Förenklat webbaserat bokföringssystem

med Medlemsregister

Rasmus Peltonen

Benjamin Ekholm

Examensarbete för Tradenom-examen

Utbildningsprogrammet för Informationsbehandling

Raseborg 2013

Examensarbete

Författare: Rasmus Peltonen och Benjamin Ekholm

Utbildningsprogram och ort: Tradenom, Raseborg

Inriktningsalernativ/fördjupning: Informationsbehandling

Handledare: Klaus Hansen

Titel: Förenklat webbaserat bokföringssystem

Datum: 1.11.2013 Sidantal: 56 Bilagor: 0

Sammanfattning

Vårt examensarbete är ett utvecklingsprojekt av ett bokföringssystem för Ekenäs Kåravdelning R.f. Slutprodukten är ett bokföringssystem samt ett medlemsregister som är webbaserat.

Bokföringssystemet lagrar transaktioner som skett under årets gång. Man skall kunna lägga till och redigera olika konton. På basis av informationen som matats in under ett år så görs det automatiskt upp ett bokslut.

Systemet har även ett medlemsregister över alla medlemmar så det är lätt att följa med hur många medlemmar föreningen har. Systemet lägger automatiskt in medlemsavgifter i bokslutet. Vi hade som mål att få systemet så automatiserat som möjligt.

Språk: svenska Nyckelord: bokföringssystem, automatisera

Bachelor's Thesis

Author: Rasmus Peltonen and Benjamin Ekholm

Degree Programme: Business Information Technology, Raseborg

Specialization: Information processing

Supervisors: Klaus Hansen

Title: A simplified web-based accounting system

Date: 1.11.2013 Number of pages: 56 Appendices: 0

Summary

Our bachelor's thesis is a development project on a bookkeeping system for the students' association Ekenäs Kåravdelning r.f. The final product is a web-based bookkeeping system including a register of members.

The system stores transactions that have occurred during the year. You can add and edit different accounts. A balance sheet is automatically calculated based on the information the users have put in.

The register of members makes it easier to keep track of the current number of members. The system will automatically insert the membership fees into the balance sheet at the end of the year. Our goal was to make the system as automatic as possible.

Language: Swedish Keywords: bookkeeping system, automation

Innehållsförteckning

1. Inledning

Detta system skall hjälpa Ekenäs kåravdelning R.f. med ekonomiuppföljning samt upprätthållning av föreningens medlemsregister. Systemet har dock sin tyngd på bokföringen.

Ekonomiuppföljningen är viktig för denna förening då den ordnar ett antal olika evenemang per år. Medlemsuppföljningen är också viktig då föreningen kan ha upp till 100 medlemmar under ett år.

Slutprodukten av arbetet är ett webbaserat bokföringssystem som enkelt skall visa vad pengarna i föreningen går till, samt automatiskt göra ett bokslut på alla transaktioner när året är slut. Systemet har även ett medlemsregister för att underlätta uppföljningen av föreningens medlemmar. Bokföringen skall underlätta den ekonomiansvariges och styrelsens arbete på ekonomifronten. De flesta som väljs till ekonomiansvarig har ingen tidigare erfarenhet av att hantera större summor pengar.

Vi har båda inblick i hur föreningens ekonomi sköts. Rasmus Peltonen har under läsåret 2012-2013 fungerat som ekonomiansvarig och Benjamin Ekholm har fungerat som sektreterare under samma tid.

För att ge läsaren en bättre bild av föreningen och dess verksamhet beskriver vi i nästa kapitel vad Ekenäs Kåravdelning R.f är.

1.1 Ekenäs kåravdelning R.f.

Ekenäs kåravdelning är numera en av Yrkeshögskolan Novias studerandeföreningar. EKA, som tidigare hörde under studerandekåren Studväst, fungerar numera långt som en självständig studerandeförening med ansvar för den verksamhet som riktar sig till Novias studerande i Raseborg. EKA har en viktig uppgift i att hålla samman studerandena på orten eftersom det finns så få alternativa aktiviteter.

I och med att föreningen är rätt så ny har EKA inte ännu hunnit skapa långa traditioner, men deltagarantalet på våra evenemang ökar ständigt och EKA hoppas på en fin och aktiv verksamhet också i framtiden.

Till EKAs mål för det kommande åren är att göra föreningen mera känd och få studerande att känna sig som hemma på sin studieort. Det skall vara låg tröskel att ta del av evenemangen och föreningen skall finnas till för alla.

För EKA är det viktigt att vara en del av Svenskfinland och vara en del av helheten. För att inte bli isolerade på studieorten måste EKA kunna delta i andra studerandes verksamhet på andra orter. Årsfester och andra representationsuppdrag är viktiga evenemang att synas på för att föra fram EKA som aktör i studentvärlden.

För att Ekenäs också i framtiden skall locka nya studerande anser föreningen det vara viktigt att upprätthålla en god och kamratlig anda utbildningsprogrammen emellan. EKA vill vara delaktig i att skapa en hemtrevlig stämning. Ekenäs har en salig blandning av studerande i olika branscher vilket innebär att föreningen måste arrangera program av olika slag. Detta betyder att föreningen behöver marknadsföringsmaterial och rekvisita för att synas och höras bland våra studerande.

Ett av föreningens och högskolans centrala mål kommer att vara att befrämja välmående och hälsa. I dagens samhälle är det viktigt att tänka på en hållbar utveckling också inom den sociala sektorn och erbjuda motionsmöjligheter. EKA har ställt som mål att varje månad ha någon form av idrottsevenemang och temakvällar för att sammanföra studerande. Evenemangen är exempelvis att ordna innebandyturneringar, tjejkvällar, temakvällar samt andra liknande evenemang. Fester är inte hela föreningens verksamhet.

1.2 Läget just nu

För tillfället består bokföringen samt medlemsregistret i föreningen av en mängd papper och excel-tabeller. Varje år röstar föreningen in en ny person som skall ansvara för området under kommande läsår. Detta bidrar till att strukturen för bokföringen samt medlemsregistret ändras varje år. Varje person har sitt eget sätt att lägga upp själva bokföringen och senare bokslutet. Detta är ett stort problem när följande ekonomiansvariga vill se tillbaka på föregående år och inte förstår varifrån summorna kommer, att så kunna jämföra boksluten från år till år försvåras. De som väljs in som ekonomiansvariga har sällan tidigare bokföringskunskaper.

Medlemsregistret är sällan uppdaterat så att alla som har betalat verkligen ingår i det. Detta resulterar till att någon som har betalat medlemsavgiften kanske inte får de utlovade fördelarna.

På figuren nr 1 på nästa sida kan vi se ett exempel på hur bokföringen ställts upp under tidigare år. Detta är dock bara ett exempel av många och utseendet har ändrat flera gånger under åren.

		Evenemang					**Övrigt Utgifter**									
														Representation		
	Namn:	**Beerpong**					Kangasmerkit(Sitz?)				-160				-230	
	Utgift :	156					Kangasmerkit(EKAvsÅKA)				-120					
	Inkomst:	180														
	Totalt:	24					SJR HOST(hemsidan)				-93,04			**Halare**		
							Kangasmerkit(gulis + EKA medlem)				-374,23			9st (45e/st)	405	
							Öl från Tyskland				-400			24st (35e/st)	840	
	Namn:	**Casualsitz**					Elmo Sport (Boboll)				-112					
	Utgift :	210														
	Inkomst:	405												**Medlemmar (bör uppdateras)**		
	Totalt:	195												52 st (5e/st)	260	
														17 st (10e/st)	170	
	Namn:	**EKAvsÅKA**														
	Utgift :	165														
	Inkomst:	310														
	Totalt:	41														

Figur 1. Bokföringen just nu

1.3 Mål

Slutprodukten är enligt följande:

- Bokföringssystem
    - Kontoplan
    - Lagring av transaktioner
    - Evenemang
    - Bokslut
- Medlemsregister
    - Alumni- och ordinariemedlemmar

Alla dessa punkter tas kort och koncist upp i detta kapitel.

Slutprodukten av arbetet är ett bokföringssystem med ett medlems- samt alumniregister. Systemet skall lagra transaktioner som det automatiskt för över till ett bokslut som räknas ut i realtid.

Medlemsregistret skall lagra information om alla medlemmar. Detta skall hjälpa föreningen att bättre veta exakt vem som är betalande medlemmar i föreningen. Med hjälp av detta system så kommer alla medlemmar att få de privilegier som de är utlovade då de betalar medlemsavgiften till föreningen.

Alumniregistret är någonting helt nytt för föreningen. Det kräver mer information om medlemmarna eftersom dessa personer redan har blivit utexaminerade. Alumnimedlemmarna består av personer som vill stöda föreningen efter sin studietid.

Bokföringsdelen består av olika delsystem som kontoplan, bokföringshändelser samt bokslut.

I kontoplanen skall man kunna lägga till och redigera konton. Man skall kunna modifiera kontoplanen som man vill. Bokföringshändelser står för lagringen av transaktioner och även här bör man kunna lägga till och redigera de olika posterna.

Man skall kunna föra in evenemang i systemet och på så sätt även få fram vilka utgifter och inkomster ett evenemang orsakat.

Bokslutsdelen gör automatiskt upp ett bokslut på basis av den information som kommer från kontoplanen och bokföringshändelserna som användare har matat in.

Produkten är en webbapplikation. Man skall kunna komma åt informationen var man än befinner sig. Detta resulterar såklart i att sidan måste vara säkerhetsbelagd.

1.4 Prioriteringar

Prioriteringarna kan delas in i tre olika områden: användarvänlighet, säkerhet och funktionalitet. Dessa delar är alla likvärdiga. Användarvänlighet har dock lite högre prioritet än de övriga.

Användarvänligheten innebär att systemets användargränsnitt skall vara lättanvänt och lätt att förstå sig på. Personer som har varken IT-, eller ekonomibakgrund bör lätt kunna använda systemet utan större svårigheter. Detta försöker vi lösa genom att göra systemet så automatiserat som möjligt. Användarvänligheten innebär även att uppställningen av systemet skall vara lätt att förstå och personen som ansvarar för bokföringen snabbt skall inse vad som finns var.

Då man behandlar information om både bokföring och personuppgifter så är säkerheten av yttersta vikt. Det har från första början varit en självklarhet att systemet måste ligga bakom lås och bom. Ingen obehörig får komma åt varken systemet eller informationen.

Funktionaliteten består av två olika delar, automatisering för att hjälpa användaren och begränsning av möjligheterna att användaren gör fel. Dessa två delar kommer speciellt upp vid uppläggningen av bokslutet, då det har varit en hel del problem tidigare år. Automatiseringen räknar ut totala summor för varje konto för året och lägger upp ett bokslut på basis av den information som användaren lägger in.

1.5 Intressenter

Det finns många intressenter för detta system. De första och viktigaste i detta läge är självklart Ekenäs Kåravdelnings styrelse, huvudsakligen den ekonomiansvariga, men även övriga styrelsemedlemmar har nytta av den informationen som lagras i systemet.

Självklart så är också föreningens medlemmar intressenter. De förväntar sig att få fördelarna de utlovades som betalande medlemmar och det kommer förhoppningsvis alla att få med hjälp av detta system.

Föreningen kommer äntligen att kunna ta emot alumnimedlemmar eftersom föreningens stadgar ändrades hösten 2013. Så alla utexaminerade studerande från Novia Raseborg är intressenter. Med hjälp av detta system kommer administrationen av ovannämnda medlemmar att bli betydligt lättare.

Andra föreningar kan även vara intressenter om utvecklarna bestämmer sig för att systemet fungerar så bra att man kunde sälja det vidare. Detta koncept fungerar för de flesta små föreningar som är i behov av en liten bokföring samt ett medlemsregister.

Övriga intressenter kan vara studerande som inte ännu är medlemmar, skolan i sig, samt lokala småföretag, som även kan ha nytta av systemet.

2. Val av verktyg

Verktygen vi använde oss av vid utvecklingen av systemet var rätt självklara från första början. Vi har använt väldigt populära metoder och programvara med öppen källkod, för att undvika licenskostnader då vi inte har någon budget för arbetet.

