TEKNISKA HÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

UTVECKLING AV ETT WEBBASERAT EKONOMISYSTEM FÖR EN LITEN ORGANISATION

David Luotonen

EXAMENSARBETE 2009
DATATEKNIK

TEKNISKA HÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

UTVECKLING AV ETT WEBBASERAT EKONOMISYSTEM FÖR EN LITEN ORGANISATION

Development of a web based economic system targeted at a small organization

David Luotonen

Detta examensarbete är utfört vid Tekniska Högskolan i Jönköping inom ämnesområdet datateknik. Arbetet är ett led i den treåriga högskoleingenjörsutbildningen. Författaren svarar själv för framförda åsikter, slutsatser och resultat.

Handledare: Magnus Schoultz

Omfattning: 15 poäng

Datum: 4/6 2009

Arkiveringsnummer:

Postadress: Box 1026 551 11 Jönköping

Besöksadress: Gjuterigatan 5

Telefon: 036-10 10 00 (vx)

Abstract

Jönköpings styrkelyftarklubb is a small organization in need of a quicker way of handling their book-keeping.

The purpose of this thesis is to investigate the development of an Internet based economic program to be used by selected individuals within a small organization, and to assess what methods and tools are suited for this task considering the organizations demands.

The OOS/UML development process model is used, dividing the process into 4 stages. Only the first 3 stages are considered in this report: planning, prototype development and construction.

The theoretical framework touches on several subjects relevant for this project. The first part contains information about open source and ends in a description of the selected scripting language: PHP. The second part focuses on databases, and develops from general design issues toward the use of MySQL and security issues. The third part continues on the subject of security, treating not only database security but also encryption on web pages. The fourth part is about economic systems and their development and use. The chapter finishes in a brief look at AJAX as a means for increased usability.

The work led to some general conclusions regarding the development of a system like this. Planning is important since time consumption will be hard to foresee and the system will not be able to fully replace accounting systems or web shops. Using existing solutions will save time and sometimes it is worth changing the design if it will help. Finally prioritizing features is a must, since a combination of systems such as this will become too large and complex otherwise. Performance, usability, cost and security are aspects competing over the same resources. In this project, performance and cost had to be prioritized.

Subject terms

Cash register, PHP, web programming, MySQL, AJAX, book-keeping, accounting.

Sammanfattning

Jönköpings styrkelyftarklubb är en liten organisation som behöver ett snabbare sätt att hantera sin bokföring.

Syftet med detta arbete är att undersöka utvecklingen av ett Internetbaserat ekonomiprogram för användning av utvalda i en liten organisation samt utreda vilka metoder och verktyg som är lämpligast för uppgiften utifrån de krav som finns.

OOS/UML-modellen som delar in utvecklingsprocessen i 4 steg används. Bara de 3 första stegen behandlas i denna rapport: planering, prototyputveckling och konstruktion. Det teoretiska ramverket berör flera relevanta ämnen för detta projekt. Den första delen innehåller information om öppen källkod och avslutas med en beskrivning av det valda skriptspråket: PHP. Den andra delen fokuserar på databaser, och går från generella designteorier till användningen av MySQL och säkerhetsproblem. Den tredje delen fortsätter på området säkerhet, och behandlar förutom säkerhet i databaser även kryptering på webben. Den fjärde delen beskriver ekonomiska system och deras utveckling och användning. Kapitlet avslutas med en kort titt på AJAX som ett medel för ökad användbarhet.

Arbetet ledde till några allmänna slutsatser om utvecklingen av ett system som detta. Planering är viktigt eftersom tidsåtgång är svårt att förutse och systemet inte kommer kunna ersätta vare sig ekonomiprogram eller webbutiker till fullo. Att använda existerande lösningar på programmeringsproblem sparar tid, och ibland är det värt att ändra i designen för att kunna använda en sådan. Avslutningsvis bör man prioritera egenskaper, eftersom en kombination av system som detta annars kan bli allt för stort och komplext. Prestanda, användbarhet, kostnad och säkerhet är aspekter som alla tävlar om samma resurser. I detta projekt visade prestanda och kostnad sig vara viktigast och fick prioriteras.

Nyckelord

Kassaapparat, PHP, webbprogrammering, MySQL, AJAX, bokföring.

Innehållsförteckning

1 Inledning

Går det skapa ett ekonomisystem som är så enkelt att vanliga medlemmar i en styrkelyftarklubb med varierande datorvana kan använda det, men som samtidigt klarar av normal bokföring?

I dag är konkurrensen bland de företagsinriktade ekonomiprogrammen hård, men de är trots det inte helt enkla att använda. Fokus ligger på helt andra aspekter (t.ex. skatt, kontoplaner och bokslut) än de som en ideell förening värderar högst. Tillgänglighet genom ett Internetbaserat gränssnitt, enkelhet och låga implementations- och driftkostnader är istället högt prioriterade.

Denna kandidatuppsats är en del av en treårig högskoleingenjörsutbildning i datalogi vid Jönköpings tekniska högskola. Den försöker utreda möjligheterna att använda verktyg och bibliotek med öppen källkod till att lösa ovanstående problem. Detta angreppssätt har valts både med utgångspunkt i kostnader för mjukvara och servrar, och ett intresse för sådan mjukvara från min sida.

Är det då genomförbart att skapa ett internetbaserat ekonomiprogram som har tillräcklig säkerhet och är tillräckligt snabbt? Svaren ligger troligtvis lite i hur stor organisationen är, och därmed hur utsatt den är för angrepp. Hur säkerheten i systemet hanteras på ett effektivt sätt beskrivs utförligt i teoridel samt genomförande.

Snabbheten är en annan viktig faktor, eftersom programmet kommer användas som ”kassaapparat”. Alla Internetbaserade program lider av problemet med att anslutningen begränsar hastigheten, men hur detta problem kan undvikas diskuteras i analys och slutsats.

1.1 Bakgrund

Jönköpings styrkelyftarklubb är en relativt stor styrkelyftarklubb jämfört med andra i Sverige, men ändå liten i jämförelse med andra idrottsklubbar. Exakt medlemsantal är svårkontrollerat, men det finns ett 30-tal tävlingsaktiva och till det hundratalet motionstränande.

Jönköpings SK sköter i dag löpande bokföring av medlemsavgifter m.m. manuellt i pärm. Detta gör bokslut och betalningshantering komplicerat och tidskrävande. Det är dessutom omständigt att kontrollera om medlemsavgifter betalats av medlemmar direkt efter att den föregående perioden tagit slut, eller om ett glapp uppstått då medlemmen av misstag tränat gratis. Eftersom klubben inte är vinstdrivande utan helt ideell, finns inte pengar eller kapacitet att inskaffa ett kommersiellt ekonomisystem. Ett enkelt datorbaserat ekonomisystem som går att nå från alla Internetanslutna datorer skulle kunna förenkla mycket, men säkerheten måste förstås vara god och användarvänligheten tillräckligt hög för att personer utan en ekonomisk utbildning ska kunna hantera det.

