a=project', '', '', 'v_projekt', '');
```

Själva sidbytargränssnittet som visas i Figur 12 byggs upp av koden i Kod 33. Variablerna $first, $prev, $next och $last är dynamiska länkar som byggs upp i getPager()-funktionen. Dessa utgör alltså länkarna till första sidan, föregående sida, nästa sida och sista sidan.

$pageNum och $maxPage visar på vilken sida man befinner sig och hur många sidor som sidbytaren innehåller.

Kod 33. Programmets sidbytargränssnitt

```
<div class="pager">' . $first . $prev . ' <span class="pages">' .
$translate->__('Sida') . ' ' . $pageNum . ' / ' . $maxPage . ' </span> '
. $next . $last . '</div>
```

14 Javascriptfiler

Eftersom vissa funktioner i programmet måste utföras i realtid utan att sidan laddas om använder programmet sig av javascript för vissa funktioner. Eftersom vi inte vill uppfinna hjulet på nytt har vi i vissa fall använt oss av färdigt utvecklade javascriptfunktioner som vi sedan modifierat för att passa in i vårt specifika program. I detta kapitel kommer vi att beskriva dessa javascriptfiler men inte i detalj gå in på kodnivå i de filer som vi inte kodat själv, eftersom koden i de flesta fall är väldigt lång. Ifall man är intresserad av koden i sin helhet finns alla kodfiler i en bilaga.

14.1.1 jQuery

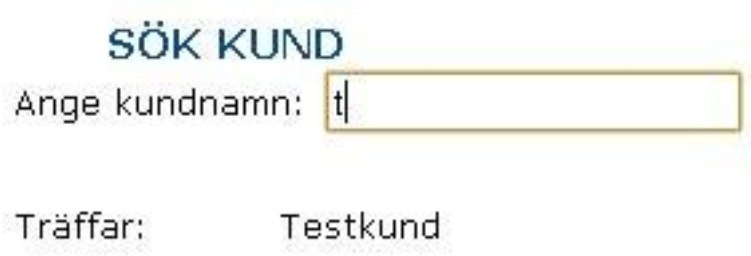
jQuery är ett javascriptbibliotek som används för att göra webbutveckling snabbare. I detta bibliotek finns färdiga funktioner som man kan använda på sin sida. I vårt projekt har vi använt funktioner från jQuery till att till exempel sortera tabeller och till kalenderfunktionen.

14.1.2 Autocomplete

Autocomplete var en javascriptdriven funktion som vi utvecklade för att söka bland sparad information i databasen då man valde kund, projekt, material eller maskin för inmatning. Vi ersatte dock denna funktion av en PHP-driven sökfunktion eftersom den var otillräcklig för det ändamål den var avsedd. Den största nackdelen var att det blev klumpigt då man var tvungen att ha flera Autocomplete-fält på en och samma sida.

Tanken var att att programmet skall kunna ge förslag i realtid då man söker något så vi skapade en funktion som vi kallade autocomplete. Autocomplete använder sig av skilda PHP-filer beroende på vilken databastabell man söker information från. Filerna var namngivna rpc_customer.php och rpc_project.php. Dessa var inte javascriptfiler i sig men

vi har valt att inkludera autocomplete under detta kapitel eftersom det använder sig av jQuery. Koden och funktionaliteten för detta beskrivs inte i detalj eftersom funktionen inte längre används av programmet. I Figur 13 kan man se hur programmet filtrerar sökresultat på basis av vad man skrivit in.



Figur 13. En funktion filtrerar sökresultaten och fyller i dem under ”Träffar” vid varje knapptryckning.

14.1.3 Tigra Calendar

Tigra Calendar är en gratis javascripkalender som man kan använda i samband med html-formulär. I vårt program har denna använts i alla situationer då ett datum skall matas in. Detta har varit lösningen att föredra eftersom man då inte behöver fundera på i vilket format användarna skriver in datumen. Tigra Calendar är beroende av de funktioner som finns i jQuery-biblioteket och jQuery följer med i paketet man laddar ner då man hämtar Tigra Calendar. I Figur 14 kan man se hur kalendern ser ut då man matar in datum i ett formulär.