För att bygga en god grund för arbetet använde vi HTML. Designen görs med hjälp av CSS.

När vi hade grunden så började det mer invecklade arbetet med skriptspråket PHP för att få funktionalitet på de olika delarna av systemet. För att utöka funktionaliteten och dynamiken av PHP använder vi oss även av MySQL. Datalagringen sker med hjälp av MySQL, som är ett databassystem.

För att få ännu mer stilrenhet och dynamik på olika ställen, har vi även valt att använda ett JavaScript bibliotek som bygger på JQuery. Detta heter JQueryUI.

Dessa verktyg behandlas närmare i enskilda underkapitel för att ge läsaren en bättre bild över hur det hela fungerar och mer specifikt, var och till vad verktygen används.

2.1 HTML och CSS

HTML står för HyperText Markup Language och är det huvudsakliga verktyget för att skapa grund åt webbsidor. HTML är kod som webbläsaren tolkar för att få struktur på webbsidor.

Detta kan till exempel jämföras med en nyhetsartikel som man vill placera på en webbsida. Artikeln måste då ha formatering av någon sort för att webbläsaren skall förstå vad som är rubrik, brödtext, bilder och så vidare. För att göra detta möjligt använder man sig av delar som heter: element, attribut och taggar.

För att ge olika element flera egenskaper och mera funktionalitet, kombinerat med andra språk, används attribut. Attribut definieras oftast inom den öppnande taggen av ett element.

Attribut används också för att ge olika namn eller klasser på element, namnen ges oftast i form av ”*id=’namn’*” och klass ges i ”*class=’klass’*” detta används oftast för att kunna ändra utseende på de olika elementen och få sidan att se ut just så som man vill. Detta görs med hjälp av CSS som står för Cascading Style Sheets

(Duckett 2011, s 1 – 10)

Med HTML kommer man en bit på vägen, men man är ganska begränsad vad gäller ändrande av design och just därför används CSS. Med hjälp av CSS kan man lätt ändra färger på olika element och definiera en mångfald olika utseenden för elementen.

CSS används genom att att ge regler till element som syns på webbsidan, dessa regler bestämmer hur webbläsaren återger de element man har gett regler åt.

(Duckett 2011, s 243 – 246)

Utan att ge oss in djupare på dessa verktyg kan vi konstatera att de ger respektive struktur samt utseende åt systemet.

2.2 JQueryUI

JQueryUI är ett bibliotek javascript-funktioner som hjälper webbutvecklare att få en stilren och enhetlig design till sina webbsidor och webbapplikationer. Detta verktyg bygger på ett existerande JavaScript-bibliotek som heter JQuery.

De viktigaste användningsområden i detta arbete för JQueryUI är att ge enhetligt utseende åt knappar, att dela in vissa sidor i flikar och användning av en datumväljare.

Speciellt datumväljaren, som visas i figuren nr 2 nedan, underlättar betydligt arbetet för användaren då ett datum skall matas in i ett formulär. Som figuren visar är den stilren och enkel. Det är betydligt lättare att bara klicka på ett datum istället för att skriva in ett och risken för att skriva fel minskas.

Figur 2. jQueryUI:s datumväljare

Eftersom vi inte ändrar eller konfigurerar dessa färdiga funktioner desto mer, så går vi inte nogrannare igenom dessa verktyg då de är relativt enkla att implementera i lösningarna och ger mycket funktionalitet och dynamik till webappen.

2.3 PHP

PHP står för ”PHP: Hypertext Preprocessor” och är ett skriptspråk som har öppen källkod. Detta verktyg är utmärkt för webbutveckling och kan lätt bäddas in i HTML dokument för att skapa mer dynamiska webbsidor.

Med PHP kan man i princip göra vad som helst som man gör med andra programmeringsspråk. PHP är dock huvudsakligen fokuserat på server-side scripting. Kombinerar du PHP med MySQL har du två utmärkta verktyg för att lagra och redigera data och för att skapa ett interaktivt system för att upprätthålla information.

PHP står oftast för programdelen av en dynamisk hemsida och gör oftast jobbet genom att lagra information i databasen eller genom att visa informationen som användaren vill ha från databasen.

Vi har använt PHP i vårt projekt till att föra in medlemmar, transaktioner och evenemang i databasen, så att det sparas och informationen lagras under en längre tid. PHP kan inte spara information under en längre tidsperiod, utan endast då applikationen excekveras kan det lagras information inom PHP.

Detta leder till att förhållandet mellan PHP och MySQL blir ett utmärkt samarbete där PHP gör det MySQL inte kan och vice versa.

(Valade 2009, s 10-20)

2.4 MySQL

MySQL är världens populäraste databasmjukvara med öppenkällkod. Nyckelord för MySQL är hastighet, pålitlighet och användarvänlighet. Dessa nyckelord och faktumet att MySQL och PHP går extremt bra ihop bidrar till att MySQL är vårt verktyg i utvecklingen av denna webbapp.

Då man som sagt kombinerar PHP med MySQL har man en utmärkt möjlighet att utföra invecklade operationer och sedan lagra resultatet av dessa i en databas. Dessutom kan man lätt söka upp information med hjälp av invecklade förfrågningar till databasen, vilket garanterar att man får just den informationen man vill ha, så länge man vet exakt hur man skall be om den.

(Valade 2009, s 10-20)

3. Objektorienterad programutveckling

Som utvecklingsmetod valde vi objektorienterad programutveckling. Genom att använda denna metod, som har ökat i popularitet på senaste tiden, så har vi en god grund om vi vill börja programmera i andra språk, eftersom metoden har ett ganska brett användningsområde i dagens läge.

I detta kapitel går vi igenom teorin för objektorienterad programutveckling för att ge läsaren lite mera insikt i den metoden vi valt. Vi börjar med att ta upp några grundläggande begrepp så att läsaren lättare hänger med när vi senare tar upp de tre olika faserna i objektorienterad programutveckling.

3.1 Grundläggande begrepp

När man använder objektorienterad programutveckling bygger man upp program med en samling olika objekt. Dessa objekt består av en samling egenskaper. För att ge läsaren en klarare bild av hur vi framskred så tar vi upp objekt och klasser i denna del av kapitlet. (Skansholm 2011, s 201)

3.1.1 Objekt

Ett objekt är en modell av ett tänkt föremål. Detta föremål kan vara verkligt eller påhittat. Ett objekt består av två olika slags egenskaper: attribut och operationer. För att få ett exempel kunde ett objekt som skall vara en television konstrueras. Detta objekt kunde kallas *Television*. Objektet behöver sedan en samling egenskaper som bestämmer funktionaliteten för objektet. Egenskaperna för objekten är som tidigare nämnts indelat i två olika grupper, attribut och operationer. (Skansholm 2011, s 202)

3.1.2 Attribut

Attribut används för att hålla reda på objektens status då objekt brukar ha ett tillstånd som kan ändras senare under exekvering. Attribut kan det finnas en hel samling av och dessa kan ha ett antal olika begränsningar, värden och så vidare.

Till exempel om en television vore ett objekt så skulle den kunna heta *Television.* Detta objekt skulle kunna ha attributen *strömläge* och *kanal.* Attributen skulle lagra information för vilket strömläge televisionen har och vilken kanal den för tillfället är på. Här skulle det finnas två alternativ för strömläget: på och av. Kanalerna skulle däremot i princip kunna vara vilken nummer som helst. (Skansholm 2011, s 202)

3.1.3 Operationer

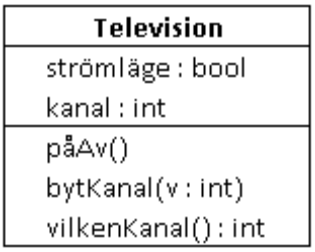
Operationer däremot används för att utföra någonting på objektet och såvida ändra ett objekts attribut.

Om vi då vill ändra våra tidigare attribut vi tagit upp så behövs det en operation för att ändra *strömläge* och en för *kanal*. Dessa operationer kunde då helt enkelt heta *påAv()*, *bytKanal(kanal)*. Inom dessa parenteser kan man sedan beroende på om operationen tar emot, ange värden. Så om man vill byta kanal till kanal 5 så skulle man anropa operationen enligt följande: *bytKanal(5)*. Dessutom skulle det kunna läggas en till operation för att hjälpa användaren att se vilken kanal som är aktuell för tillfället. Denna operation kunde heta *vilkenKanal()* och skulle ge tillbaka ett heltal till användaren. (Skansholm 2011, s 202)

3.1.4 Klasser

När vi väl har ett objekt, en modell av ett föremål, så måste det föras in i programmet. När man för in objekt i program så beskrivs de på ett särskilt sätt, ganska liknande som när vi gjort exemplet för televisionen. Inne i programmet så blir vårt objekt dock en klass. Med hjälp av klasser kan man ha objekt med samma egenskaper men som skiljs åt då man angett ett annat variabelnamn åt de två klasserna.

Nedanför i figur nr 3 ser vi ett klassdiagram på vårt exempel *Television*. I figuren kommer det klart fram hur både attributen och operationerna är indelade. Detta diagram är uppbyggt enligt Unified Modeling Language som är brett använt i dagens läge.



Figur 3. Klassdiagram av exemplet

För att ännu understryka skillnaden mellan objekt och klasser, så kan det nämnas att klass är en sorts ritning av ett objekt och ett objekt är sedan skapat enligt denna ritning.

(Skansholm 2011, s 202-203)

3.2 Objektorienterad analys

I första fasen med objektorienterad analys, gör man upp en grov design av de objekt som kommer användas och vilka egenskaper samt funktioner dessa objekt skall ges. Här tas även upp relationer bland de olika objekten och det läggs upp hierarkier för objekt som ärver egenskaper av andra objekt och hur dessa kopplas ihop.

Under denna fas får man i princip en grov skiss av systemet och vilka delar det består av. I senare faser gör man sedan denna skiss mer konkret.

3.3 Objektorienterad design

När man väl analyserat och gjort upp en plan för arbetet och har allt väl uttänkt så ger man sig in på andra den fasen; objektorienterad design. Här tar man det material som man fått ihop från första fasen och gör det materialet, som tidigare sagt, betydligt mer konkret.

De första två faserna kan vara iterativa och man kan vid behov gå igenom någon fas igen då det är smidigt att övergå från analysfasen till designfasen. Typiskt så funderar man i analysfasen på just *vad* programmet skall göra medan man sedan i designfasen tar *vad*et och börjar fundera på *hur* det skall göras inom programmet.

Denna fas brukar delas in i två olika delar: systemdesign samt objektdesign.

Systemdesign går ut på att fundera vad programmet som helhet skall utföra. Här bestäms i hurdana delar programmet skall delas in i, samt hur dessa delar skall kommunicera med varandra.

Objektdesign går sedan, som namnet klart antyder, ut på att planera de olika objekten som systemet består av. Här tar man den grova skissen som görs upp i analysfasen och börjar ge de olika planerade objekten betydligt mer egenskaper och detaljer. Här planeras också operationerna för objekten noggrant igenom och det bestäms vilka parametrar operationerna skall ta in. En viktig sak att påpeka här är även att det bör funderas ut vilka objekt som kan återanvändas för att få ett så effektivt program som möjligt. (Skansholm 2011, s 209-210)

3.4 Objektorienterad programmering

Vid tredje fasen börjar själva programmeringen då man efter mycket utförlig planering börjar få något konkret till stånd. Målet med fasen är att förverkliga programmet. Här eftersträvas ett så välskrivet program som möjligt. De klassiska ”kraven” på ett bra program är att det skall vara: korrekt, effektivt, återanvändningsbart och ändringsbart.

Vad man menar med att ett program bör vara korrekt, är att det skall utföra koden felfritt och som avsett. Detta är självklart det viktigaste kravet för om programmet har en massa fel och problem så räddas inte programmet av andra goda sidor.