1.2 Syfte och mål

1.2.1 Syfte

Syftet med detta arbete är att undersöka utvecklingen av ett Internetbaserat ekonomiprogram för användning av utvalda i en liten organisation samt utreda vilka metoder och verktyg som är lämpligast för uppgiften utifrån de krav som finns.

1.2.2 Akademiska mål

Akademiska problemformuleringar inriktade på att skapa ny kunskap blir följande:

1. Hur uppnås tillräcklig prestanda, säkerhet och användbarhet för ett system som detta (som kombinerar traditionella ekonomiprogram och webbapplikation på ett nytt sätt)?

2. Vad i utvecklingsprocessen skapar svårast problem och var är tidsåtgången som störst?

3. Varför är de valda verktygen och metoderna att föredra framför alternativen i det här fallet?

Frågorna svarar på frågorna *hur*, *vad* och *varför*, vilket ger en bättre bredd av kunskap genom att ge både beskrivande och förklarande svar än om alla frågor varit formulerade med samma inledning. De är samtidigt formulerade som öppna frågor, och bidrar på så sätt till ett utforskande arbetssätt i studien. Utforskande tillvägagångssätt leder till att en djupare kunskap för vad problemen verkligen består av skapas, och en bättre förståelse för svårigheterna uppnås (Jacobsen, 2002).

1.2.3 Organisationens mål

Målen för systemet, framtagna och diskuterade med Björn Stolphammar i Jönköpings styrkelyftarklubb, blir följande:

1. Systemet ska vara användarvänligt (personer utan ekonomisk utbildning ska kunna hantera det) och snabbt (man ska kunna bokföra betalningar när de inträffar, som en kassaapparat).

2. Det ska finnas stöd för att betala medlemsavgifter med utgångspunkt från olika priskategorier, så som studerande, tävlande, arbetslös osv.

3. Systemet ska kunna sammanställa inbetalningar och utgifter på månads- och årsbasis.

4. Ett medlemsregister ska finnas, med grundläggande uppgifter om medlemmarna.

5. Genom inloggning ska administratörer kunna administrera medlemmar och försäljning, se rapporter osv.

6. Man ska kunna se hur mycket pengar varje medlem erhållit i sponsring från klubben.

7. Man ska kunna skapa och ladda ner backup-kopior av databasen från webbgränssnittet.

8. Säkerhetsåtgärder som kryptering av medlemmars lösenord måste vidtas.

Eftersom målen innebär att en stor bredd av funktionalitet krävs, kommer avgränsningar vara nödvändiga för att uppnå de viktigaste målen fullt ut.

1.3 Avgränsningar

Estetiken av systemet kommer inte läggas något fokus på. Webbplatsen ska vara funktionell och användarvänlig, men utseendet i övrigt ägnas ingen uppmärksamhet. Systemet kommer bara omfatta ekonomi, och ingen tid läggs på inloggning för medlemmar utan administratörsprivilegier. Användarvänligheten sträcker sig bara till att personer utan ekonomisk utbildning kan hantera det.

Införande av systemet och efterföljande justeringar kommer inte ägnas någon uppmärksamhet i denna rapport, fokus ligger istället på utvecklingen i linje med syftet. Kompabilitet med olika webbläsare läggs inte heller någon större vikt på, eftersom applikationen inte behöver fungera på så många datorer. Applikationen behöver bara fungera i de vanligaste webbläsarna med versioner som kommit de senaste åren. Testning utförs enbart i Mozilla Firefox och Microsoft Internet Explorer.

1.4 Disposition

1. Introduktion. Inriktning och problem beskrivs i början, och följs sedan av syfte och mål. Begrepp och förklaringar som kan vara bra att ha med sig i resten av uppsatsen läggs fram kortfattat.

2. Metod. Detta kapitel beskriver hur undersöknings- och arbetsprocessen gick till, och hur valen av verktyg och bibliotek som använts gjorts.

3. Teoretisk bakgrund. Vetenskapliga artiklar och andra källor utgör grunden för en teoretisk genomgång av de viktigaste områdena som berörts under utvecklingsarbetet.

4. Genomförande. Beskriver utvecklingen av projektet, från design till uppnådda mål. Val av komponenter och planering beskrivs i kapitel 2) Metod.

5. Analys. Knyter samman den teoretiska bakgrunden med det tillvägagångssätt som använts. Diskuterar även svagheter och styrkor.

6. Slutsats. Beskriver hur väl syftet och målen har uppfyllts, och ger förslag på eventuell framtida vidareutveckling av arbetet.

1.5 Begrepp

Medlemsavgift och träningsavgift används omväxlande i rapporten, då dessa är samma sak i Jönköpings styrkelyftarklubb. För att få träna måste man vara medlem.

Plattformar och operativsystem avser samma sak och används således omväxlande i rapporten. Ett operativsystem är ett grundläggande program som sköter datorns hårdvara och tilldelar resurser till övriga program. Vanliga operativsystem är Microsoft Windows, Linux, och MacOS.

SSL står för secure sockets layer, och är en teknik för att överföra information mellan två datorer utan att utomstående kan avlyssna den.

Webbsidan, webbapplikationen och ekonomisystemet syftar alla på samma sak: ekonomiprogrammet som håller på att utvecklas.

2 Metod

I detta kapitel beskrivs planeringen och de olika utvecklingsfaserna. Även kriterier för de olika valen som gjorts tas upp.

2.1 Utvecklingsfaser och projektplanering

Kriterierna för den färdiga sidan omfattar saker som användarvänlighet, enkelhet och kostnadseffektivitet. Tidigt i projektet drogs en ganska omfattande tidsplan upp, som är återgiven i förenklad form här:

- Förstudie där befintliga lösningar undersökts och kommenteras (1/1 2009 – 20/1 2009).
- Utveckling av prototyp med grundläggande funktionalitet (25/1 2009 – 1/3 2009).
- Informationssökning och litteraturstudier, början på rapportskrivning (1/2 2009 – 1/3 2009).
- Eventuell vidareutveckling av prototyp, alternativt nyutveckling av slutgiltig lösning (1/3 2009 – 1/5 2009).
- Färdigställande av rapport genom empiri, analys och slutsats (1/3 2009 – 1/5 2009).

Förstudien tittade på ett stort antal lösningar och verktyg, varav de viktigaste finns beskrivna under *4.2 Val av komponenter och verktyg*. Prototyputvecklingen skedde som ett steg i inlärningsprocessen och utgjorde sedan grund för den färdiga webbplatsen. Informationssökningen och litteraturstudierna påbörjades tidigare än planerat, för att kunna ha en god grund att stå på även vid prototyputvecklingen. Rapportens färdigställande bestod av tillägg till och ändringar av befintliga kapitel samt tillägg av analys och slutsats. Empiri-kapitlet döptes istället till genomförande, som i dokumentmallen, eftersom projektet hade mer ingenjörsmässig karaktär än teoretisk och undersökande.