Figur 14. Kalendern då man matar in datum i ett html-formulär.

15 Sökning och inmatning av projektresurser

Då en anställd matar in arbetstimmar, maskintid och material i ett projekt behövs en sökfunktion för att hitta rätt projekt, material och maskin. Vi hade ursprungligen tänkt använda oss av en utfällbar lista (drop-down) men eftersom det i framtiden kommer att finnas en stor mängd projekt och material i systemet blir det med den lösningen väldigt svårt att hitta rätt projekt eller material. Vi utvecklade därför en javascriptdriven funktion som filtrerade sökresultat för varje bokstav man skrev in i ett sökfält och som därefter fyllde i det valda resultatet man valt i rätt fält. Denna funktion skulle ha varit tillräcklig ifall man endast behövt ett sökfält per sida. På vissa sidor behövs det dock två sökfält eftersom man exempelvis skall kunna välja både projekt och material då man matar in materialåtgång i ett projekt. Vi löste detta genom att utveckla en helt ny PHP-driven sökfunktion som öppnas i ett nytt fönster. Vi kommer i detta kapitel gå igenom hur funktionen är uppbyggd.

När man lägger till material som använts i ett projekt ser formuläret ut som Figur 15. Märk Lägg till-knapparna vid projekt- och materialfältet. Dessa används för att öppna sökrutan. Själva fältet är låst för att förhindra inmatning av felaktig information.

Figur 15. Formulär för inmatning av material.

Då man klickar på Lägg till vid Projekt-rutan öppnas fönstret i Figur 16. Här kan man söka efter projekt som finns inmatade i programmet. Sökresultaten visas i en lista under sökrutan. För att välja projekt klickar man i alternativknappen före projektnamnet och väljer Lägg till. Projektet läggs till automatiskt i projektfältet och sökfönstret stängs.

Figur 16. Sökruta och listning av project.

Eftersom sökfunktionen öppnas i ett nytt fönster med hjälp av javascript så kan vi inte använda oss av existerande klasser. Funktionaliteten byggs upp i en självständig php-fil vid namn project_search.php. I vanliga fall inkluderas filer i programmet, men eftersom den här filen öppnas i ett nytt fönster är den i princip en självständig modul som förser programmet med tilläggsfunktionalitet. Detta betyder att en ny kontakt till databasen måste skapas i början av filen för att kunna få ut sökresultat. I projektklassen ser databasfrågan ut som i Kod 34. När man använder LIKE och %-tecken runt sökordet betyder det att det inmatade sökordet inte måste stämma exakt överens med projektets namn utan bara behöver innehålla den teckenkombination som matats in. Projekten har också status som anger om de är aktiva eller inaktiva. Denna sökfunktion söker enbart bland aktiva projekt med villkoret aktiv=’ja’, eftersom man inte ska kunna mata in någonting då ett projekt är inaktiverat.

Kod 34. Databasfråga för sökning

```
$query = "SELECT * FROM projekt WHERE projektnamn LIKE '%$keyword%' AND
aktiv = 'ja' LIMIT 10";
```

För att formuläret sedan ska kunna skicka det valda projektet till projektfältet i programmet används en javascriptfunktion vid namn post_value. Kod 35 visar funktionen i sin helhet.

Kod 35.Funktionen post_value

```
function post_value()
{
	var pid = '';
	var name = '';
	for( i = 0; i < document.search.pid.length; i++ )
```

```
        {
              if(document.search.pid[i].checked == true)
        {
        pid = document.search.pid[i].value;
        name = document.search.namn[i].value;
        opener.document.hours.projekt.value = name;
        opener.document.hours.pid.value = pid;
        self.close();
}
```

Då man har markerat det projekt man vill använda och tryckt på ”lägg till” körs funktionen i Kod 35. Den kör med en for-sats igenom listan på projekt och kollar om något projekt har valts. Från det projekt som valts hämtas projektets ID (pid) och projektets namn. För att fylla i namnet i rätt fält i det ursprungliga formuläret används opener.document.hours.projekt.value = name;. Opener.document syftar till filen som öppnade det nya fönstret, hours är formuläret där Lägg till-knapparna fanns och project är fältet där namnet skall matas in. Value syftar till det värde som läggs till i fältet. Projektets namn används dock inte av programmet utan är enbart till för att användaren ska se att han valt rätt projekt. Det som används för att koppla ihop projektet med den inmatade informationen är pid, som skickas till ett gömt fält i formuläret.