Att ett program skall vara effektivt betyder att resurser som programmet har till hands skall utnyttjas så väl som möjligt. Beroende på programmet man utvecklat är detta en otroligt viktig punkt att ta i beaktande.

Om man väl har ett korrekt och effektivt program, har man redan kommit ganska långt. Men för att ännu vidareutveckla programmet så bör man kunna återanvända delar av programmet för att underlätta jobbet i framtiden då man utvecklar något liknande. Har man ett återanvändningsbart program så hjälper det betydligt då man har olika delsystem som kan återanvändas i ett annat program vilket leder till att programutvecklingen blir betydligt snabbare och således även förmånligare för kunden.

Denna fas, likt tidigare faser, kan också vara iterativ och man kan relativt smidigt gå tillbaka och ändra sina tidigare planer om man märker att någonting inte riktigt fungerar som tänkt.

(Skansholm 2011, s 210-212)

4. Planering av systemet

I detta kapitel tas det upp hur vi gått igenom de två första planeringsfaserna av objektorienterad programmering och vilket tillvägagångssätt vi använt för detta arbete.

Vi tar även upp hur vi ändrat lite på metoden och hur vi har lånat vissa delar, som vi ansett vara lämpliga för projektet, från andra metoder vi varit bekanta och bekväma med under vår studietid.

4.1 Objektorienterad analys

I denna fas skulle man typiskt redan börja göra upp grova planer på objekt och fundera på relationer mellan objekten. Men då vi använder PHP som inte är så dynamiskt i exekveringen, som andra programmeringsspråk valde vi att i denna fas av utvecklingen börja lägga själva grunden till arbetet så att vi snabbt kommer igång.

Vi gjorde tre olika saker i första fasen. Dessa saker var att skriva en kravspecifikation för arbetet för att få ut de centrala kraven och målen för arbetet, att göra upp en databasstruktur, samt att göra en grov design av systemets utseende.

4.1.1 Kravspecifikation

Detta dokument har i vår utbildning fungerat som ett slags kontrakt mellan en projektgrupp och kunden. Här strävade vi till att så tydligt som möjligt behandla alla de olika delar som kunden var ute efter för att få en klar bild av vad arbetet skall resultera i.

De viktigaste punkterna som behandlas i detta dokument är:

- Bakgrund
- Effektmål
- Intressenter
- Systemet
- Krav

I inledningen av detta arbete tog vi kort upp några delar: bakgrund, mål, prioriteringar och intressenter, dessa delar jobbades fram ur kravspecifikationen. Även många andra viktiga saker behandlas i detta dokument.

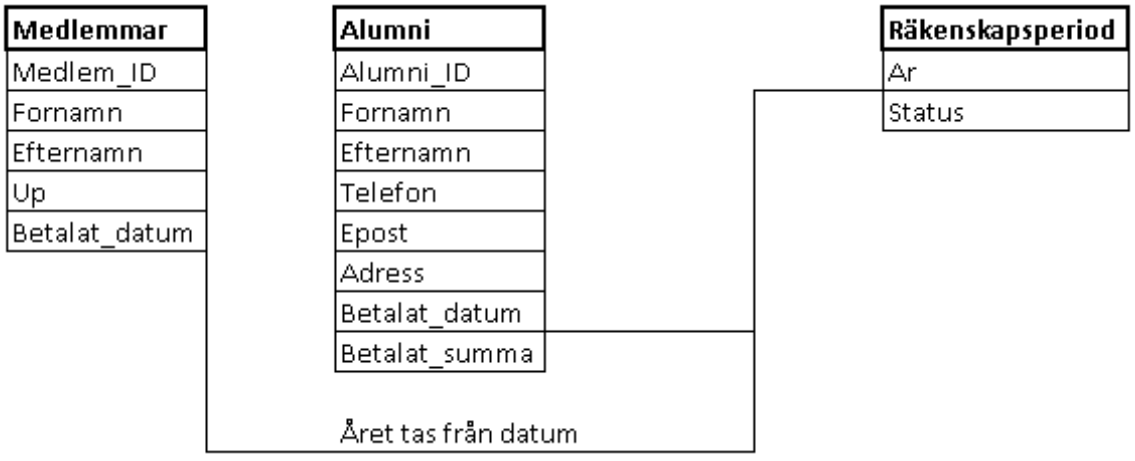
Med hjälp av kravspecifikationen kunde vi lätt plocka ut de olika delarna av systemet som skulle programmeras, fick vi även en bättre överblick över hur systemet skall fungera. Genom kravspecifikationen.

När vi fått ut de viktigaste delarna började vi jobba på att rita upp ett diagram för databasen. Detta diagram gav oss sedan en god grund för att göra upp databasen och fundera ut olika klasser i nästa fas då vi fått en stor överblick på de olika delarna av systemet.

4.1.2 Databasstruktur

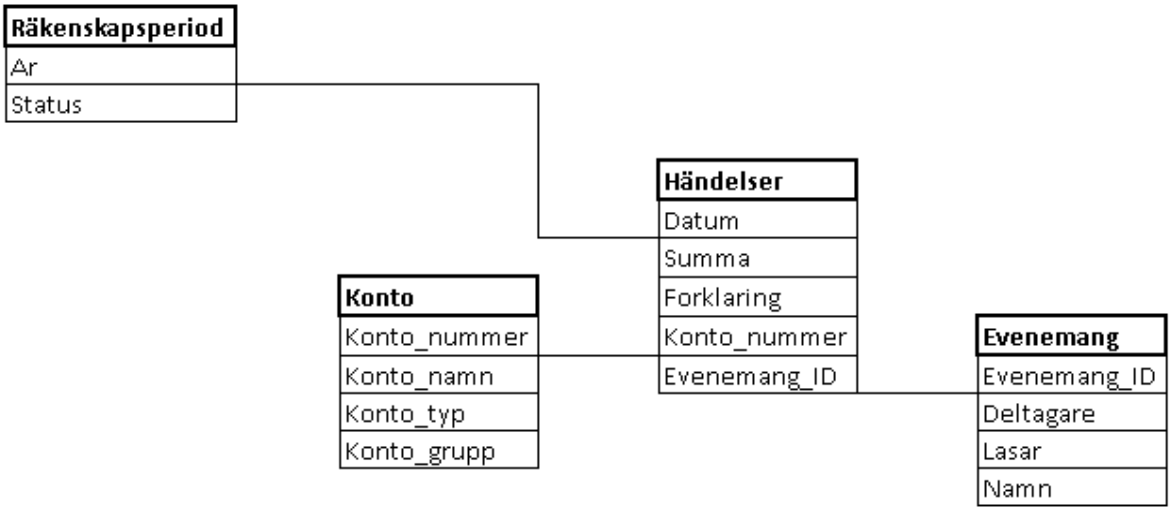
Uppbyggnaden av databasen på ett bra sätt är mycket viktigt i detta arbete. Det här resulterade i att planeringen av databasen tog en hel del tid, även då vi hade en väldigt god grund för planering i kravspecifikationen. Vår ursprungliga plan höll inte och den slutliga databasstrukturen blev inte helt som vi hade planerat, men tack vare vår projektmetod orsakade detta inte större problem.

Medlemsregistret var betydligt enklare att lägga upp än bokföringen. Här finns inte lika många delar som går in i varandra. Medlemsregistret har som tidigare nämnts, två olika sorters medlemmar. Vi har de vanliga Ekenäs Kåravdelnings medlemmarna, samt alumnimedlemmarna. Vi måste ha mer information av alumnimedlemmarna på grund av att de inte studerar längre. De summor som betalas in från de två olika medlemmarna läggs senare, då man lägger upp ett bokslut, automatiskt in i bokföringen. Som figuren nr 4 nedan visar så plockas året direkt från datumet då medlemsavgiften är betalad. Från figuren får man även en god överblick över hur många mer fält som krävs för en alumnimedlem.



Figur 4. Databasstruktur för medlemsregistret

Bokföringen krävde lite mera planerande. Många små delar, som man inte hade räknat med spelade stor roll och vållade problem när systemet automatiskt skulle räkna ut summor. Detta resulterade i att vi senare var tvungna att fullständigt tänka om för att få alla att skulle samarbeta med varandra.



Figur 5. Databasstruktur för bokföringen

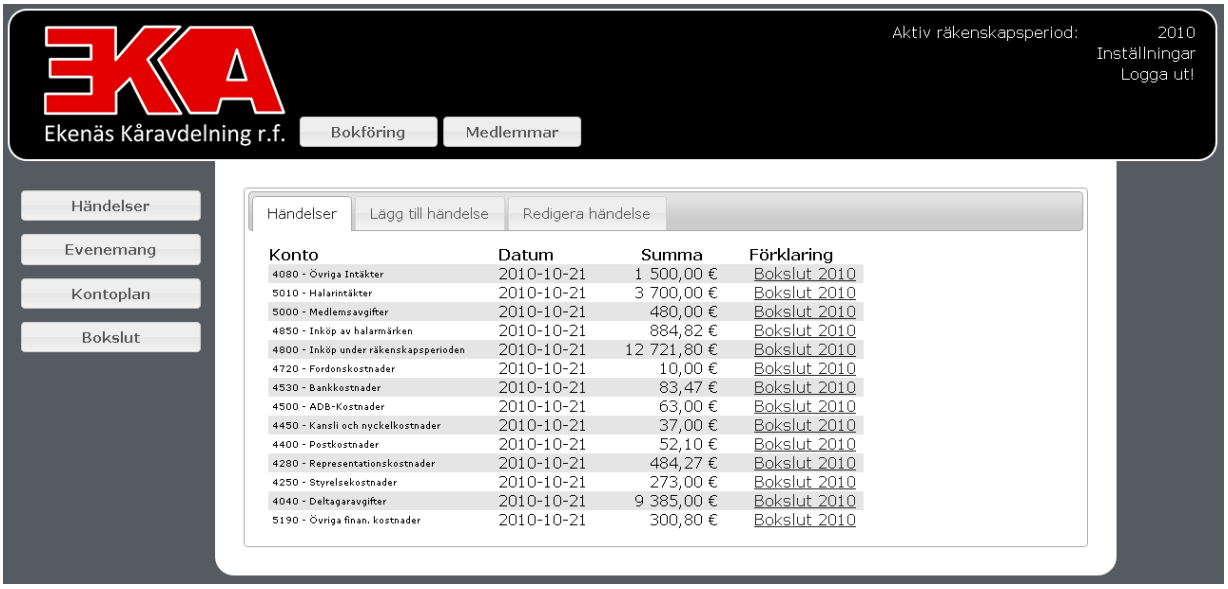
Som figuren nr 5 ovan visar så plockas alla bokföringshändelser in i rätt räkenskapsperiod med hjälp av att plocka ut året från datumet. På detta sätt kopplas händelserna till rätt räkenskapsperiod. Händelserna behöver i sin tur ett konto, så på varje händelse matas det in ett kontonummer så att man får dessa två ihopkopplade. Evenemang för händelserna är däremot valfria och man kan välja om händelsen behandlar ett evenemang eller inte.

Vi hade i början endast nummer, namn och typ för konton men det blev ganska snart klart att vi måste lägga till en grupp, så vi får korrekt uppställning på bokslutet. Här skulle man även kunna utveckla numreringen av konton på så sätt att de kommer i rätt ordning men vi valde att lägga olika typer för kontona istället då detta gick smidigare.

4.1.3 Design av systemet

Designen var det första konkreta vi gjorde och vi bestämde oss att göra en väldigt grov version redan i början av utvecklingen, för att få något att jobba med så tidigt som möjligt.

Designen hade inte hög prioritet i vårt arbete. Vi siktade in oss på att göra en enkel och stilren design som inte hade något extra i sig. Tanken var att det skulle vara lätt att hitta det man är ute efter. Färgerna vi använde oss av var Ekenäs Kåravdelnings färger, svart och rött som bas, med lite vitt så att texten lättare kommer fram. Nedanför i figur 6 ser vi hur designen ser ut.