2.2 Utvecklingsmodell

För att beskriva utvecklingsprocessen i detta projekt kan en modell kallad OOS/UML (object oriented system development / unified modeling language). Modellen beskrivs av Apelkrans och Åbom (2001) och går ut på att dela in processen i fyra faser. Den första fasen kallas ”tidig analys”, och innehåller verksamhetsanalys, lite UML-modellering och bedömning av de långsiktiga kraven på systemet. Fas nummer två heter ”prototyputveckling” och innebär en fortsatt analys av verksamheten, databasutveckling och gränssnittsdesign mm. I fas tre övergår man på konstruktion av den ”slutgiltiga” produkten, och det är då inte ovanligt att en vidareutveckling av prototypen används som bas. Den fjärde och sista fasen kallas ”införande och anpassning”, och går helt enkelt ut på att införa produkten i verksamheten, göra nödvändiga justeringar, skapa dokumentation och utbilda användare.

Istället för att följa modellen på pricka görs vissa mindre förändringar. Främst handlar det om mindre interaktion med användarna under utveckling och prototyputveckling, eftersom tid för omfattande tester och intervjuer inte finns. En av användarna väljs dock ut för att ge råd och föra kontinuerlig diskussion med om funktionalitet och kvalitet. Fas fyra kommer inte heller behandlas i detta arbete, då andra projekt fördröjer denna något. För övrigt används en något mer iterativ process än modellen antyder, eftersom kraven och vilka avgränsningar som behövs är mycket svåra att förutse i ett tidigt skede.

Andra populära utvecklingsmodeller är vattenfallsmodellen, Boehms spiral och SASD (Apelkrans och Åbom, 2001). Dessa modeller hade i anpassade varianter kunnat användas, men väljs bort eftersom de har svagheter så som avsaknad av planeringsfas, uppmuntran av ett icke iterativt arbetssätt (genom fasmål som måste uppfyllas för att gå vidare till nästa fas) och ett allt för stort fokus på informationsflöden snarare än processer i verksamheten.

2.3 Metodkritik

När man gör en kvalitativ studie snarare än kvantitativ bör man alltid tänka på att resultaten inte alltid kommer bli generaliserbara (Jacobsen, 2002). De slutsatser som dras är därmed inte nödvändigtvis applicerbara på andra projekt, även om de liknar detta.

En annan aspekt av metoden som bör kritiseras är att olika utvecklingsmodeller inte undersökts särskilt ingående. Den modell som valts kan mycket väl vara sämre än flera andra.

Vidare består de uppnådda resultaten till stor del av subjektiva bedömningar, då ingen studie av hur väl systemet fungerar på längre sikt finns med. Även de kortsiktiga utvärderingarna av systemets funktionalitet är bristfälliga, både genom att de baseras på subjektiva diskussioner och att mätbarheten är dålig. Det finns skillnader mellan fakta, åsikter och förklaringar, men vanligtvis hänger de samman. Det är därför viktigt att tänka på i vilken kontext informationen inhämtas och från vilken källa den kommer (Svenning, 2003). För att uppfylla dessa kriterier har diskussionerna förts med utgångspunkt i att verkligen hitta och kritisera brister med en person som känner sig bekväm med att kritisera.

Avslutningsvis kan det finnas problem med kvaliteten om milstolpar med hårda krav på funktionalitet finns, som i detta projekt. För att hinna uppfylla målen till ett visst datum finns det en uppenbar risk med att kod skrivs slarvigt och ostrukturerat för att hinna i tid.

3 Teoretisk bakgrund

Detta kapitel går översiktligt igenom de olika teoretiska områden som projektet berör. Denna information används sedan för att motivera val och förklara teknik i genomförandet. Kapitlet behöver inte läsas om inte en fullständig förståelse för projektet önskas.

Man kan se användbarhet, prestanda, kostnad och säkerhet som olika intressen som tävlar om samma resurser. Man får göra avvägningar i enskilda projekt om vad som ska prioriteras. Aspekter som bör beaktas är värdet på informationen som hanteras, budgeten för projektet, antalet förväntade besökare och eventuella brister som användarna kommer kunna acceptera (Welling och Thomson, 2005). Fokus ligger i denna rapport främst på kostnad och prestanda eftersom användarbasen kommer vara relativt begränsad, systemet måste vara snabbarbetat för att kunna fungera som kassaapparat och tid för övning i hur systemet fungerar finns.

3.1 Öppen källkod

Det finns ett stort antal fristående mjukvarukomponenter med olika funktionalitet på marknaden. Många är kommersiella och kostar pengar, men det finns även ett stort antal som både är gratis och har öppen källkod.

Förutom uppenbara fördelar för utvecklare med att förfoga över koden till en komponent själva och undvika höga kostnader för inköp har undersökningar (Lakhani och von Hippel, 2003) visat att supporten genom s.k. communitybildning ofta är lika omfattande för komponenterna som drivs som ”öppen källkod”-projekt. Lakhanis och von Hippels (2003) arbete visar att av alla frågor som ställdes inom Apache-communityn, blev över hälften besvarade samma dag eller dagen efter. Undersökningen visar också att nästan 97 % av användarna som besvarade andra användares frågor inte kände individen i fråga. De största drivkrafterna till att agera support var både ett sätt att ge någonting tillbaka till komponentprojektet och att skapa en känsla av kompetens hos sig själv.

Bland de kommersiella alternativen finns unika komponenter som inte har någon motsvarighet bland de öppna. Det kan röra sig om komponenter som presterar extraordinärt bra på sina specialområden, eller komponenter som har funktioner unika i sammanhanget. Man bör således se kommersiella komponenter och komponenter med öppen källkod som komplement snarare än konkurrenter (Bessen 2005).

3.2 PHP: Hypertext Preprocessor

PHP är ett mycket utbrett skriptspråk som går att använda till både vanliga applikationer och webbprogrammering. Det är dock speciellt lämpligt för användning på Internet och kan bäddas in i vanlig html-kod (PHP.net, 4/3 2009).

PHP skapades 1994 av Rasmus Lerdorf, och bestod då av några funktioner för att räkna besökare och hantera gästböcker på hemsidor skrivna i Perl/CGI. Sedan dess har mycket hänt. Redan året efter släppte Lerdorf en ny version med funktioner skrivna i programmeringsspråket C, och kallade paketet för "Home Page Tools" (därav förkortningen PHP). Två år senare formligen exploderade användandet då version 3 släpptes. Den hade helt ny funktionalitet i form av en skriptexekveringsmotor, och PHP hade således tagit steget från hjälpbibliotek till fullfjädrat skriptspråk. Eftersom funktionaliteten hade förändrats så drastiskt, döptes språket om till den rekursiva akronymen PHP: Hypertext Preprocessor (Cullen, K. 2002).