En likadan funktion används även till de övriga Lägg till-rutorna. Det som ersätts är endast vilka värden som hämtas ut ur databasen beroende på vilken tabell vi söker informationen ifrån. Då det handlar om material hämtas materialets namn, id och enhet och då det är frågan om en maskin hämtas maskinens namn och ID.

16 Översättning av gränssnitt

Enligt den ursprungliga projektspecifikationen skulle programmet enbart vara svenskspråkigt och de knapparna som var synliga för vanliga användare skulle vara tvåspråkiga. Kunden hade tänkt sig att alla länkar och knappar skulle skrivas ut på två språk så att användaren såg beskrivningen på båda språken samtidigt. Då vi byggde upp gränssnittet insåg vi dock att det var en omöjlighet att lösa språkdilemmat på det sättet eftersom menylänkarna och de olika knapparna tog allt för mycket utrymme då texten var utskriven på båda språken. Vi utvecklade därför en relativt enkel översättningsfunktion som ersätter texten i kodfilerna med text från en översättningsfil. Utgångsspråket är svenska och den ersättande texten finns i filen fi.txt i mappen translation. Själva funktionaliteten för översättningen är uppbyggd i klassfilen Translator.php. I början av

varje fil som innehåller text initieras sedan Translator med sessionens språk. Initieringen görs med strängen i Kod 36.

Kod 36. Skapande av instans för Translator

```
$translate = new Translator($_SESSION['language']);
```

Då man vill översätta något i själva kodfilen använder man sig av koden $translate->__(Exempeltext'). Den text som skall översättas måste sedan läggas till i fi.txt-filen för att automatiskt ersättas då användaren gör ett språkbyte. I textfilen skriver man en översättningssträng per rad, alltså ser raden för detta exempel ut enligt följande: Exempeltext=Esimerkkiteksti.

Då man loggar in väljer man språk och språkets sessionsvariabel tilldelas sitt värde. Då en sida laddas initieras Translator enligt Kod 36. Vid initieringsskedet får Translator veta vilket språk sidan skall visas på tack vare sessionsvariabeln language. Translator är en klass innehållande funktionerna som sköter om översättningen. Den viktigaste funktionen i Translatorklassen är function __($str). Kod 37visas hur den är uppbyggd.

Kod 37. Funktionen __

```
function __($str)
{
	if (!array_key_exists($this->language, $this->lang))
	{
		if (file_exists('./translation/' . $this->language . '.txt'))
		{
			$strings = array_map ( array ( $this, 'splitStrings'),
			file('./translation/' . $this->language.'.txt'));
			foreach ($strings as $k => $v)
			{
				$this->lang[$this->language][$v[0]] = $v[1];
			}
			return $this->findString($str);
		}
		else
		{
			return $str;
		}
	}
	else
	{
		return $this->findString($str);
	}
}
```

Då vi förklarar hur översättningen fungerar kommer vi att använda oss av ordet sträng. En sträng är ett ord eller en mening i programmet som översättningsfunktionen har tillgång till och kan översätta. Array_key_exists() i Kod 37 granskar vilket språk som är valt. Eftersom standardspråket som programmet är utvecklat i är svenska skall ingenting översättas ifall man valt svenska som språk. I vårt program finns för närvarande endast svenska och finska som alternativ. Vi har utvecklat översättningsfunktionen så att man i efterhand kan lägga till språk enligt behov. Då det finska språket är valt granskar funktionen att textfilen fi.txt finns. Som tidigare togs upp är fi.txt filen där översättningarna finns. Om en översättning inte hittas visas strängen på standardspråket, i vårt fall svenska. Vid $strings i Kod 30 ovan skapas en array av varje rad i textfilen. Arrayen som skapats innehåller en ny array för varje rad i textfilen med den svenska och den finska strängen skilt för sig. Detta betyder att $strings blir en array som innehåller värdet för varje rad i textfilen. Varje post i denna array innehåller en ny array med två värden, alltså den svenska strängen och den finska. Sedan granskas varje array i $strings skilt för sig med en foreach-loop. Foreach loopen söker igenom $strings och skapar en ny array som heter lang[’fi’]. För varje rad i $strings skapas en ny rad till arrayen lang[’fi’] som innehåller den svenska strängen som Id och den finska strängen som värde. Resultatet av detta är en tabell som ser ut enligt följande:

$this->lang[’fi’]
Array{
[’Välkommen’] => [’Tervetuloa’]
[’Program’] => [’Ohjelma’]
[’Tabell’] => [’Taulukko’]
}

Efter att foreachloopen gått igenom alla poster i strings-arrayen anropas funktionen findString med vår översatta textsträng som parameter. Funktionen findString som presenteras i Kod 38 granskar om en motsvarighet till den sträng som skall översättas hittas bland arrayen lang[’fi’]:s Id-värden. Ifall en motsvarighet hittas ersätts den svenska strängen med den finska strängen som sparats i arrayen lang[’fi’]:s värdefält.