Figur 6. Designen av systemet

Designen, som man ser i figur 6 är inte alltför invecklad, utan vi satsade som sagt på enkelhet och att allt skall vara så tydligt som möjligt. Designen ändrades aldrig radikalt under hela utvecklingsprocessen, utan den har ganska långt hållit sin ursprungliga form. Här ser man även användningsområdet för jQuery-UI biblioteket då vi gjort alla knappar enhetliga och stilrena.

Första stora delen av sidan är banner-delen ovanför innehållet, som bär Ekenäs Kåravdelnings logo. Här kommer det klart upp vilken räkenskapsperiod man för tillfället är inne på, samt en länk till inställningar och en för att logga ut ur systemet. Här är också knapparna till de två stora olika delarna av systemet, nämligen bokföringen och medlemmar.

Sidomenyn är, som figuren visar, rätt självklar och länkar till de olika funktionerna inom bokföringen respektive medlemsregistret. Här har vi också försökt dela in sakerna så logiskt som möjligt för att användaren inte skall ha några problem då det bör ändras på någonting.

När man sedan gått in på någon del, till exempel händelserna, så får man en lista på de händelser som senast skett inom den aktiva räkenskapsperioden. Sedan har vi använt jQuery-UI igen för att få en stilren indelning på listan, inläggning samt redigering av händelser, genom att dela in dessa i olika flikar som man lätt kan välja bland enligt önskemål.

Alla olika delar av systemet fungerar ganska långt på liknande sätt och har en enhetlig design så det blir tydligt och användaren känner igen sig och snabbt vänjer sig att använda systemet.

Designen skulle självklart kunna förbättras och många olika justeringar skulle kunna göras, men då detta system endast är internt och kommer att användas av en eller två personer per år, var designen inte vår högsta prioritet och därför blev designen som den blev.

4.2 Objektorienterad design

Efter en del planering i första fasen var det dags att gå vidare till nästa fas. I den här fasen började vi fundera ut i vilka delar systemet delas upp och hurudana klasser som behövs för att systemet skall fungera enligt våra önskemål. Vi började med att dela funktionaliteten i olika grupper och sedan fokuserade vi på de grupper vi fick fram för att få fram de olika klasserna inom grupperna.

4.2.1 Uppdelning av systemet

Då det tidigare varit ganska klart för oss att systemet delas in i två större olika delar så började vi fundera hur dessa två delar ytterligare skulle spjälkas upp i mindre bitar.

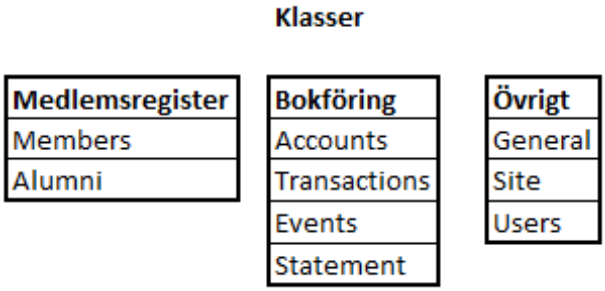
Medlemsregistret var rätt så lätt att dela upp i olika bitar, då det bestod av en del för ordinarie medlemmar och en för alumnimedlemmar. Här var det relativt enkelt att plocka ut de saker som behöver lagras för användning. Under dessa två större kategorier var det sedan rätt självklart att vi vill ha en lista på medlemmarna, och vi vill kunna lägga till och redigera medlemmar. Så detta gjordes det rum för i planerna.

Det som krävde betydligt mera planering var bokföringsdelen. Här måste vi först fundera ut i hur många olika delar systemet skall spjälkas upp och sedan hur delarna skall planeras. Här blev en hel del fler delar än medlemsregistret, då vi började med kontoplanen och jobbade upp till bokföringshändelser och evenemang. Dessa delar hade, som medlemsregistret, funktionalitet för att ge listor på olika saker, samt möjlighet att lägga till och redigera data.

När alla övriga delar var färdigt planerade, kunde vi ge oss på delen som krävde mest planering och som orsakade mest problem under själva utvecklingen; nämligen bokslutet. Vi kom ganska snabbt fram till att vår originella databasstruktur inte kommer att hålla då vi, som tidigare nämnt, blev tvungna att lägga till ett fält för kontogrupp till kontona, för att få en sådan uppställning på bokslutet, som vi önskade.

4.2.2 Planering av klasser

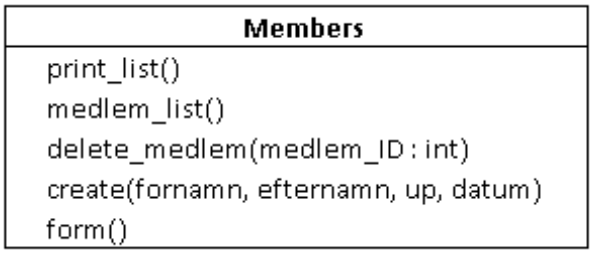
Då vi hade systemet uppdelat i olika delsystem blev det enkelt att göra upp planer för klasserna vi skulle använda oss av. Dessa blev ganska långt lika som tidigare uppdelningen.



Figur 7. Klasser i systemet

Som figuren nr 7 visar så blev systemet i stort sett indelat i tre olika delar. Här är de två delsystemen som tagits upp tidigare bokföringen och medlemsregistret. Men sedan behövs även en hel del annan funktionalitet för systemet, som blir lite av sin egen grupp.

Alla våra klasser är i stort sett ganska lika, med en hel del skillnader på detaljnivå. I figur nr 8 på nästa sida ser vi hur vår klass *Members* ser ut enligt planerna. Här kan vi ännu poängtera att vi sällan använder många attribut i våra klasser, då vi har informationen vi ändrar i en databas. Detta leder till att informationen återges åt användaren via operationer som *medlem_list()* och användaren får välja en post som skall redigeras eller tas bort från denna lista.



Figur 8. Diagram av klassen Members

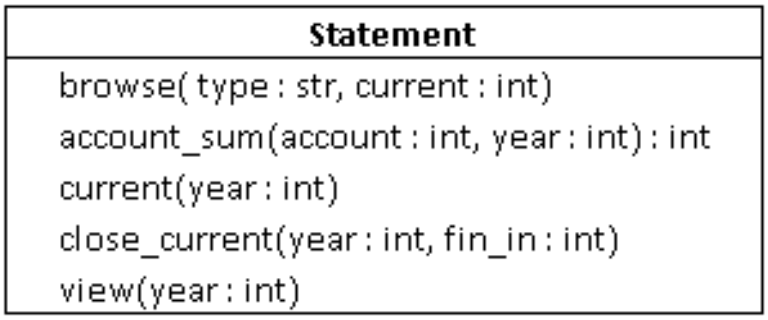
Alla klasser är som sagt ganska lika varandra, de centrala delarna i varje klass är operationerna för att återge en lista av alla databasposter åt användare, möjlighet att redigera enskilda poster, samt återgivningen av formuläret för att skapa eller redigera användare.

Vissa klasser blev naturligtvis helt annorlunda än de andra eftersom de gör olika saker. Dessa klasser är i princip alla klasser i den övriga kategorin samt klassen för bokslutet: *Statement*.

Klassen *Site* har operationer för att återge själva webbsidan och är således systemets grund och botten. Detta betyder att den har en hel del olika operationer för att ge ut HTML kod och några operationer som tar in parametrar och ändrar på sidan beroende på vad den aktuella sidan kräver.

Klassen *Users* innehåller all funktionalitet för autentisering på hemsidan, vilket är viktigt då allt ligger bakom lösenord. Klassen *General* blev en salig samling av allt som kan krävas vid bokföringen men även på andra ställen. Här har vi funktionalitet för att söka upp den aktiva räkenskapsperioden och således även få ut läsåret i samma veva. Dessutom finns det operationer för att återge listor och formulär för konton, evenemang och bokföringshändelser.

Bokslutet krävde en hel del planering, då det var ganska invecklat. I figuren nr 9 nedanför ser vi ett klassdiagram av klassen i fråga. Dessa operationer kom vi fram till i slutändan av planeringen. Här ser man att vi har operationer för allt vi önskar, för att bläddra igenom olika bokslut, för att se både gångna och aktuella bokslut. Operationen som kanske inte är så självförklarlig är *account_sum(account, year)* som räknar ihop totala summan från ett konto med hjälp av parametrarna den tar in för kontonummer och år. Denna operation är väldigt viktig då den gör det mesta arbetet i bokslutsdelen av systemet.



Figur 9. Diagram av klassen Statement

5. Utveckling av systemet

När vi väl hade planerat i sådan utsträckning att vi ansåg att vi hade allt material som behövdes så påbörjade vi utvecklingsfasen. Denna fas tog självklart mest tid och innefattade flest motgångar. Det blev även en hel del omplanerat då vi i utvecklingsfasen kom fram till att alla våra planer inte nödvändigtvis fungerar.

5.1 Filstruktur

För att hålla reda på filerna, och således ge en god struktur åt koden och slutliga produkten, var filstrukturen kanske inte precis livsviktig men den underlättade ändå arbetsprocessen betydligt när vi lade upp en klar plan för filstrukturen i ett tidigt skede.

Som figuren nr 10 nedanför visar valde vi att ha bara de sidor som visas för användaren i rotmappen av systemet, medan vi sen plockat in övrig kod under en mapp som heter *core*, det vill säga systemets kärna. I denna mapp finns filerna *config.php* och *init.php*. Filen *config.php* innehåller information för att kontakta databasen. Den andra filen, *init.php,* sköter initieringen av programmet och innehåller även inkluderingen av alla klass-filer.

Mapp nivå 1	**root**					
Mapp nivå 2		**core**				
Mapp nivå 3			**classes**	**styles**	**gfx**	**jquery-ui**
	accounts.php	config.php	class_accounts.php	print.css	logo.png	jquery filer
	alumni.php	init.php	class_alumni.php	style.css		
	economy.php		class_events.php	tables.css		
	events.php		class_general.php			
	home.php		class_members.php			
	index.php		class_site.php			
	login.php		class_statement.php			
	logout.php		class_transactions.php			
	members.php		class_users.php			
	print.php		class_years.php			
	statement.php					
	transactions.php					

Figur 10. Filstruktur

Filerna för de olika klasserna ligger under mappen *classes* i *core* mappen och här finns filer för alla klasser. Övriga mappar under *core* mappen är jquery-ui som innehåller allt som behövs för jquery. Mappen för våra stylesheets ligger under *styles* och mappen för bilder och grafik heter *gfx*.

För att ännu tydligare visa läsaren vad filen *init.php* gör så visar vi den i kod nr 1 nedan. Det här är den viktigaste filen i vårt system. I figuren ser vi att den inkluderar alla övriga filer vi behöver, skapar anslutningen till databasen, samt initierar alla de olika objekt som är skapade. Denna fil drar samman alla filer till en stor familj.

Kod 1. Filen init.php

```php
<?php
ob_start();
require_once("classes/class_users.php");
require_once("classes/class_general.php");
require_once("classes/class_site.php");
require_once("classes/class_years.php");
require_once("classes/class_accounts.php");
require_once("classes/class_transactions.php");
require_once("classes/class_events.php");
require_once("classes/class_statement.php");
require_once("classes/class_members.php");
require_once("classes/class_alumni.php");
require_once("config.php");
$mysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD, DB_DATABASE);
$jquery = "<link rel='stylesheet' href='http://code.jquery.com/ui/1.10
          <script src='http://code.jquery.com/jquery-1.9.1.js'></scr
          <script src='http://code.jquery.com/ui/1.10.3/jquery-ui.js
$site = new Site;
$users = new Users($mysqli);
$general = New General($mysqli);
$years = new Years($mysqli);
$accounts = new Accounts($mysqli);
$events = new Events($mysqli);
$transactions = new Transactions($mysqli,$events);
$statement = new Statement($mysqli);
$members = new Members($mysqli);
$alumni = new Alumni;
?>
```

5.2 Login

För att skydda informationen och se till att endast behöriga kommer åt systemet så måste vi utveckla ett login-system. Detta sker med hjälp av PHP, HTML och JavaScript. Vi hade från början tänkt lagra användarinformationen i databasen, men på grund av diverse problem vi hade, valde vi att göra något enklare för tillfället, så vi kommer vidare med utvecklingen.