Några av PHPs huvudsakliga konkurrenter är Perl, Microsoft ASP.NET, JavaServer Pages (JSP), och ColdFusion. Jämför man med dessa lösningar har PHP flera styrkor (Welling och Thomson, 2005):

- **Prestanda.** Det finns många väldigt välbesökta sidor på Internet som använder PHP. En liten, lågpresterande server räcker för att klara flera miljoner besök per dag.

- **Databasintegration.** PHP har inbyggd funktionalitet för att interagera med databassystem så som MySQL, PostgreSQL, mSQL, Oracle, dbm, FilePro, Hyperwave, Informix, InterBase, Sybase o.s.v.

- **Inbyggd funktionalitet.** Eftersom PHP designades för användning på nätet, finns det funktioner för cookie-hantering, PDF-dokument, GIF-bilder, email, och mycket annat inbyggt från start.

- **Låg kostnad.** PHP är gratis, och alla vanliga servrar kan köra det som standard.

- **Låg inlärningskurva.** Eftersom syntaxen i språket är baserad på andra språk, främst C och Perl, kan programmeringsvana användare komma igång med att koda i PHP på några minuter.

- **Objektorientering.** Från och med PHP version 5 finns all vanlig funktionalitet för objektorienterad programmering inbyggd. Denna inkluderar arv, abstrakta klasser, gränssnitt och objekt.

- **Plattformsoberoende.** PHP finns tillgängligt till väldigt många operativsystem. Alla populära har fullt stöd för språket och kod behöver inte modifieras för att fungera på olika plattformar.

- **Öppen källkod.** Med källkoden tillgänglig och en stor användarbas behöver man inte oroa sig för att inte få support eller att språket skulle upphöra att utvecklas. Man kan också lägga till egen funktionalitet eller ge förslag till utvecklarna.

Det finns dock vissa svagheter som bl.a. Cullen (2002) belyser. PHP saknar en standardiserad utvecklingsmiljö. Istället finns det en mängd olika alternativ, alla med styrkor, svagheter och egenheter. Förutom avsaknaden av standardisering kan en del webb-editorer ha problem att känna igen PHP-kod, och förstöra den när man försöker redigera en html-sida med inbäddad PHP. Tidigare versioner av PHP hade sämre felhantering än t.ex. ColdFusion, men detta har åtgärdats i senare versioner. Just att PHP är drivet på frivillig basis är dock ett problem, eftersom inget stort företag garanterar support och kontinuerliga uppdateringar. Sedan företaget Zend startades har dock detta problem marginaliserats. Zend sköter nu uppdateringar och support för PHP, uppbackat av fristående utvecklare. Det återstår dock ett för många störande problem med PHP, och det är att varje databas har en egen uppsättning kommandon direkt integrerade i språket. Vill man byta databas, måste man ersätta alla dessa kommandon med nya. Det finns ett abstraktionslager som åtgärdar detta problem under utveckling, men som standard använder man fortfarande olika kommandon beroende på databas (Cullen, K. 2002).

3.3 Databaser

Det finns flera stora databassystem på marknaden. Oracle och Microsoft SQL server är två stora kommersiella alternativ, medan PostgreSQL är en akademiskt utvecklad databas med öppen källkod (Locke, 2004). MySQL är troligtvis den mest populära databasen i dag, och den har öppen källkod men är ändå utvecklad kommersiellt och har således dubbla licenser för kommersiellt och icke-kommersiellt bruk (Welling och Thomson, 2005).

Om man är intresserad av att använda öppen källkod finns det därmed egentligen bara två stora alternativ: MySQL och PostgreSQL. PostgreSQL har länge stött s.k. databastransaktioner och sub-urval, vilka båda är bra att ha för datasäkerhet i t.ex. ekonomisystem (Locke, 2004). PostgreSQL har också mycket funktionalitet för att hantera komplexa applikationer och göra mycket av jobbet åt programmeraren (Locke, 2004).

Gemensamt för samtliga databaser är de designtekniker som brukar användas när tabellerna, attributen och deras relationer bestäms i ett specifikt projekt. En av dessa tekniker kallas för normalisering, och är beskriven nedan.

3.3.1 Normalisering av databaser

Normalisering går i korthet ut på att reducera den sparade informationen till ett minimum för att spara lagringutrymme samt få alla logiskt sammanhängande attribut att ligga i samma tabeller. Flera olika normalformer har definierats, där varje ny form kräver att villkoren för den föregående är uppfyllda. Andra normalformen (2NF) kräver därmed att villkoren för första normalformen (1NF) är fullständigt uppfyllda. Första normalformen definieras som att varje attribut i en databas bara får innehålla ett värde, och att varje rad måste vara unik jämfört med andra rader. Andra normalformen utökar kraven och kräver att inga funktionella beroenden mellan delar av primärnyckeln och attribut i tabellen existerar (Connolly och Begg, 2005).

Ett vanligt problem som undviks med normalisering är att människor tenderar att gruppera flera värden i samma fält i en tabell (figur 3.1).

OrderID	Summa	Produktnummer
1	15	3
2	28	3 och 4
3	49	2, 5 och 3

Figur 3.1. Onormaliserad tabell med flera produkter i samma fält.

När man exempelvis ska skapa en design för att lagra ordrar, börjar många med att göra en tabell för ordrar där varje rad har en ordernyckel. Så långt är allting bra, men sedan fortsätter de flesta med att försöka spara alla produkter som hör till det givna ordernumret på samma tabellrad. Detta leder till att miniatyrtabeller skapas för lagring av produktnumren, och försvårar databasfrågor som t.ex. "hur många produkter med nummer x har blivit beställda", eftersom detta tvingar en genomsökning av samtliga miniatyrtabeller (Welling och Thomson, 2005). Lösningen i detta fall är att helt enkelt skapa en ny tabell med produktnummer som kan länkas från ordertabellen istället för själva produkterna, illustrerad i figur 3.2 nedan.

OrderID	Produktnummer	Antal
1	3	1
2	3	1
2	4	1

3	2	1
3	5	1
3	3	1

Figur 3.2. Exempel på lösning för ordertabell.

3.3.2 Databastransaktioner

Databastransaktioner är när man skickar flera kommandon till databasservern samtidigt, och om något kommando misslyckas utförs inte de andra heller (Locke, 2004). Det finns fyra villkor som ska vara uppfyllda för en transaktion. För det första ska en transaktion vara atomisk; antingen utförs den fullständigt eller inte alls. För det andra måste den lämna databasen i ett öppet tillstånd, redo för nästa kommando. För det tredje får inte en ännu ej helt slutförd transaktion synas för andra användare. Slutligen ska resultatet av en slutförd transaktion vara bestående (Welling och Thomson, 2005). Detta är användbart i ekonomisystem, eftersom en misslyckad kontokreditering efter en lyckad kontodebitering skulle leda till obalanserade konton.