Kod 38. Funktionen findString

```
private function findString($str)
{
	if (array_key_exists($str, $this->lang[$this->language]))
	{
		return $this->lang[$this->language][$str];
	}
```

```
        return $str;
}
```

Då Kod 38 exekverats betyder detta att den svenska strängen har ersatts med den finska. Hela översättningsfunktionen körs såklart igenom för alla de svenska strängarna som finns på den sida man öppnar då man har valt finska som språk. Slutresultatet är att användaren enbart ser en finsk version av programmet, oberoende av vilken sida han besöker. Det som inte i detta program översätts är den inmatade informationen i databasen, alltså de anställdas titlar, maskinernas och materialens namn osv. I detta fall ville kunden att dessa skulle vara enbart på ett språk eftersom de har sina egna beteckningar på materialen och maskinerna och alla anställda känner igen dessa oberoende av vilket språk de pratar.

17 Versionshantering och säkerhetskopiering

Då man arbetar med utveckling av programvara är säkerhetskopiering och versionshanteringen väldigt viktig. Alltid då man utvecklar nya funktioner bör man vara beredd på att man kan vara tvungen att återgå till en tidigare version ifall något inte går enligt planerna.

Vi säkerhetskopierade vår programvara och databas vid varje dags slut. För att lättare kunna följa med vilka ändringar som gjorts förde vi också logg över alla gjorda förändringar. Säkerhetskopiorna sparades alla i sin helhet tills projektet var slutfört. Säkerhetskopiorna arrangerades såklart enligt datum, men programmet har även ett versionsnummer. Versionen står i jämförelse till hur långt framskridet projektet är och vilka delmål vi nått. Om man vill återgå till en tidigare version kan man med hjälp av loggen välja rätt datum att återgå till.

De säkerhetskopierade filerna sparade vi såväl på hårddisk som på ett för ändamålet avsett USB-minne. Tjänsteleverantören kör även säkerhetskopiering varje dag på servern vi arbetade mot. I slutet av varje dag säkerhetskopierade vi filerna till en gemensam katalog i ett program som kallas Dropbox. Genom att använda Dropbox försäkrade vi oss om att alla filer fanns tillgängliga på alla datorer vi arbetade på. Detta är både en bekvämlighet och en säkerhetsåtgärd eftersom filerna fanns sparade på flera olika ställen och dessutom var tillgängliga även ifall vi inte hade tillgång till internet då alla datorer kopplade till Dropbox-katalogen laddade ner filerna lokalt.

Versionerna går enligt följande:

0.1: Filstruktur och databasstruktur klar. Planering av klasser gjord.

0.2: Inmatningsfunktionalitet för projekt klar

0.3: Inmatningsfunktionalitet för kunder klar

0.4: Inmatningsfunktionalitet för maskiner och material klar

0.5: Profilsida för anställd, projekt och timmar för lönerperiod klar

0.6: Alla de viktigaste inmatningsfunktionerna är programmerade. Finslipning och komplettering gjord.

0.7: Sökfunktion för kund, material, maskin och projekt färdigställd. Sökningen implementerad även då man matar in information i någon av tabellerna.