Senare då vi hade det mesta klart, hade vi dock tidsresurser för vidareutveckling av login systemet och vi utvecklade således ett säkrare system där användarinformationen faktiskt finns i databasen och är krypterad.

Det första man ser när man försöker komma åt systemet är ett login formulär, eftersom man inte har något att göra i systemet om man inte är behörig.

5.2.1 Inloggning och lagring av användarinformation

Själva inloggningen är relativt enkel; man kommer till en login sida, fyller i informationen och trycker på en knapp. Sedan kontrollerar koden att allt stämmer och man blir inloggad för en bestämd tid.

Koden för inloggningen ligger huvudsakligen i tre filer nämligen *index.php, login.php* och *config.php*. Filen *index.php* innehåller själva login formuläret. Filen *login.php* kontrollerar om användaren redan är inloggad och innehåller några kommandon för vad som skall hända då användaren trycker på logga-in-knappen.

Före koden gör någonting alls, så kontrollerar den om man är inloggad, detta sker, som man ser i bilden nedan, med hjälp av klassen *users* operation *check()* om det redan finns en cookie i webbläsaren. I kod nr 2 nedanför ser vi precis hur funktionen *check()* fungerar.

Kod 2. Funktionen check()

```
// LOGGED CHECK FUNCTION
public function check() {

    if(isset($_COOKIE['ID_EKA']) && isset($_COOKIE['KEY_EKA'])) {

        $query = "SELECT * from Anvandare
        WHERE Anvandarnamn = '".$_COOKIE['ID_EKA']."'";
        if ($result = $this->mysqli->query($query)) {
            while($row = $result->fetch_assoc()) {
                $user2 = $row['Anvandarnamn'];
                $pass2 = $row['Losenord'];
            }
        $result->free_result();
        }

        if($_COOKIE['ID_EKA'] == $user2 && $_COOKIE['KEY_EKA'] == $pass2) {
            return true;
        }
```

Om det finns en cookie och all användarinformation stämmer så visas själva webbapplikationen, eftersom man ännu är inloggad från en tidigare session. Om funktionen *check* däremot ger falskt värde, så kommer formuläret för inloggningen fram och sedan väntar koden på att man trycker på inloggningsknappen.

När användaren trycker på knappen utförs login funktionen. Som vi ser från kod nr 3 nedan, krypterar funktionen först om det lösenord som har matats in till md5 format. Sedan söker den upp en användare med samma användarnamn från databasen. Om namnet sedan hittas så läggs det till två strängar till lösenordet för att få lite mera säkerhet. Sedan om detta stämmer så läggs det två cookies för att visa att användaren är inloggad.

Kod 3. Funktionen login($user, $pass)

```
// LOGIN FUNCTION
public function login($user, $pass){

    $pass = md5($pass);

    $query = "SELECT * from Anvandare WHERE Anvandarnamn = '$user'";
    if ($result = $this->mysqli->query($query)) {
        while($row = $result->fetch_assoc()) {
            $user2 = $row['Anvandarnamn'];
            $pass2 = $row['Losenord'];
        }
        $result->free_result();

        $pass = HASH_1 . $pass . HASH_2;

        // CHECK INPUT INFORMATION AGAINST STORED INFORMATION AND CREATE COOKIES
        if($user == $user2 && $pass == $pass2) {
            setcookie("ID_EKA", $user, time()+3600*3);
            setcookie("KEY_EKA", $pass, time()+3600*3);
            return true;
        }
        else if($pass != $pass2) {
            die('Fel lösenord');
        }


    }
```

Dessa två funktioner lägger grunder för login-systemet. Men en annan viktig funktion som vi tar upp i nästa kapitel är en funktion för vidareföring av användaren.

5.2.2 Vidareföring av användaren

Problemet vi hade med PHP var att den inbyggda funktionen för att föra vidare användaren till en annan sida vägrade fungera på servern, fastän den fungerade på en lokal lösning. Vi kämpade länge med detta problem utan resultat. Efter att suttit länge med detta problem valde vi att införa några rader JavaScript för att föra användaren vidare.

Efter att vi väl fått själva koden att fungera så skapade vi en operation i klassen *General* för att lätt komma åt funktionen i framtiden. Kod nr 4 nedanför visar denna funktion. Här ser man tydligt att den tar in en valfri parameter för destination och beroende på parametern som ges så förs användare vidare enligt önskemål. Ges ingen parameter så laddas den aktuella sidan om, detta för att exekvera kod som kräver att sidan laddas om.

Kod 4. Funktionen redirect($dest

```
public function redirect($dest = '') {

    if($dest == 'logged') { $url = 'home.php'; }
    else if($dest == 'unlogged') { $url = 'index.php'; }
    else if(!$dest) { $url = $_SERVER['PHP_SELF']; }
    else { $url = $dest; }
    echo
    "<script type='text/javascript'>
    <!--
    window.location = '".$url."'
    //-->
    </script>";

}
```

5.3 Uppbyggnad av sidan

Efter att vårt autentiseringssystem blivit funktionellt kunde vi börja lägga grunden till själva systemet. Detta gjorde vi genom att ta vår design från planeringsfasen och sedan börja förverkliga den med hjälp av HTML och CSS.

Denna process tog en hel del tid då det blev en del finjustering av de olika elementen som sidan byggdes upp på. Senare tog det även tid att plocka in formatering för de olika tabellerna då vi ville ha ett enhetligt utseende för de flesta tabellerna som används inom systemet.

När vi hade en grundsida som fungerade för alla olika delar av systemet var det dags att plocka in koden till ett objekt för att centralisera så mycket struktur av systemet som möjligt på en plats. Allt detta samlades i den tidigare nämnda *Site* klassen.

Genom att implementera denna klass har vi sett till att då något behöver ändras på i systemet kan vi ändra det på ett ställe istället för att ändra det på alla de olika filer som det annars skulle ligga i.

Här finns dock också lite funktionalitet på platser vi kom fram till att behövs på, till exempel då sidomenyn ändras beroende på om man är på bokförings- eller medlemsdelen. Funktionalitet finns även för att ändra mängden flikar man behöver på en viss sida, samt möjlighet att använda sig av olika JavaScript tillämpningar i *head* sektionen av sidan.

Koden nr 5 nedan innehåller funktionen *Menu(type)* från *Site* klassen. Här kan vi klart och tydligt se att funktionen tar emot en parameter för typ av meny som önskas. Här finns som val två olika typer, en för medlemsregistret och en för bokföringsdelen av systemet. Beroende på vad man ger för parameter åt funktionen får man en önskad meny. Genom att ha båda menyerna i en fil sparar vi mycket tid om något bör ändras vare sig en viss funktion läggs till eller tas bort.

Kod 5. Del av site klassen

```
public function Menu($type) {

    if($type == 'economy') {
        echo
            "<div id='controls'>
                <ul id='sidemenu'>
                    <li><a href='transactions.php' class='button'>Händelser</a></li>
                    <li><a href='events.php' class='button'>Evenemang</a></li>
                    <li><a href='accounts.php' class='button'>Kontoplan</a></li>
                    <li><a href='statement.php' class='button'>Bokslut</a></li>
                </ul>
            </div>";
    }
    else if($type == 'members') {
        echo
            "<div id='controls'>
                <ul id='sidemenu'>
                    <li><a href='members.php' class='button'>Medlemmar</a></li>
                    <li><a href='alumni.php' class='button' >Alumni</a></li>
                </ul>
            </div>";
    }

}
```

Denna klass innehåller också en del annan funktionalitet, men för att hålla denna del kort går vi inte in på det närmare. Med hjälp av denna klass har vi, som tidigare sagts, mycket lättare att göra små justeringar på designen och uppbyggnaden av systemet, vare sig det är för funktionella orsaker eller för att något stör vårt öga.

För att närmare illustrera hur mycket lättare de enskilda filerna som visas går att ändra med hjälp av *Site* klassen så har vi tagit ett exempel i koden nr 6 nedan.

Kod 6. Websida med site klassen

```
<?php
require_once('core/init.php');
$site->doctype;

$check = $users->check();if($check == true) {
$site->Head();
$site->Banner($general->active);
?>

<div id="content-wrapper">

<div id="content" style='float: center;'>

<p>Välkommen till Ekenäs kåravdelnings bokföringssystem komplett med medlmesregister.</p>

</div>

</div>

<?php
$site->Footer();
}
else if($check == false) {$general->redirect('unlogged');}
?>
```

I koden nr 6 ser vi att vi får en enskild webbsida att rymmas på cirka 30 rader kod i en fil då vi har det flesta lagrat i en klass. För att få detta i perspektiv så kan vi poängtera att en typisk sida kan ta allt från 60 rader kod eller långt uppåt. Detta underlättar arbetet betydligt då man skall in i de olika filerna och lägga till funktionalitet för olika delar av systemet, eller när man märker att någonting är fel på designen av sidan och man behöver ändra den.

5.4 Medlemsregister

Medlemsregistret är, som tidigare nämnts, inget nytt koncept i föreningen. Vi hoppas att, med denna metod, kunna ha bättre kontroll på föreningens medlemmar i framtiden.

Vi har dessutom integrerat ett nytt register i detta projekt som föreningen inte använt tidigare, nämligen ett register på alumnimedlemmar. Det här innebär att vi kommer att ha två olika register, som automatiskt uppdaterar bokföringen när man lagt in personer som betalat sin medlemsavgift.

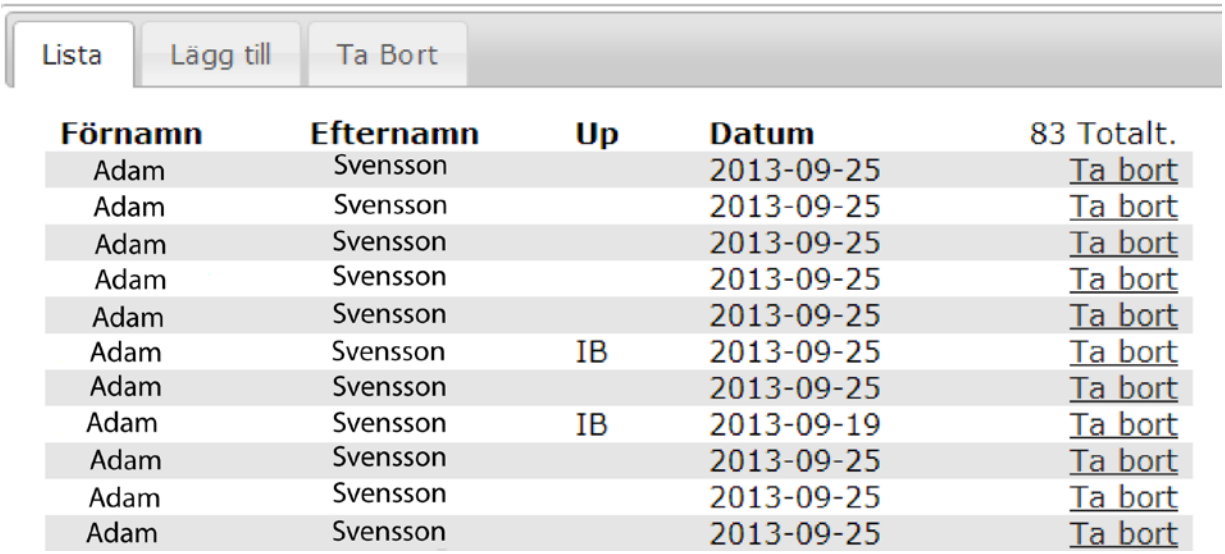
I detta kapitel tas det upp hur vi gått till väga när vi utvecklade funktionaliteten för medlemsregistret och dess två olika sorters medlemmar.