För att påbörja en databastransaktion i en MySQL-databas som inte är inställd på att inte använda transaktioner som standard kan man ge kommandot ”start transaction;” till MySQL. När man sedan vill utföra efterföljande databasförfrågningar ger man kommandot ”commit;”. Det går även att avbryta transaktioner genom att ge kommandot ”rollback;” (Welling och Thomson, 2005).

3.4 MySQL

3.4.1 Prestandaoptimering

Prestandaoptimering går till på liknande sätt i alla databaser, och samma grundläggande principer gäller överallt. För att uppnå optimal prestanda är det några saker man bör ta i beaktande (Welling och Thomson, 2005):

1. Optimera designen. Allt i databasen ska vara så litet som möjligt. Man bör också minimera antalet tillåtna ”NULL”-värden, som innebär kolumner som får vara ”tomma”. Viktigast att hålla nere storleken på är de primära indexen, som söks upp snabbast om de består av enkla nummer. Kolumner med tillåten variabel längd (som t.ex. varchar, text och blob) bör också undvikas om inte den potentiella platsbesparingen är stor.

2. Privilegier (rättigheter). Eftersom varje SQL-kommando kräver kontroll av användarens privilegier, kommer enkla privilegier utan komplexa villkor leda till bättre prestanda.

3. Tabelloptimering. Data kommer fragmenteras efter viss tid om uppdaterings- och borttagningskommandon utförs ofta. Det finns speciella kommandon i MySQL som defragmenterar tabeller och därmed gör sökningar snabbare.

4. Index. Kolumner som söks igenom ofta kan indexeras för att snabba upp sökningarna. För många index leder dock till att onödigt mycket utrymme krävs och sökningar går långsammare, så en avvägning måste göras.

5. Genom att använda standardvärden på så många kolumner som möjligt och inte sända andra värden oftare än nödvändigt kan s.k. ”insert”-kommandon utföras snabbare på de aktuella tabellraderna. Ett vanligt standardvärde är att låta det primära indexet för varje rad i en tabell vara ett nummer som automatiskt ökas med 1 för varje ny tillagd rad.

Man måste dock alltid tänka på att prestandaråd som dessa inte kan ersätta en väl genomtänkt design anpassad för den aktuella situationen.

3.4.2 Lagringsmotorer

Det finns flera olika s.k. lagringsmotorer att välja bland i MySQL. Man kan även använda flera olika motorer inom samma databas. Varje motor har sina styrkor och svagheter, men de som används oftast i MySQL heter MyISAM och InnoDB (Welling och Thomson, 2005).

MyISAM används som standard om inget annat anges (ISAM står för Indexed Sequencial Access Method), och är mycket snabb samt innehåller många funktioner för reparation och felsökning av tabeller. Den möjliggör även komprimering av data och fulltextsökning. Främmande nycklar och fulltextsökning stöds dock inte, och detta försöker InnoDB råda bot på (Welling och Thomson, 2005).

InnoDB är långsammare än MyISAM, men skillnaden är liten. Istället stöds främmande nycklar och databastransaktioner. Man kan som sagt använda olika motorer inom samma databas, och det görs då på tabellnivå (Welling och Thomson, 2005). InnoDB kan då användas till exempelvis ekonomiska tabeller och MyISAM till övriga (beroende på innehåll).

3.4.3 Säkerhet

PHP har som tidigare nämnts flera inbyggda funktioner för att interagera med olika databaser, bl.a. MySQL. Användare lagras med skilda privilegier, och man kan ha olika inloggningar till databasen beroende på vad användaren måste kunna göra. Generellt bör man inte ge en användare mer makt över databasen än nödvändigt. Man ansluter till MySQL med ett kommando där man ger användarnamn och lösenord, och kan sedan koppla från med ett annat när informationen är hämtad från databasen. Kommandot för att koppla ifrån är inte helt nödvändigt, eftersom detta sker automatiskt när skriptfilen slutar exekveras (Welling och Thomson, 2005).

Information som webbsidebesökare matar in, som ska lagras i databasen, bör förberedas innan den skickas till MySQL. Detta för att vissa tecken har speciell betydelse för MySQL och därför måste maskeras med s.k. escape-tecken (tecken som talar om för MySQL att ett specialtecken följer), och onödiga mellanslag ska tas bort. För denna förbehandling av text finns speciella funktioner i PHP (Welling och Thomson, 2005).

3.5 Kryptering

Inte bara servern, utan även databasen på servern är ofta lösenordsskyddad. Databaser har generellt sett stöd för olika användare med olika privilegier. En viss webbsida behöver inte binda sig till att enbart använda en användare, utan kan med fördel använda flera beroende på den aktuella besökarens behov av databasaccess. Genom att ha en speciell användare med extra databasprivilegier för t.ex. webbplatsadministratörer, kan potentiella säkerhetsrisker bland vanliga besökare undvikas (Welling och Thomson, 2005).

Även om både databas och PHP-kod använder inloggningsmekanismer finns det en risk i att skicka t.ex. lösenord över Internet, eftersom kommunikation kan avlyssnas. Krypterade lösenord förbättrar säkerheten, men även sådana kan i vissa fall användas för intrång om de kan avlyssnas. Lösningen på sådana problem är säkra förbindelser, som uppnås genom en kombination av krypteringsmetoder. Den vanligaste kallas SSL (Locke, 2004).

3.5.1 Secure sockets layer

Secure Sockets Layer (SSL) kallades några av de första versionerna av ett system för säkra, krypterade webbservrar. Det nuvarande korrekta namnet för systemet är Transport Layer Security (TSL), men de flesta kallar det fortfarande för SSL. Kärnan i systemet är att använda asymmetrisk kryptering, där olika nycklar används för att kryptera och dekryptera information. Genom sådan kryptering överförs en helt vanlig, symmetrisk, krypteringsnyckel som alltså är samma på server och klient. Den faktiska informationen överförs sedan efter kryptering med denna nyckel. Den asymmetriska krypteringen, även kallad RSA efter initialerna i de tre uppfinnarnas efternamn, går ut på principen att det tar väldigt lång tid att hitta primtalsfaktorerna i ett stort tal. Algoritmen är inte oknäckbar, men med dagens datorer skulle det ta allt för lång tid att knäcka för att vara praktiskt möjligt. Anledningen till att inte all data krypteras asymmetriskt är att detta kräver mer beräkningskapacitet än symmetrisk kryptering, men genom kombinationen av metoder blir kommunikationen så gott som oknäckbar ändå (Locke, 2004).

3.6 Ekonomiska system

3.6.1 Dubbel bokföring

Dubbel bokföring har under mer än 500 år varit den föredragna metoden att bokföra affärshändelser. Grundkonceptet är att varje transaktion bokförs på två konton, och man får därigenom en felkorrigerande mekanism som är mycket värdefull när bokföring sker för hand med penna och papper. När datorer gjorde sitt intåg på området blev denna mekanism helt plötsligt onödig, eftersom datorer inte gör sådana fel, och mekanismen kräver mer minne än nödvändigt eftersom varje siffra återfinns på två ställen (Perry och Schneider, 2003).