0.8: De beställda funktionerna är färdigställda och redo för testning

0.9: Alla funktioner är testade och felkorrigerade, gränssnittet är färdigt.

1.0: Systemet är klart för leverans och testning hos kunden.

1.1: Utveckling av tilläggsfunktionalitet och stödfunktioner för smidigare användning färdig.

1.2: Optimering och fullständig kommentering av kod färdig.

18 Implementering av affärssystemet Rasmet hos kunden

Till vårt uppdrag hör även installation av affärssystemet hos kunden samt skolning av personalen i dess användning. Installationen av själva affärssystemet kräver ingen större satsning eftersom systemet är helt webbaserat. Det finns dock några kriterier som de datorer som skall använda systemet måste uppfylla. Datorerna måste vara utrustade med antingen Internet Explorer 8 eller 9, en uppdaterad version av Google Chrome eller Mozilla Firefox. I dessa webbläsare måste även javascript och popup-fönster vara tillåtna.

Skolningen av personalen delades upp i två kategorier, skolning av administratörer och skolning av användare. Eftersom alla administratörer var svenskspråkiga gick denna

skolning enbart på svenska. I personalen finns såväl finsk- som svenskspråkiga så valde att ordna två separata skolningstillfällen, ett på finska och ett på svenska. Skolningen hölls i Yrkeshögskolan Novias utrymmen och förutsättningen var att alla deltagare hade tillgång till en egen dator så att alla deltagare enkelt kunde följa med. För detta ändamål valde vi att boka en datasal ifall alla i personalen inte hade tillgång till en egen bärbar dator.

18.1 Installation av trådlöst system hos Raseborgs Metall

För att möjliggöra att systemet ska kunna användas av de anställda för att mata in materialåtgång och maskintid under arbetsdagen ska en dator placeras i verkstaden. Eftersom det skulle vara svårt att dra kabel från switchen skall ett trådlöst system som täcker större delen av verkstaden läggas igång. I framtiden är det tänkt att flera datorer ska placeras runtom i verkstaden så att man enkelt kan mata in information oberoende av vid vilken station man arbetar. Till att börja med kommer det dock enbart att finnas en gemensam dator i verkstaden. All konfiguration och installation sköts av oss själva.

18.1.1 Beskrivning av nuvarande system

För närvarande finns på kontoret en switch till vilken alla datorena på kontoret är kopplade. I skrivande stund finns ingen dator i själva verkstaden. Från switchen går en nätverkskabel till varje dator. Eftersom kontoret är relativt litet har det tidigare inte funnits behov för ett trådlöst system. Det skulle vara möjligt att med detta system dra en nätverkskabel ner till en dator i verkstaden, men tanken är att vi ska undvika kabeldragning med tanke på bekvämlighet och möjlighet till utvidgning av systemet. Datorerna på kontoret används enbart av den personal som jobbar där och kommer i första hand att utnyttjas av systemets administratörer.

18.1.2 Trådlös lösning med Netgear router och bärbar dator

För att det trådlösa nätverket skall täcka så mycket som möjligt av verkstaden använder vi oss av en WNDR3400, en av Netgears effektivare routrar. Tanken är att den bärbara datorn skall kunna placeras nästan var som helst i verkstaden. Routern kopplas till den befintliga switchen och placeras så nära verkstaden som möjligt. Den bärbara datorn kan sedan flyttas runt enligt behov. Till routern kan även kopplas ett större antal datorer ifall behovet av stationer i verkstaden ökar. Ifall man vill utöka området som det trådlösa nätverket täcker finns det även enkla sätt att göra detta. Eftersom vi använder en Netgear router är

det rekommenderade sättet att göra detta att använda en Netgear Range Extender som placeras innanför det trådlösa nätverkes täckningsområde och som tar emot den trådlösa signalen och förstärker den. Eftersom man i framtiden inte vill vara tvungen att lita till endast en tillverkare är detta system även kompatibelt med en teknik som kallas Universal Repeating. Detta är en funktion som finns inbyggt i vissa routermodeller, oberoende av tillverkare. Funktionaliteten är densamma som vi Range Extending, men ifall man köper en router som man dedikerar till Universal Repeating betyder det att man stänger ner merparten av dess funktioner och betalar för sånt man inte använder.