5.4.1 Ekenäs Kåravdelnings medlemmar

Inkomsterna från medlemmarna i föreningen är grunden för hela årsbudgeten. Ekenäs Kåravdelning är ingen vinstinbringande förening, så det blir stort sett ingen vinst kvar från föregående år. Inkomsterna som kommer in från medlemmarna är viktiga för att vi skall kunna hålla igång vår verksamhet. Därför är det viktigt att de medlemmar som verkligen betalar får vad föreningen lovar åt dem. För att upprätthålla detta register på ett bra sätt valde vi förutom bokföringen att även integrera detta på sidan.

Medlemsregistret fungerar så att man skriver in personen i formuläret. Personen lagras i databasen med den information man skriver in. Personen får en unik ”*medlem_ID*” i databasen, denna ID behövs för att vi skall komma åt exakt den person vi vill, vi kan inte anropa namn eller dylikt i koden eftersom personer kan ha samma namn.

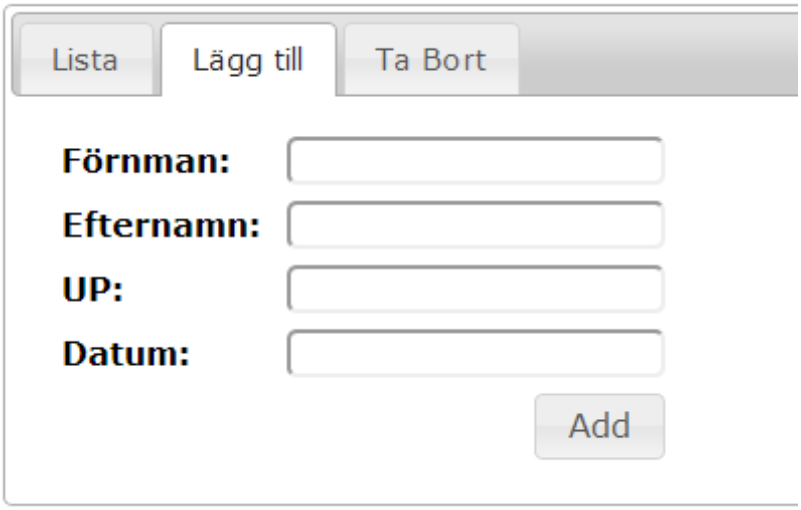
I figuren nr 11 nedan ser vi hur listan över medlemmar återges åt användaren. Då vi inte har många fält med information om medlemmarna så har vi möjlighet att skriva ut all information som finns i databasen. På figuren syns även länken för att ta bort en medlem om man gjort något skrivfel.



Figur 11. Lista på medlemmar

Vi kräver inte mycket information när vi lägger in nya föreningsmedlemmar. Alla medlemmar går i Yrkeshögskolan Novia, Raseborg, så vi får lätt kontakt med dem runt om i skolan samt med hjälp av skolans e-post om det finns behov för det.

I figuren nr 12 nedan syns formuläret för att lägga till medlemmar. Detta formulär har ganska få fält eftersom vi som tidigare poängterats, inte behöver så mycket information från ordinarie medlemmar.



Figur 12. Lägg till medlem

Detta register var relativt enkelt att utveckla och består i princip av en HTML tabell och en MySQL förfrågning med lite PHP inblandat. Det finns även ett alternativ att direkt från webben skriva ut en lista på alla medlemmar i alfabetisk ordning. Det finns också ett alternativ att fullständigt ta bort en medlem från databasen, vilket ”*medlem_ID*” används till.

5.4.2 Alumnimedlemmar

Alumniregistret skiljer sig naturligtvis från det vanliga medlemsregistret som är avsett för ordinariemedlemmar. Eftersom alumnimedlemmar redan har blivit utexaminerade från Yrkeshögskolan Novia, Raseborg, så är vi tvungna att samla in mer information av dem för att kunna nå dem när föreningen skickar ut inbjudningar till olika evenemang. Detta syns på figur nr 13 nedan.

Figur 13. Lägg till alumnimedlem

Koden i de två olika medlemsregistren är i stort sett samma men med små skillnader. I figuren ovanför ser man att det finns en hel del fler fält för alumnimedlemmar jämfört med ordinarie medlemmar. Vi har dock i detta register lagt till möjlighet att redigera medlemmar. Detta på grund av att det är så mycket information som skall matas in, att det lättare sker misstag.

Alumnimedlemmar skall även meddela åt föreningen om de byter adress eller dylikt så att vi kan hålla våra register uppdaterade. Formulären som används skiljer sig endast med att man är tvungen att skriva in mer information om de medlemmar man lägger till i alumniregistret.

Systemet kommer självklart att automatiskt räkna ut en summa varje år som kommit från nya alumnimedlemmar och lägga till det i rätt konto så att det kommer med i bokslutet.

5.5 Bokföring

Bokföringsdelen av systemet består i stort sett av fyra olika komponenter:

- Bokföringshändelser
- Evenemang
- Kontoplan
- Bokslut

Bokföringshändelser står för lagringen av de olika utgifter och inkomster som föreningen har. Detta sker med hjälp av de olika kontona från kontoplanen. Evenemang är en del som följer upp de olika evenemang som föreningen ordnat och härifrån får man fram vilka inkomster och utgifter ett evenemang orsakat. Bokslutet räknar automatiskt ut ett års inkomster och utgifter för att sedan lägga upp resultaten i läsbart format.

I detta kapitel tar vi närmare upp utvecklingen av alla dessa olika komponenter samt en del problem som uppstått i utvecklingsskedet, och vilka lösningar vi använt för att lösa problemen.

Bokföringssystemet skall kunna följa upp händelser under en räkenskapsperiod och sedan upprätta ett bokslut för det året. Man skall även kunna bläddra igenom evenemang och se vilka händelser som tillhör de specifika evenemangen.

Då vi började utveckla bokföringen startade vi från det lättaste och jobbade oss upp till mera invecklade saker. Detta ledde till att vi långsamt fick mer insikt om hur saker skall göras och vi blev mer bekanta med kodningen och de lösningar vi sedan använde.

5.5.1 Kontoplan

För att få en god början på grunden till bokföringen och för att det vart det lättaste, bestämde vi oss för att börja med kontoplanen, då det lägger grundläggande funktionalitet för att kunna sätta in bokföringshändelser.

Implementeringen av kontoplanen var relativt simpel och fungerar i stort sett som tidigare funktioner. Vi började som tidigare med att bygga upp ett formulär för allt som behövs när man matar in ett nytt konto.

Fälten för konton blev ursprungligen:

- Kontonummer
- Kontonamn
- Kontotyp

När vi senare började utveckla bokslutet konstaterade vi att ett fält till måste införas för de olika kontona. Detta på grund av att det blev vissa problem med att ställa upp bokslutet på det sätt vi önskade, utan ett extra fält för kontogrupp. Själva implementeringen av detta fält gick smidigt undan och gav önskat resultat.

När vi hade lagt till alla konton så utvecklade vi funktioner för att redigera existerande konton. Denna funktion fungerar på så sätt att man kan ändra allt utom kontonummer. Vill man ändra kontonumret måste man lägga in kontot på nytt.

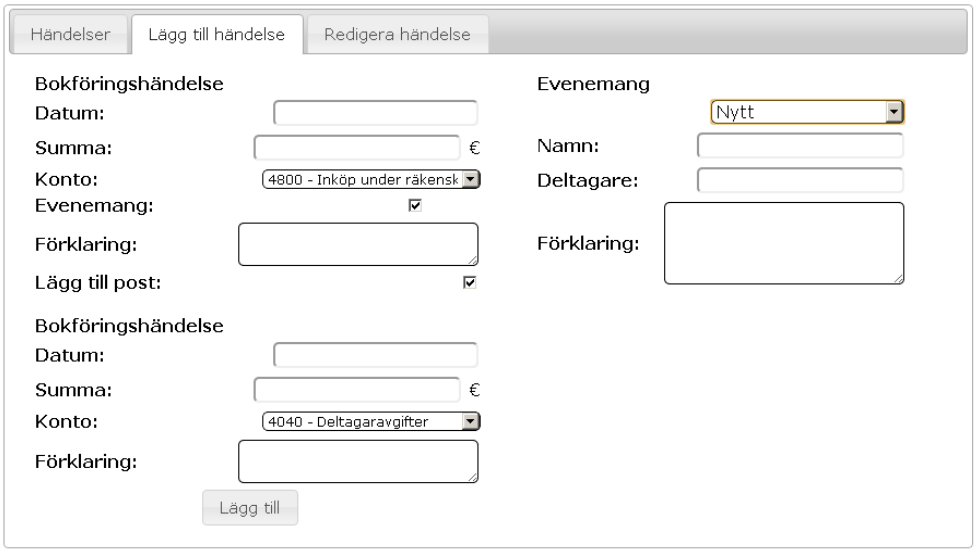
När delen för kontoplanen var färdig kunde vi börja gå vidare och utveckla själva bokföringshändelser.

5.5.2 Bokföringshändelser

En bokföringshändelse är en transaktion där föreningens pengar kommer in eller går ut. Eftersom bokföringen för föreningar är relativt enkel, så behövs endast ett inlägg för en händelse och inte två inlägg för händelser som det behövs i vissa andra metoder för bokföring. Detta leder till att implementeringen av bokföringshändelserna var enkel, men inte helt problemfri.

Problem vi stötte på orsakades av att vi ville koppla evenemang till samma formulär som själva händelserna, så man enkelt kan fylla i information om evenemang då man för in transaktioner.

Största problemet var att få formuläret för evenemanget att inte synas förrän man markerat en checkbox. Eftersom vi inte använt så mycket JavaScript tidigare, var detta lite problematiskt och vi måste läsa in oss på området och försöka oss på några olika lösningar, innan vi hittade den rätta. Resultatet visas i figuren nr 14 nedanför.



Figur 14. Inmatning av bokföringshändelser

När vi väl fått det att fungera, bestämde vi oss för att lägga till möjligheten att föra in två händelser på en gång, om man så önskar, eftersom man ofta gör händelserna för ett evenemang på samma gång och man på så vis smidigt får in utgifter och intäkter för evenemanget. Även detta framgår ur föregående figur nr 14.

Ett annat problem som uppenbarade sig var att vi hade svårigheter med att få listan på konton in i formuläret för händelserna. Detta på grund av att vi inte riktigt hade optimala klasser för att utföra det vi vill utföra på rätt sätt. Felet uppstod då vi försökte plocka in en lista för konton in till bokföringshändelserna och detta inte fungerade då vi på något sätt måste plocka in klassen som innehöll funktionaliteten för kontona in till klassen med bokföringshändelserna. Skulle vi varit mera bekanta med objektorienterade metoden skulle vi helt kunna undgå detta problem men då det var relativt nytt blev detta orsakade problem lite problematiskt.

Problemet blev efter en tid löst då vi blev tvungna att skriva om en hel del kod och skapa en klass som innehåller alla formulär och listor över saker som krävs i bokföringen. När vi väl fått denna klass att fungera kunde vi se till att klassen för händelserna ärver dess egenskaper och på så vis kommer vi åt listorna över konton utan vidare problem och kom således vidare med utvecklingen.

Vid ett senare skede av testningen upptäckte vi även ett fel som garanterat skulle orsaka problem senare: inmatning av decimaltal skedde med punkter istället för kommatecken. Det är visserligen välkänt för de flesta som har erfarenhet av programmering att det typiskt brukar vara punkt som skiljer åt decimalen men då vi inte har programmerare som användare måste vi åtgärda detta. Det orsakade lite problem i början men efter en tid kom vi fram till vår lösning som fungerade.

Kod 7. Ersättning av kommatecken

```
$amount = str_replace(',', '.', $_POST['amount']);
$amount = round($amount,2);
```

Som koden nr 7 ovan visar så var det relativt enkelt. Det enda man, i princip, behövde göra var att köra PHPs inbyggda ersättnings funktion *str_replace()* för att ersätta kommatecknet med en punkt. Sedan för att försäkra oss om att variabeln är ett tal så valde vi ännu att avrunda det till två decimaler då det inte behövs flera.