Felkorrigering är dock inte enda anledningen att använda dubbel bokföring, som Locke (2004) påpekar. Det går att utläsa mer information ur ett sådant system, och bokföring blir i viss mån enklare. Privatpersoner och små organisationer har ofta bara noteringar på vilka inköp som gjorts och av vilken typ. Banker och företag använder fortfarande för det mesta dubbel bokföring istället. Ena kontot i en transaktion debiteras, och det andra krediteras. När man sedan räknar ut saldon på konton, kan de ha debit- eller kreditsaldon. Kreditsaldon är summor som organisationen är skyldig någon, medan debetsaldon är pengar som organisationen troligtvis kommer få eller redan har. Fördelen med dubbel bokföring är att varje belopp både går att härleda varifrån det kom, och till vad det använts. Det är dessutom enkelt att hantera fordringar och skulder vilket företag måste kunna, och nästan alla stora mjukvarusystem inom ekonomi använder sig således av dubbel bokföring (Locke, 2004).

3.6.2 Kontantmetoden

Kontantmetoden är ett sätt att bokföra som innebär att man bokför inkomster och utgifter när betalning sker. Det vanligaste sättet att bokföra är egentligen fakturametoden, där bokföringen sker redan när fakturor erhålls snarare än när betalning sker, eftersom lagen kräver det i de flesta fall. Ideella organisationer och enskilda näringsidkare får dock använda kontantmetoden då denna anses enklare (bokforingstips.se, 2009).

3.7 Användarvänlighet

3.7.1 AJAX

Asynkront Javascript och XML är en teknik för att skapa bättre användarinteraktion på webbsidor. Grunden är att javascript gör det möjligt att undvika den så vanliga ”skicka - invänta svar”-cykeln som alla internetanvändare är så vana vid. Ett javascript som körs i klientens webbläsare skickar en förfrågan till servern om uppdaterat innehåll, när användaren t.ex. börjar skriva i en textbox. På detta sätt kan användaren snabbt få information om det denne skriver i boxen är felaktigt på ett eller annat sätt (Olson, S.D. 2007).

Man skulle i praktiken kunna bygga en hel webbplats runt AJAX. Dock kan då problem uppstå med webbläsarens bakåt-knapp etc eftersom webbläsaren ser samma adress hela tiden, oberoende av vilket innehåll användaren tittar på (Olson, S.D. 2007).

4 Genomförande

Genomförandet är strukturerat efter OOS / UML-modellen som beskrevs i *2.2 Utvecklingsmodell*. Den första fasen, ”tidig analys”, innehåller planering, verksamhetsanalys, lite UML-modellering och bedömning av de långsiktiga kraven på systemet. Fas två kallas ”prototyputveckling” och innebär en fortsatt analys av verksamheten, databasutveckling och gränssnittsdesign mm. I tredje fasen påbörjas konstruktion av den ”slutgiltiga” produkten, och det är då inte ovanligt att en vidareutveckling av prototypen används som bas (Apelkrans och Åbom, 2001).

4.1 Fas 1 – planering

4.1.1 Verksamhetsbeskrivning

Vanliga medlemmar ur klubben tar på sig ansvar för att sköta försäljning. Medlemmen tränar ofta samtidigt som han/hon sköter försäljningen, så kunder får ibland vänta lite på hjälp när de vill köpa något. Medlemsavgifter noteras i en pärm, och medlemmen som ansvarar för försäljningen för tillfället tar med sig kassan hem för senare inlämning till någon som samordnar ekonomin. Kosttillskott har en separat kassa, som lämnas i gymmet men töms regelbundet.

4.1.2 Milstolpar

För att kunna kontrollera att projektet utvecklas i rätt riktning och med tillräcklig fart, sätts milstolpar i form av olika kravspecifikationer upp. Varje uppnådd milstolpe diskuteras med personer som senare ska använda systemet, och feedback samlas på så vis in.

Milstolpe 1

Namn: Prototyp

Klar: Mars

Egenskaper: Enkelt medlemshanteringssystem (registrera, logga in, logga ut), försäljningssystem (sälj från medlem till medlem, användarvänligt med automatkomplettering i textfält), CSS-stil (temporär), databashantering, bokföring av affärshändelser, enkel rapportvisning.

Milstolpe 2

Namn: Beta 1

Klar: April

Egenskaper: Kryptering av lösenord, CSS-stil (genomarbetad), rapportvisning, administratörsprivilegier för utvalda medlemmar (medlemmar ska kunna listas och administratörsprivilegier ställas in).

Milstolpe 3

Namn: Version 1.0

Klar: Maj

Egenskaper: Verifikationer ska kunna skapas och läggas till, med valfritt antal transaktioner på varje. Prislista för försäljning ska kunna visas och redigeras, och produkter placeras i kategorier. En kontoplan ska kunna visas och redigeras något, och medlemmar ska kunna sökas. Man ska även kunna se om en medlem betalat träningsavgift för perioden. Olika databasprivilegier för olika användare.