Datorn kommer att vara av modell Lenovo TP L520. Denna modell har vi valt på grund av dess tålighet. Den klarar av att fungera i såväl höga som låga temperaturer, den tål damm och fukt och är även vibrations- och stötsäker. Lenovo har även en mycket välutvecklad felkontroll och anses av många vara en av de främsta tillverkarna av tillförlitliga datorer för företagsbruk. Ifall kunden konstaterar att någon del av datorn inte klarar de förhållanden som råder i verkstaden finns här även möjlighet att koppla till en extern skärm och ett externt tangentbord. Genom att exempelvis använda ett externt tangentbord kan man undvika att få smuts och olja på själva datorn. Datorns pris är 799 euro och ett tangentbord kostar omkring 15 euro.

19 Testning

Under utvecklingsskedet utsatte vi konstant programmet och dess kod för testning. Vi träffades med kunden under projektets gång för att granska de funktioner som redan utvecklats och för att klargöra vilka funktioner som saknas. Vi klargjorde redan under planeringsskedet våra delmål för projektet och har följt upp dessa med versionshanteringen beskriven i kapitel 17.

Vi reserverade en implementerings- och testningsfas på två veckor för systemet. Under denna tid använde kunden systemets olika funktioner och granskade hur dessa håller ihop under dagligt bruk. Kunden hade även en utomstående konsult vars uppgift var att kritiskt genomsöka affärssystemet för möjliga brister. De brister som hittades under testningsfasen kunde lätt korrigeras.

20 Affärssystemets uppdatering

I framtiden kommer kunden möjligtvis att vilja uppdatera systemet att innehålla fler funktioner, eller kanske ändra på någon del av funktionaliteten. För att undvika långa serviceavbrott är det möjligt att arbeta på en kopia av systemet medan kunden har det i användning. En eventuell uppdatering kan sedan göras efter arbetstid utan att orsaka något avbrott i användningen. Man kan då helt enkelt ersätta de gamla filerna med de nya, exempelvis via en FTP-klient, och på detta sätt lägga till funktionalitet utan att röra den information som sparats i databasen.

21 Affärssystemets fortsättning

Vi har valt att inte ge upp upphovsrätten till vårt system eftersom det finns en potentiell marknad för just ett sådant affärssystem som vi utvecklat. Eftersom vi redan äger ett företag som säljer IT-tjänster har vi tänkt inkludera en modifierad version av RasMet, som vi sedan konfigurerar för varje kund, i vårt utbud. Då vi fram tills nu har haft så många kunder vi kan klara av och det arbete vi gjort till kunderna alltid varit skräddarsytt för just dem har varje kund upptagit ganska mycket av vår tid. Nu när grunden för systemet är kodad och det mesta kan återanvändas för varje kund kan detta resultera i större ekonomisk vinning jämfört med antalet arbetstimmar för företaget ifall vi lyckas marknadsföra programmet effektivt.

Ifall vi väljer att börja marknadsföra systemet kommer vi till en början att göra detta i relativt liten skala. Vår hemsida kommer nu liksom tidigare att vara där vi är mest synliga. För att locka besökare och kunder i en mängd vi kan hantera skulle vi använda oss av en Google Adwords-kampanj. Med en sådan kan man lätt locka in en större mängd besökare till hemsidan per dag. Det goda med Adwords är att reklamen som kommer upp i samband med att man gör googlesökningar enbart syns för dem som har angivit relevanta sökord. Om vi så skulle vilja skulle reklamen endast synas när någon skriver in ordet ”affärssystem”.

Företag som säljer affärssystem gör ofta stora pengar på underhåll. Vi har planerat att sälja underhållspaket för att kunden skall få tillgång till uppdateringar och support. Innan vi kan bestämma någon prissättning måste en grundligare undersökning om vad som redan finns på marknaden göras. Allt detta är dock enbart i planeringsskedet ännu och inga riktiga

beslut om vad vi kommer att göra med programmet har gjorts. Vad som händer efter att vi examinerats beror på hur mycket tid vi kan lägga på vårt företag.