5.5.3 Evenemang

Själva evenemangen är ganska tätt kopplat till händelserna. Nya evenemang skapas endast då man fyller i nya händelser, vilket sker i samband med ett evenemang. Man kan också lägga till händelser till existerande evenemang om det är så att man har missat något, eller det senare har kommit till någon utgift eller intäkt som är relaterat till evenemanget.

Alla evenemang läggs upp i en lista som visar alla evenemang som förekommit under detta läsår. I denna lista kommer det inte upp annan information än namnet, deltagare och läsåret för de olika evenemangen.

Genom att trycka på ett evenemang i listan så får man upp mer information. Som figuren nr 15 nedan visar så ser man här vilka konton som är inblandade i just detta evenemang, hur mycket pengar det är fråga om, samt en mer genomgående förklaring på evenemanget ifall man undrar över något om evenemanget. Vi strävar efter att det alltid skrivs en förklaring till varje evenemang eller händelse. Detta underlättar frågor man kan stöta på i framtiden om man funderar över någon summa i bokslutet. Det finns självklart även möjlighet att redigera ett evenemang.

Figur 15 Fakta om ett evenemang

Det är viktigt att noggrant fylla i evenemang då det i framtiden betydligt underlättar arbetet för planering av kommande evenemang. När man äntligen har tillgång till noggrann information om utgifter, inkomster samt deltagarantal från tidigare års evenemang.

5.5.4 Bokslut

Då den övriga funktionaliteten för bokföringshändelser, konton och evenemang var färdiga, så kunde vi börja lägga upp bokslutet som sista del av hela bokföringssystemet. Bokslutet byggdes utseendemässigt upp likadant som i excel-tabellerna som tidigare använts.

Då de olika posterna skulle räknas ut, hade vi lite svårt i början. Med hjälp av lite undersökningsarbete kom vi fram till att vi med MySQL-förfrågningar kan göra en ganska stor del av jobbet. För varje konto sökte vi upp händelser från året och tog en summa av dessa händelser för att skriva ut posten på bokslutet. För att underlätta kodande så gjorde vi en funktion för att utföra detta smidigare i själva huvudsakliga den funktionen för bokslutet.

Kod 8. Funktionen account_sum($account,$year)

```
public function account_sum($account, $year) {

$query = "SELECT SUM(Summa) AS Total FROM Handelser
WHERE Konto_nummer = '$account' AND YEAR(Datum) = '$year'";

if($result = $this->mysqli->query($query)) {
    while($row = $result->fetch_assoc()) {
        $total = round($row['Total'], 2);
    }
}
$result->free_result();

return $total;

}
```

Som vi ser i koden nr 8 på föregående sida, tar denna funktion in parametrar för ett konto och ett år. Som bilden visar så utför vi med hjälp av MySQL-förfrågningen, som definieras i variableln *$query*, det mesta av jobbet. Förfrågningen räknar automatiskt ut totala summan från bokföringshändelserna där kontonumret och året är det som matats in. Som tidigare poängterats så lägger denna funktion grunden för bokslutet och gör det viktigaste jobbet.

För varje kontogrupp utför systemet en förfrågning som kör igenom alla konton i den aktuella gruppen. Inom denna förfrågning körs sedan vår tidigare nämnda funktion *account_sum($account, $year)* för att räkna ut summan för kontots händelser på det aktuella året. Summan plockas sedan in i en matris. Detta för att räkna ut summan av kontogruppen lite senare. Som vi ser i kod nr 9 nedan så visas allt detta ganska klart. Här visas även att med hjälp av en if-sats så kontrollerar systemet även upp att det finns transaktioner på kontot före det skriver ut resultatet. När både while-slingan och if-satsen körts till slut räknas det ännu ut totala summan på kontogruppen ifråga med hjälp av den tidigare nämnda matrisen.

Kod 9. Uträkning av summor från kontogrupper

```
while($row = $result->fetch_assoc()) {

    $ordinarie_ut[$row['Konto_nummer']] = $this->account_sum($row['Konto_nummer'],$year);
    if($ordinarie_ut[$row['Konto_nummer']]) {
        echo "<tr><td>".$row['Konto_nummer']." - ".$row['Konto_namn']." </td><td></td>
        <td>-".number_format($ordinarie_ut[$row['Konto_nummer']], 2, ",", " ")."e</td></tr>";
    }
}
$ord_ut_total = array_sum($ordinarie_ut);
```

Dessa förfrågningar och uträkningar körs så att bokslutet läggs upp i rätt ordning. I figur nr 16 på nästa sida kan vi se bokslutets uppställning och hur de olika kontogrupperna indelas. Denna uppställning är ganska liknande den som använts på excel-kalkylbladen under tidigare år.

ORDINARIE VERKSAMHET		2010
ADMINISTRATION		
4080 - Övriga Intäkter		1 500,00e
4040 - Deltagaravgifter		9 385,00e
	Totalt:	10 885,00e
Övriga verksamhetskostnader		
4250 - Styrelsekostnader		-273,00e
4280 - Representationskostnader		-484,27e
4400 - Postkostnader		-52,10e
4450 - Kansli och nyckelkostnader		-37,00e
4500 - ADB-Kostnader		-63,00e
4530 - Bankkostnader		-83,47e
4720 - Fordonskostnader		-10,00e
4800 - Inköp under räkenskapsperioden		-12 721,82e
4850 - Inköp av halarmärken		-884,82e
	Totalt:	-14 609,48e
RESULTAT AV ORDNIARIE VERKSAMHET		-3 724,48e
TILLFÖRDA MEDEL		
Intäkter		
5000 - Medlemsavgifter		480,00
5010 - Halarintäkter		3 700,00
	Totalt:	4 180,00e
INVEST-O FINANSIERINGSVERKS.		
Kostnader		
5190 - Övriga finan. kostnader		-300,80e
	Totalt:	300,80
RESULTAT		455,52e

Figur 16. Bokslutet

När vi väl fått fram något som såg lovande ut så kom vi till en annan vägg nämligen: byte av räkenskapsperiod och uppgörande av bokslutet.

Vid uppgörande av bokslutet var det redan ganska långt bestämt vad som skall räknas ut automatiskt och vad som användaren skall mata in. Vi kom fram till att ränte- och dividendintäkter matas in endast här. Detta ledde till att vi måste begränsa användare från att lägga in händelser på detta konto och istället ha ett fält för det då man vill upprätta bokslutet.

Poster som automatiskt läggs in i detta skede var medlemsavgifter från både ordinarie medlemmar och alumnimedlemmar. Det här sker genom att automatiskt räkna ut antalet medlemmar under året och sedan multipliceras resultatet med tio euro vilket ger slutsumman för ordinarie medlemmarna. Efter detta adderas summan till allt som alumnimedlemmarna betalat in, och totala summan förs in som en händelse.

Om man däremot redan har matat in något på medlemskonto så räknas detta inte automatiskt, utan man måste då räkna manuellt och mata in önskat värde som en vanlig händelse, detta lades till för att kunna mata in tidigare års bokslut.

Uppgörandet av bokslutet medförde ett antal problem. Huvudsakligen för att vi hade använt fel syntax på några MySQL-förfrågningar, vilket ledde till att vi spenderade mycket tid på att hitta felet.

Största felet här var att när vi försökte få systemet att automatiskt byta räkenskapsperiod till året efter den aktuella perioden. Vi försökte lägga till ett år till den existerande variabeln. Men den variabeln vi tar in blir ett datum och när man försöker addera något till det så fungerar det inte riktigt som planerat. Vi fick resultat som 1970 och 2001, då önskade värdet var 2009. Slutligen efter mycket arbete och funderande så hittades en lösning som fungerade.

Kod 10. Uträkning av nästa år

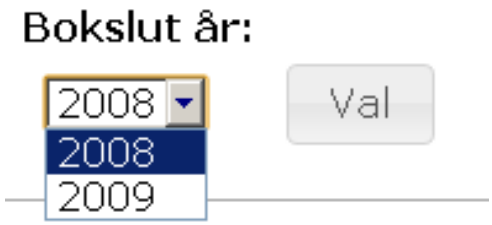
```
$next = $year;
$next = strval($next);
$next = intval($next);
$next++;
```

Som kod nr 10 ovan visar så konverterade vi först året vi tog in till en sträng och sedan den strängen till ett heltal och när variabeln väl var ett heltal var det bara att addera ett till talet och vi fick det önskade värdet. Efteråt låter det ju lätt men då man

höll på och försökte med många olika alternativ, vissa med betydligt mer kod för att fungera, så kändes det ganska tungt.

När vi kommit så långt att man kan upprätta ett bokslut, lade vi till funktionalitet för att ändra den aktiva räkenskapsperioden, samt för att slå upp bokslut från föregående år.

Äldre bokslut kan endast öppnas om deras status är stängd och boksluten är slutförda. Man kan inte heller byta till en räkenskapsperiod där statusen är stängd man måste alltså se till att man har allt rätt så man kommer vidare utan problem. När man slår upp tidigare års bokslut kan man, om så önskas skriva ut detta bokslut. Som figur nr 17 nedan visar så valde vi att utnyttja en dropdown-meny för att välja tidigare års bokslut. Om man sedan trycker på knappen så får man upp bokslutet för det valda året.



Figur 17. Tidigare års bokslut

Vid uppgörande av bokslut skapas det automatiskt en ny räkenskapsperiod för nästa år och byter sedan till den räkenskapsperiod som aktiv räkenskapsperiod. Denna automatisering syns på kod nr 11 på nästa sida.

Kod 11. Byte av räkenskapsperiod

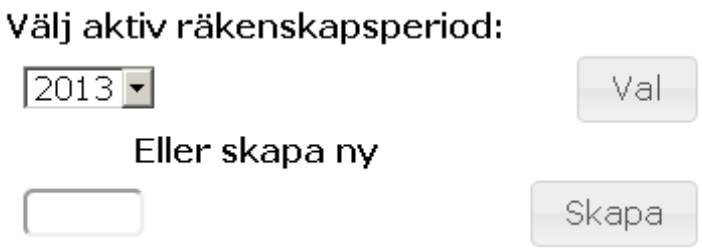
```
$query = "UPDATE Rakenskapsperiod SET Status = 'closed' WHERE Ar = '".$year."'";
$this->mysqli->query($query);

$query = "SELECT * FROM Rakenskapsperiod WHERE Ar = '".$next."'";
if(!$result = $this->mysqli->query($query)) {
    $result->fetch_assoc();
    $query = "INSERT INTO Rakenskapsperiod (Ar, Status) VALUES ('".$next."', 'active')";
    $this->mysqli->query($query);
}
else if($result = $this->mysqli->query($query)) {
    $result->fetch_assoc();
    $query = "UPDATE Rakenskapsperiod SET Status = 'active' WHERE Ar = '".$next."'";
    $this->mysqli->query($query);
}
$result->free_result;

$this->redirect();
```

Som kod nr 11 ovan visar så körs det ett antal olika MySQL-förfrågningar efter varandra då man byter räkenskapsperiod. Den första stänger den aktiva räkenskapsperioden, den andra försöker hitta en räkenskapsperiod med året efter den aktiva, och om det inte finns så skapas det en ny. Om det däremot färdigt finns en för nästa år så byter en förfrågning status på denna till aktiv.

Om det däremot inte har fungerat som det skall, har vi lagt till stöd för att manuellt kunna mata in en ny räkenskapsperiod. Denna funktion är även nyttig om man vill mata in andra års bokslut. I figur nr 18 nedanför så syns det klart hur denna del av bokslutet fungerar.



Figur 18. Byte eller skapande av räkenskapsperiod

Med denna sista funktion är bokslutet färdigt som vi hade planerat det och hela systemet är redo för användning för föreningen.

6. Utvecklingsmöjligheter

Detta arbete gjordes för Ekenäs Kåravdelning, en förening för studerande. Men konceptet kan lätt användas av andra liknande föreningar, eller skräddarsys för andra föreningar. I detta kapitel har vi räknat upp en hel del utvecklingsmöjligheter som skulle förbättra vår slutliga produkt. En hel del av dessa kunde ha implementerats men på grund av en ganska stram tidtabell hade vi inte möjlighet att göra precis allt som vi önskat.