4.1.3 Val av komponenter och verktyg

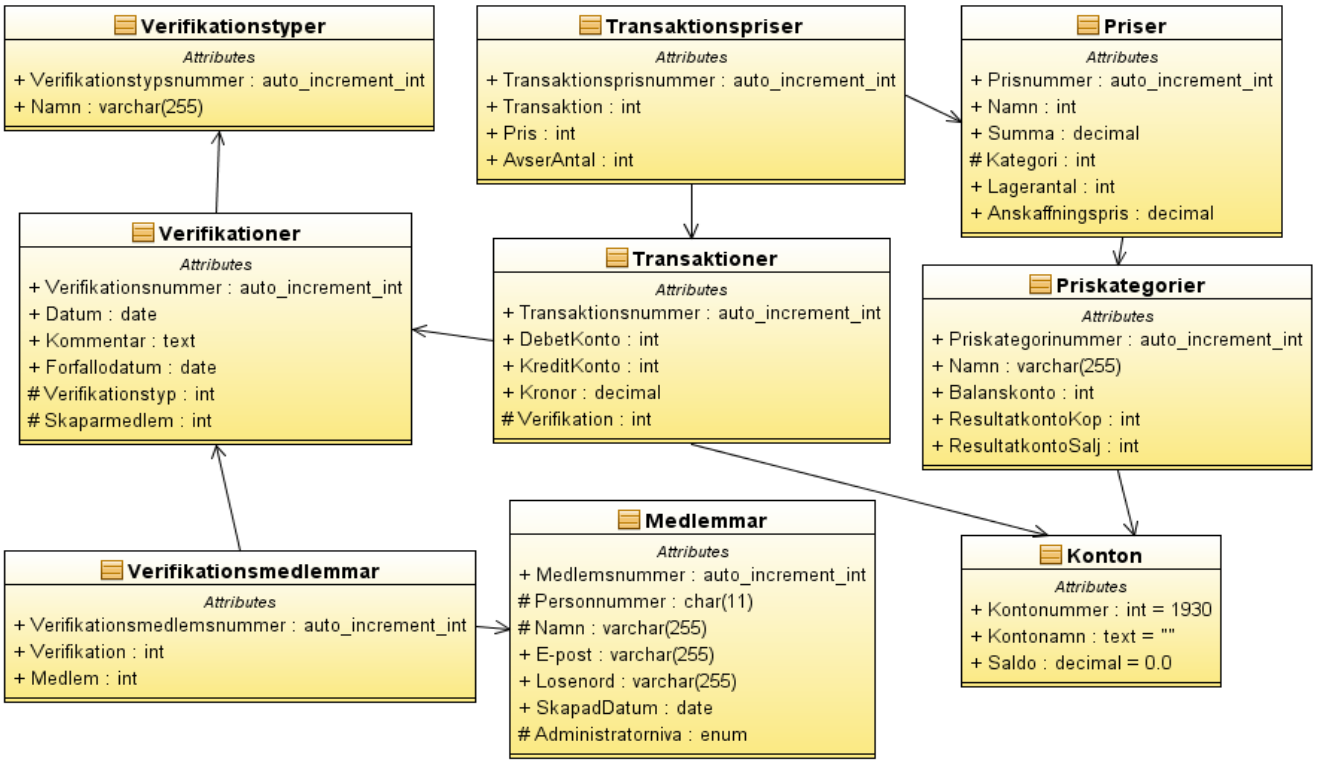
Under grundutbildningen låg fokus på Microsofts utvecklingsplattform kallad ”.Net”, med utvecklingsmiljön Visual Studio och webbprogrammering genom Asp. Valet att använda PHP som språk och Netbeans som utvecklingsmiljö baserades främst på två kriterier. För det första har PHP och tillgängliga utvecklingsmiljöer med öppen källkod så som Eclipse och Netbeans utvecklats och mognat väldigt mycket de senaste åren kanske tack vare den hårda konkurrensen (Unger 2005), och för det andra finns det uppenbara utmaningar med att låsa sig till en företagsspecifik (Microsoft) teknologi (Currie et al. 2004). Mjukvara med öppen källkod har generellt sett betydande styrkor i kostnadseffektivitet och bra support, men kanske inte finns med den funktionalitet man önskar (Lakhani och von Hippel, 2003; Bessen 2005). Detta diskuteras vidare i *3.1 Öppen källkod*.

Förutom NetBeans och Eclipse utvärderades även utveckling i Aptana Studio, som är en java-baserad utvecklingsmiljö, samt i Notepad++, som är en enkel texteditor med funktioner som syntaxigenkänning för programmeringsspråk och flikhantering. NetBeans valdes slutgiltigt som utvecklingsmiljö, med stöd av Notepad++. Eclipse valdes bort eftersom det inte är lika smidigt att komma igång med eftersom funktionalitet får installeras som paket efter hand, och Aptana Studio innehöll inte lika många funktioner och inte heller lika bra dokumentation.

4.2 Fas 2 – prototyputveckling

4.2.1 Databasutveckling

Databasen har två huvudsakliga krav i detta projekt: att vara flexibel nog att kunna utökas utan att designas om totalt och att stödja både vanliga webbshops-liknande funktioner och bokföringsmässig funktionalitet. Genom dessa krav har databasens delar samlats kring en central tabell med s.k. verifikationer.



Figur 4.1 Databasdesign. Pilar anger ett många-till-en-förhållande i pilens riktning. ”#” anger att värdet indexeras och därmed kan sökas upp snabbare av MySQL. Efter kolon står typen för varje dataelement: int är heltal, varchar(255) är en maximalt 255 tecken lång textsträng, date är datum, decimal är ett decimaltal och enum betyder att några utvalda läsbara värden är möjliga.

En verifikation innehåller information som datum när den skapades, medlem som skapade den, och vad den avser för affärshändelse. Denna information är fortfarande inte nog för att tillfredsställa de olika funktionerna i programmet, och för att lösa detta kopplas transaktioner från en transaktionstabell och övrig verifikationsdata till verifikationen efter behov. Transaktionerna utgör då databasens bokföringsmässiga del, medan den övriga datan kan bestå av vilka produkter som sålts i en säljverifikation och vilken medlem som köpte den.

Tidigt i utvecklingsprocessen bestämdes att varje tabell skulle få ha enkla automatiskt ökande primära indexeringsnycklar istället för komplexa nycklar bestående av flera element. Detta innebär att både första och andra normalformen uppnås utan hinder, och prestandan och enkelheten som vinns på detta är värd det extra lilla utrymme dessa nycklar tar i anspråk.

4.2.2 Felhantering i databasen

Storleken på detta projekt och faktumet att enbart speciellt utvalda personer ska hantera systemet har gjort detta område lägre prioriterat än andra. Vissa aspekter har dock tagits hänsyn till, så som säkerhetskopiering av databasen, utökade möjligheter att göra förändringar i redan inmatade data och trasiga data till följd av anslutnings- eller webbläsarproblem.

Problemet med att göra t.ex. kreditering och debitering och misslyckas med den ena i ett databassystem är att det kan leda till obalanserade konton (Locke, 2004). Detta kräver vanligtvis s.k. databastransaktioner för att lösa, men problemet undviks istället här genom att både debitering och kreditering sker i samma kommando. Ett liknande problem kvarstår dock i och med att verifikationer och verifikationsdata är uppdelade i olika tabeller, och ibland kräver fler än ett sql-kommando för att uppdatera. Detta är inget stort problem då det är mycket ovanligt så länge alla kommandon som berör en viss operation utförs från samma PHP-sida. Webbläsaren behöver då inte hämta nya sidor för att slutföra databasuppdateringen. Problemet skjuts därmed på framtiden.

4.2.3 Programdesign

Eftersom tidigare versioner av PHP inte innehöll funktionalitet för objektorientering (Welling och Thomson, 2005), har praxis blivit att inte använda PHP på ett fullständigt objektorienterat sätt. För att utnyttja kraften i objektorientering, men inte låta ett allt för strikt struktureringskrav utgöra ett hinder för snabb utveckling delas koden i projektet in i tre delar:

- **Sidor.** Blandad HTML och PHP, som bildar webbplatsens struktur. Dessa gör att webbläsarens bak- och framknappar och bokmärken fungerar eftersom varje undersida motsvaras av en egen PHP-fil (Olson, S.D. 2007).

- **Klasser.** Instansieras i sidorna och kapslar in funktionalitet specifik för ett visst område. Exempelvis finns det en klass som hanterar databashanteringen, och en annan som sköter autokomplettering av textfält med hjälp av AJAX. Även element som ska synas på flera undersidor på webbplatsen kapslas in i klasser, exempelvis navigationspanelen och inloggningspanelen.