22 Slutdiskussion

Att arbeta med detta projekt har varit såväl roligt som givande. Under projektets gång har vi sysslat mest med PHP-programmering men även stött på en del javascript. Vi har utvecklat egna lösningar och även lärt oss att tillämpa färdigskriven kod för att undvika att ”återuppfinna hjulet”. Då vi började med detta projekt för vad som känns som en halv livstid sedan funderade vi mången gång om vi någonsin kommer att bli färdiga med programmet. Tanken att vi tagit oss vatten över huvudet har dykt upp ett antal gånger under projektets gång. Efter sommaren såg vi dock ljuset i slutet av tunneln och fördubblade därefter våra insatser för att kunna leverera ett så kvalitativt och användarvänligt system som möjligt. Efter närmare 800 sammanlagda arbetstimmar kan vi lugnt säga att de känns bra att ha programmet klart för leverans.

När vi mottog detta projekt under våren 2011 var våra PHP-kunskaper inte på nära håll tillräckliga. Vi har under vår studietid enbart gått en kurs som handlat om PHP-programmering. Vi hade grundläggande kunskaper i ämnet som vi erhållit via vårt projektarbete med Drupal. I Drupal handlar det dock mest om att använda färdiga funktioner och att skriva ut innehåll på webbsidor och inte att verkligen utveckla ny funktionalitet, så ett programmeringsuppdrag av denhär omfattningen var något helt nytt för oss. Vi hade dock enorm hjälp av vår klasskamrat, vän och affärspartner Rasmus Werling som hjälpte oss komma igång under våren 2011, och som från tidigare var insatt i PHP-programmering.

Som i alla programmeringsuppdrag har det uppstått situationer då det känts som om man inte kunnat lösa ett problem efter att ha kämpat med det i många timmar. I sådana situationer har vi använt oss av en teknik som vi kallar ”att bollplanka”. Det är möjligt att det finns något officiellt namn för det och vi påstår oss inte ha uppfunnit metoden. Då någon av oss känt att problemet inte går att lösa med den kunskap han har nu brukar vi ta en paus för att gå igenom problemet tillsammans. Processen är simpel, den som stött på ett problem berättar om problemets natur, vad han vet och hur han ursprungligen hade tänkt lösa det. Ifall det handlar om något mer komplicerat har vi använt oss en tavla (whiteboard) för att få en bättre översikt. Ofta resulterar detta i att personen som förklarar problemet

själv hittar lösningen under den tid han förklarar. Ett missat steg eller ett tankefel är ofta orsaken till varför man inte kommit vidare och när man ensam sitter och går igenom koden är det svårt att hitta felet. I programmering, lika som med all slags skriveri, är man ofta blind för sina egna fel. Ifall man inte kommer på lösningen är det sedan lätt för den som lyssnar att ställa frågor och komma med förslag. Ofta kan en annan person se problemet från en annan synvinkel och också se ett annat sätt att tackla det.

Att arbeta i par har för oss varit något helt naturligt. Utbildningen vi gått är till stor del projektbaserad och vi har under tre års tid aktivt jobbat tillsammans nästan varje dag. Under sommaren träffades vi en gång i veckan, eftersom vi båda jobbade heltid på varsin arbetsplats. Från och med början av september har vi dock arbetat tillsammans fem dagar i veckan. Eftersom vi till största del har arbetat samtidigt på samma plats har de oss utvecklare emellan inte uppstått några kommunikationsproblem. Vi tror på att arbetet löper smidigare om man hela tiden har tillgång till varandra för att utbyta idéer och kunna hjälpa ifall den andra stöter på problem. När man jobbar tillsammans med någon blir risken för att bli distraherad också mindre. Vi har med denna arbetsmetod kunnat arbeta effektivt den största delen av dagens timmar. Ofta har kaffepauserna uteblivit för att man varit så inne i det man håller på med.