6.1 Medlemsregister

Utvecklingsmöjligheter för medlemsregistret är begränsade. Men ett alternativ skulle vara att personer skulle kunna ansöka om att bli medlem på hemsidan. Hemsidan skickar personen vidare med hjälp av internetbanken till betalningen. Efter att personen betalat skulle namnet automatiskt komma upp i medlemsregistret. Man skulle inte behöva skriva in personen manuellt. Allt skulle skötas automatiskt. Här är det dock många säkerhetsaspekter gällande nätbetalningar som måste tas i beaktande.

Ett annat alternativ är att överföra alla händelser till nätet. Alla anmäler sig till evenemang på nätet. Varje person har ett eget användarkonto, här kommer det upp om personen har betalat medlemsavgift eller inte. Evenemangen betalas även på nätet, vilket betyder att ingen kassa behövs för evenemangen. Detta minskar arbetet för styrelsen och man behöver inte handskas med så mycket kontanter.

I alumniregistret har vi lagt till en balk med namnet ”skicka e-post”. Den fungerar tyvärr inte ännu i detta skede, så det kommer vara att en utvecklingsmöjlighet. Meningen med detta är att lätt kunna skicka e-post åt alla i registret. Det underlättar all kontakt med medlemmarna efter att de har blivit utexaminerade från Yrkeshögskolan Novia, Raseborg. I nuläget har vi endast lagt till detta i alumniregistret, men inget hindrar att man vidareutvecklar det till att användas också i det vanliga medlemsregistret.

6.2 Bokföring

I bokföringsdelen finns det ett antal olika möjligheter för att förbättra funktionalitet och användbarhet.

Kontoplanen kan förenklas ganska långt genom att använda sig av mer uttänkta kontonummer, vilket skulle leda till att det helt skulle gå att ta bort kontogrupperna från användningen. För tillfället är det ändå lättare och klarare att man väljer en grupp för varje konto.

Bokföringshändelser fungerar för tillfället rätt bra. En stor förbättring skulle vara att kunde välja hur många händelser man vill föra in på en gång. På så sätt kunde man underlätta arbetet, då man inte lagt in några poster på ett tag och man vill mata in allt på en gång.

Evenemangsdelen skulle kunna förbättras ganska grovt genom att utveckla den och noggrannare planera vad man vill ha ut av denna funktion. Man skulle kunna göra evenemang till en större, skiljd del, där man sedan kunde göra föra in händelser på detaljnivå och spara mer information angående evenemangen. Exempelvis vilka medlemmar som har deltagit i evenemanget samt mer utförligt vilka utgifter och intäkter evenemanget haft. Alla dessa funktioner skulle hjälpa styrelsen att planera budgeter till evenemang i framtiden och man skulle ha lättare att beräkna om hur många som kommer att delta i ett visst evenemang.

Till bokslutet skulle man kunna införa någon funktion för att jämföra vissa års bokslut sida vid sida för att enkelt få fram om det gått bättre eller sämre under det gångna året.

6.3 Login

Login funktionen skulle kunna förbättras ganska drastiskt då den blev relativt tillfällig när vi började projektet. Detta för att vi körde fast på att vidareföringen inte fungerade, vilket ledde till att så fort vi fick inloggningen att fungera, så lämnade vi den orörd. I det skedet ville vi inte röra till det som fungerade.

Man skulle kunna göra ett mer utförligt användarsystem med användarnivåer för att begränsa vad vissa användare tillåts och inte tillåts göra. Då kunde man få stöd för flera användare istället för bara en. Så som det är nu. Här skulle man även kunna lägga in något sorts system för att hålla reda på vilken användare som senast har redigerat eller lagt till någon post. På så sätt vet man alltid vem som har gjort vad och när.

6.4 Övrigt

Som tidigare nämnt har designen inte varit vår högsta prioritet. Detta har lett till att en stor förbättring på arbetet skulle vara att jobba vidare och finslipa designen. Det finns hundratals små justeringar som skulle kunna göras för att få produkten att se mer attraktiv ut. Designen skulle även kunna göras mer anpassbar, så att man själv fick mata in färgkoder, eller ladda upp sin egen logo, för att ge användaren mer möjlighet att ändra på utseende om något särskilt stör ögat.

Databasstrukturen skulle kunna förbättras en hel del eftersom vi inte varit så insatta som vi önskat. Här skulle man till exempel kunna förbättra medlemsregistret och ta bort en hel del fält för att optimera det ytterligare. Även konto-delen av databasen kunde förbättras ganska grovt genom att få ut en kontogrupp på basis av kontonummer istället för att ha skilda fält för kontogrupperna.

Typiskt så skulle man använda sig av heltal i program som behandlar pengar. Detta för att undvika problem med avrundningar och dylikt. Vi har dock använt oss av decimaltal vilket inte direkt är att rekommendera, men det fungerar. Detta underlättar för den ekonomiansvarige, som nu vet att talen alltid skall anges med två decimaler överallt. För att använda sig av heltal måste man dock sätta ut heltal utan decimaler och detta kan vara knepigt för vissa användare.

En utvecklingsmöjlighet skulle vara att göra en faktureringsmodul för att underlätta arbetet då man gör räkningar för diverse saker och underlätta arbetet ganska mycket då man inte behöver fundera var man har excel-modellen för fakturor.

En annan stor förbättring skulle vara att införa PDF-formatet då man skriver ut. Detta skulle innebära att man lätt skulle få filen nerladdad och skickad per e-post eller utskriven, istället för att skriva ut en HTML-fil som det nu är.

Om man ser på layouten så har vi även lagt in ”inställningar”. Tanken var att lägga till alternativ för att välja hur mycket information man vill att krävs i de olika formulären, samt hur mycket information som skall visas i de olika listorna. Tyvärr hann vi inte implementera detta. Detta skulle ha gett friare händer åt administratörerna på sidan så att man skulle kunna ha tillgång att ändra de olika formulären som finns på sidan.

I dagens läge då det är stor efterfrågan på mobila lösningar är i stor efterfrågan så skullen en stor utvecklingsmöjlighet vara att få detta system att fungera för olika mobila lösningar för att få så bred användning av systemet som möjligt. Detta skulle vara relativt tidskrävande men skulle vara en stor fördel för systemet som då skulle kunna användas när och var som helst.

Någon färdig möjlighet att få ut säkerhetskopior från databasen finns inte för tillfället och detta borde läggas till. Så som det är nu hamnar man själv gå in i MySQL-databasen och ta en dump av databasen för att se till att man har en säkerhetskopia av den. En möjlighet att enkelt få en säkerhetskopia från systemet kunde till exempel, läggas till i delen för inställningar men som det nu var fanns det helt enkelt inte tillräckligt tid för en sådan implementering.

En möjlighet att ladda upp information i olika former skulle också vara en utmärkt möjlighet till förbättring. Här kunde man då lätt få in till exempel tidigare års medlemmar och dylik information.

Största utvecklingsmöjligheten är dock att få hela systemet betydligt mera anpassbart, vilket skulle ge en kommersiell produkt, som olika slags föreningar skulle kunnadra nytta av och kunde tänkas betala en liten summa per år för att använda.

7. Slutsatser

I detta kapitel drar vi samman tankar, som upstått vid utvecklingen, skrivningen samt finslipningen av arbetet och produkten. Vi tar upp noggrannare resultat, lärdomar och avrundar sedan med slutsatser.

7.1 Resultat

Slutprodukten av systemet uppfyller de mål vi satte när vi planerade. Alltid finns det delar som skulle kunna göras bättre, men resultatet blev en användbar produkt som kommer att hjälpa föreningen i de ekonomiska frågorna. Frågor som ofta tidigare förorsakat problem, bland annat bokslutet. De som har betalat medlemsavgiften kommer nu dessutom kunna garanteras de privilegier som de har rätt till. Alla får ta del av det som de har lovats dem.

Som resultat har Ekenäs kåravdelning fått, som utlovats, ett användbart verktyg för att följa upp föreningens medlemmar, spara information om föreningens bokföring och utföra enhetliga bokslut år efter år. Detta kommer att, som tidigare nämnts, underlätta arbetet för föreningens styrelse i stor grad, då allt är enhetligt och lagrat på samma plats. Det finns nu en kontinuitet i skötseln av både ekonomin.

Denna produkt har även potential för att vidareutvecklas och på så sätt få till stånd en kommersiell produkt. Genom att skapa olika skräddarsydda versioner av denna produkt har man möjlighet att ge ett utmärkt verktyg för bokföring samt upprätthållande av medlemsregister åt en mångfald olika sorters föreningar av olika storlekar.

7.2 Lärdomar

Vi har samlat på oss en hel del lärdomar under arbetets gång.

Det som framkom allra tydligast var hur viktig planeringen är vid dylika projekt. Även om man tycker att man planerat väl, så kommer det alltid något som är oförutsägbart och som leder till förändringar i planerna.

Vad vi har kommit fram till under utvecklingen av systemet är att man bör sträva efter att ha olika delsystem såpass isolerade att man lätt kan plocka ut dem och använda dem i framtida projekt. Detta underlättar arbetet då man gör något som kräver liknande lösningar man gjort i tidigare projekt. Det sparar således mycket tid och pengar för både kunden och utvecklaren. Dessa delsystem bör även planeras så klart som möjligt för att få ut så mycket som möjligt av dem och få dessa så anpassningsbara som möjligt.

När man utvecklar något med verktyg man är rätt så bekant med blir det lätt så att man går miste om de många förbättringar, som skulle kunna användas om man noggrannare hade läst in sig på områden, som man inte tidigare har haft nytta av.

Tidsplaneringen håller för det mesta inte. Lägg alltid upp tidsplanen lite i överkant så det inte blir onödigt bråttom i slutfasen av arbetet. Saker är ofta mer invecklade än vad de verkar vara på pappret.

När man arbetar med en produkt och skriver slutarbete om denna är det mycket viktigt att kontinuerligt dokumentera under arbetets gång. Det är så gott som omöjligt att, i efterhand konstruera produktiv text när man inte har arbetat med något på flera veckor. När en sak börjar fungerar, glöm inte dokumentationen.

7.3 Sammanfattning

Arbetet gick snabbt framåt i början. Det tog inte många veckor innan vi hade en fungerande produkt som gjorde allt utom bokslutet. Vi kunde säkert ha gjort många delar av bokföringen mer optimerade. Vi undersökte inte särskilt mycket vilka andra alternativ man skulle kunna använt istället för de lösningar som vi har implementerat. Vi satte oss ner och diskuterade hur vi vill att de olika delarna av systemet skulle fungera och hurdant slutresultat vi ville ha. På basis av den information vi kom fram till, utvecklade vi systemet. Vi ville utveckla ett system, som var fullständigt enligt vår egen modell, för att få en produkt som verkligen känns som om den är vår egen.

Vi hade ingen speciell beställare, som ville ha något på ett speciellt sätt. Detta ledde till att vi hade relativt fria händer medan vi planerade systemet. Vi byggde systemet på basen av våra egna erfarenheter av bokföringen för föreningen och frågade nu och då efter feedback av föreningens ordförande. Vi är själva nöjda över slutresultatet och ordförande var också imponerad av resultatet vi åstadkom.

Det finns, som tidigare sagts, många saker som skulle kunna vidareutvecklas, och många saker som kunde göras bättre eller mera anpassningsbart. Detta kommer möjligtvis att hända i ett senare skede. Men så som systemet nu ser ut, så kommer det att fylla sin uppgift, det vill säga att hjälpa föreningens ekonomiansvariga med bokföringen och medlemsuppföljningen.

Källförteckning

Ducket J. (2011) *Beginning HTML, XHTML, CSS, and JavaScript,* ***Wrox***

Skansholm J. (2011) *C++ direkt,* ***Studentlitteratur AB, Lund***

Valade J. (2009) *PHP & MySQL for Dummies, 4th Edition,* ***Wiley***