Då vi mottog projektet diskuterade vi enbart kundens önskemål och vad de ville uppnå med sitt affärssystem. Vi träffades sedan hos kunden där vi fick en rundur av kontoret och verkstaden och kunden berättade hur systemet skulle implementeras i praktiken. Här fick vi en mycket bättre bild av hur vi skulle förverkliga programmet för att verkligen passa kundens unika behov. Vi har även hållit regelbundna möten med tre veckors mellanrum och däremellan hållit kontakt via e-post. Genom att effektivt hålla kontakt med kunden har vi kunnat undvika missförstånd och onödigt arbete. Under projektets gång har vi endast en gång missförstått kundens önskemål. Trots missförståndet blev den onödiga arbetsmängden dock ganska minimal eftersom det handlade om en liten ändring för att ändra funktionaliteten så att den motsvarade kundens önskemål. I vårt fall handlade missförståndet om hur programmet hanterar inlagda maskintimmar. Vi hade trott att maskintimmar enbart skulle hanteras som en projektresurs och inte något som även sparades i varje anställds arbetstimmar. Vi löste detta dock enkelt genom att inkludera maskintiden för varje användare i hans personliga timuppföljning.

Det enda som inte har fallit ut som vi menat är vår tidsplanering. Vi förväntade oss vara klara med programmerandet vid mitten av oktober. Eftersom önskemålen och kraven från kunden sida ökat under projektets gång har även tiden det tagit att utveckla blivit längre. Detta är till stor del på grund av att vi inte i planeringsskedet gått igenom allt tillräckligt detaljerat med kunden. Ifall vi i framtiden gör liknande projekt vet vi att kartläggningen av kundens behov måste göras mycket grundligare än vad vi gjorde i det här projektet. Vår oförmåga att göra en bättre kartläggning berodde på att vi inte ställde tillräckligt med frågor, samt att vi inte visste vilka frågor vi borde ställa. Vi är från tidigare vana att göra kartläggningar för kunder som vill ha en hemsida och är inte från tidigare bekanta med utvecklingen av affärssystem. Många saker som vi utgick ifrån att var självklara och inte krävde någon större programmeringsinsats var rätt så svåra att få att fungera. Saker som säkerhet, validering av formulär, översättning och sökningar från databasen då man fyller i formulär är något som tidigare har skötts av innehållshanteringssystemet som vi byggt upp hemsidorna i. Att från grunden utveckla dessa funktioner var mycket mera komplicerat än vi tänkt oss.

23 Avslutning

Resultatet av vårt arbete är ett välfungerande och för kunden användbart system som motsvarar och på många punkter överträffar det system kunden beställde. I skrivande stund har systemet genomgått testning hos kunden i två veckor och skall tas i bruk i den dagliga verksamheten om tre dagar. Återkopplingen har varit väldigt positiv och vi ser fram emot att se systemet i användning. Med kunden har vi kommit överens om att vi kan utveckla systemets funktionalitet vid behov under det kommande året. Vi vill tacka vår kund, Raseborgs Metall, för möjligheten att jobba på detta projekt samt för helhjärtat deltagande under hela utvecklingsprocessen. Ett så bra slutresultat som vi har uppnått med detta projekt kan endast nås genom gott samarbete beställare och utvecklare emellan. Vi väntar med spänning på att se vilken affärsnytta ett sådant här system kan bidra med till vår kunds verksamhet och ser fram emot gott samarbete även i framtiden.

Källförteckning

Englund, T & Olsson, L-J. (2003). *Objektorienterad programmering*. Undervisningsmaterial i objektorienterad programmering för Språkteknologiprogrammet på Uppsala Universitet.

Flanagan, D. (2011). *JavaScript: The Definitive Guide: Activate Your Web Pages*. California: O' Rielly Media Inc.

Johnsson, V. (2001). *Webbprogrammering med PHP*. Lund: Studentlitteratur Ab.

Mathiassen, L & Munk-Madsen, A & Nielsen, P A & Stage, J. (2001). *Objektorienterad analys och design*. Lund: Studentlitteratur.

PHP.net. (2001-2011). http://php.net/manual/en/function.mysql-real-escape-string.php

Rantala, A. (2005). *Web-ohjelmointi*. Borgå: Docendo Finland Oy.

Vaswani, V. (2004). MySQL*: The Complete Reference*. California: McGraw-Hill/Osborne.

W3Schools.com. (1999-2011). http://www.w3schools.com/php/func_string_md5.asp

Figurförteckning

Kodförteckning