- **Skript.** När man klickar på en skicka-knapp på en sida är det i HTML inbyggda svaret att skicka användaren till en ny sida. Denna nya sida har här inte givits någon design, utan behandlas istället som ett skript som utför operationer på den av användaren inmatade informationen på föregående sida. När skriptet körts färdigt skickas användaren till en vanlig sida igen.

Återanvändning av kod genom arv används där det anses leda till mindre arbete och förenklat underhåll, och således inte främst för att göra koden strukturerad. Om tid finns kan delar av koden sippras uppåt i arvshierarkin i ett senare skede, men första prioritet är att uppnå den funktionalitet som krävs på ett genomtänkt sätt.

4.3 Fas 3 – Konstruktion

Enligt OOS / UML-modellen är det vanligt att man fortsätter utveckla prototypen som den slutgiltiga lösningen (Apelkrans och Åbom, 2001), och så sker även här. De huvudsakliga förändringarna bestod i att göra projektet öppet för vidare utveckling i ett senare skede, och att förbättra funktionalitet som inte är kritisk för användbarhet. Sådan funktionalitet är exempelvis säkerhet, prestanda och utseende.

Databasen gjordes om något för att tillåta vidare utveckling efter att detta projekt är färdigt. Verifikationer och transaktioner fick möjlighet att innehålla olika typer av data beroende på typ, och medlems- och pristabellerna kopplades på så sätt dynamiskt till verifikations- och transaktionstabellerna. Detta är positivt för framtida vidareutveckling av produkten.

Säkerhetsfunktionalitet var redan delvis implementerad i prototypen. Det som saknades var bl.a. att kunna koppla varje inloggning till ett databaskonto med lämpliga rättigheter. Detta är viktigt för att inte användare ska kunna kringgå sina rättigheter i systemet och få tillgång till information de inte bör kunna se. Säkra överföringar med SSL undersöktes också, men valdes bort tills själva installationen av systemet skulle ske.

Tester av sidans utseende och funktionalitet utfördes i Mozilla Firefox och Microsoft Internet Explorer, och justeringar av kod gjordes på de platser där det fanns betydande skillnader.

5 Analys

Analysen undersöker det färdiga resultatet med stöd av teorin. Funktionalitet och struktur behandlas och jämförs med krav från organisationen.

5.1 Sidans delar och funktionalitet

Det är viktigare att det går att nå en viss sida snabbt, än att en ny användare ska kunna hitta dit eftersom det kommer vara ett relativt begränsat antal användare av systemet som dessutom kommer ha tid på sig att lära sig använda det. Därför läggs alla undersidor som alternativ direkt i huvudmenyn, och kräver därmed bara ett klick för att navigeras till.

Figur 5.1. Huvudmenyn.

5.1.1 Medlemmar och inloggning

Designen av systemet fokuseras som tidigare nämnts runt verifikationer. En verifikation sparar vem som skapat den, eftersom det gör det lättare att hålla koll på vem som för tillfället ansvarar för delar av kassan, vem som sålt vissa produkter osv. Att fylla i sina personuppgifter varje gång man säljer någonting skulle bli tidskrävande och mindre användarvänligt, och därför skapas ett inloggningssystem som håller reda på medlemmar och administratörsprivilegier. Samma medlemsregister används för inloggning och för att hålla reda på medlemmar att sälja i huvudsak träningskort till, eftersom träningskorts giltighet måste kunna kontrolleras snabbt mot databasen. För att uppfylla kravet på säkerhet krypteras lösenorden tillsammans med inloggningsnamnet och sparas i databasen. Inloggningen i själva databasen görs varje gång databasen ska användas, och avslutas direkt när den inte behövs längre i varje skriptfil. Man behöver egentligen inte avsluta inloggningen till databasen eftersom det sker i slutet av varje skriptfil automatiskt (Welling och Thomson, 2005), men det görs ändå här för att det är praxis och för att inte upprätthålla anslutningar till databasen längre än nödvändigt. För att uppnå ännu bättre säkerhet bör SSL användas åtminstone vid inloggning i systemet, men detta måste genomföras under fas 4 i utvecklingsprocessen vid installationen av systemet. Detta för att SSL konfigureras på datorn som ska agera server för applikationen. Den asymmetriska krypteringen gör så att information kan skickas i princip utan att kunna läsas av någon utomstående alls, eftersom bara servern känner till all information som krävs för att avkoda den (Locke, 2004).

5.1.2 Inbetalningar / sälj

Försäljningssidan är den inledningsvis viktigaste delen på webbplatsen, eftersom det är främst försäljning av träningskort som ska kunna hanteras till en början. Endast försäljning av träningskort kräver att köparen anges, och man får därför välja produktkategori först innan övriga uppgifter matas in. För att visa alla produkter i den valda kategorin utan att sidan behöver hämtas om, används ett xAJAX-skript. XAJAX använder AJAX som bas. AJAX är en teknik för att uppdatera delar av sidan utan att hämta om hela (Olson, S.D. 2007).

Varje ny produkt som köps skapar en verifikation, som dyker upp nedanför säljformuläret. När alla produkter som ska säljas till aktuell köpare valts, kan man avläsa totalt pris under verifikationerna. Om någon produkt valts som kräver information om köparen, dyker även sådana fält upp. För att snabbt kunna ange köpare bland nuvarande medlemmar, används AJAX även här för att uppdatera innehållet i en söklista beroende på vilka bokstäver eller siffror man matat in i namn- respektive personnummerboxarna. Finns ingen träff bland befintliga medlemmar, läggs det nya namnet och personnummret helt enkelt till som en ny medlem, och köpet knyts till denna. Detta system skapades för att göra försäljningen så snabb som möjligt, utan krav på att gå till speciella sidor för att registrera eventuella nya medlemmar.

Där finns även en ruta för mottagen summa. Efter att ha accepterat verifikationslistan genom att trycka på säljknappen längst ner visas växel om summa-rutan fyllts i, där verifikationslistan tidigare var. Möjliga svagheter med denna design är att rutan för mottagen summa kan bli ifylld för tidigt av säljaren av misstag, och ett avbrytet köp kräver ändringar i databasen eftersom verifikationerna för varje köpt produkt sparas löpande istället för alla på en gång. Detta kan komma att ändras, men ännu har inte detta visat sig vara en nackdel av betydelse.

Om detta ändras, kommer det ske genom användning av databastransaktioner. Eftersom lagringsmotorn MyISAM används genom hela databasen i nuläget, finns inget stöd för det just nu. Det är dock enkelt att byta lagringsmotor på enskilda tabeller till t.ex. InnoDB, som stöder databastransaktioner (Welling och Thomson, 2005).

Säljsidan använder förutom medlemstabellen även en pristabell, som kan redigeras separat på en pris-sida. Varje pris har dessutom en kategori för att kunna skilja t.ex. träningskort från kosttillskott.

5.1.3 Utbetalningar / köp

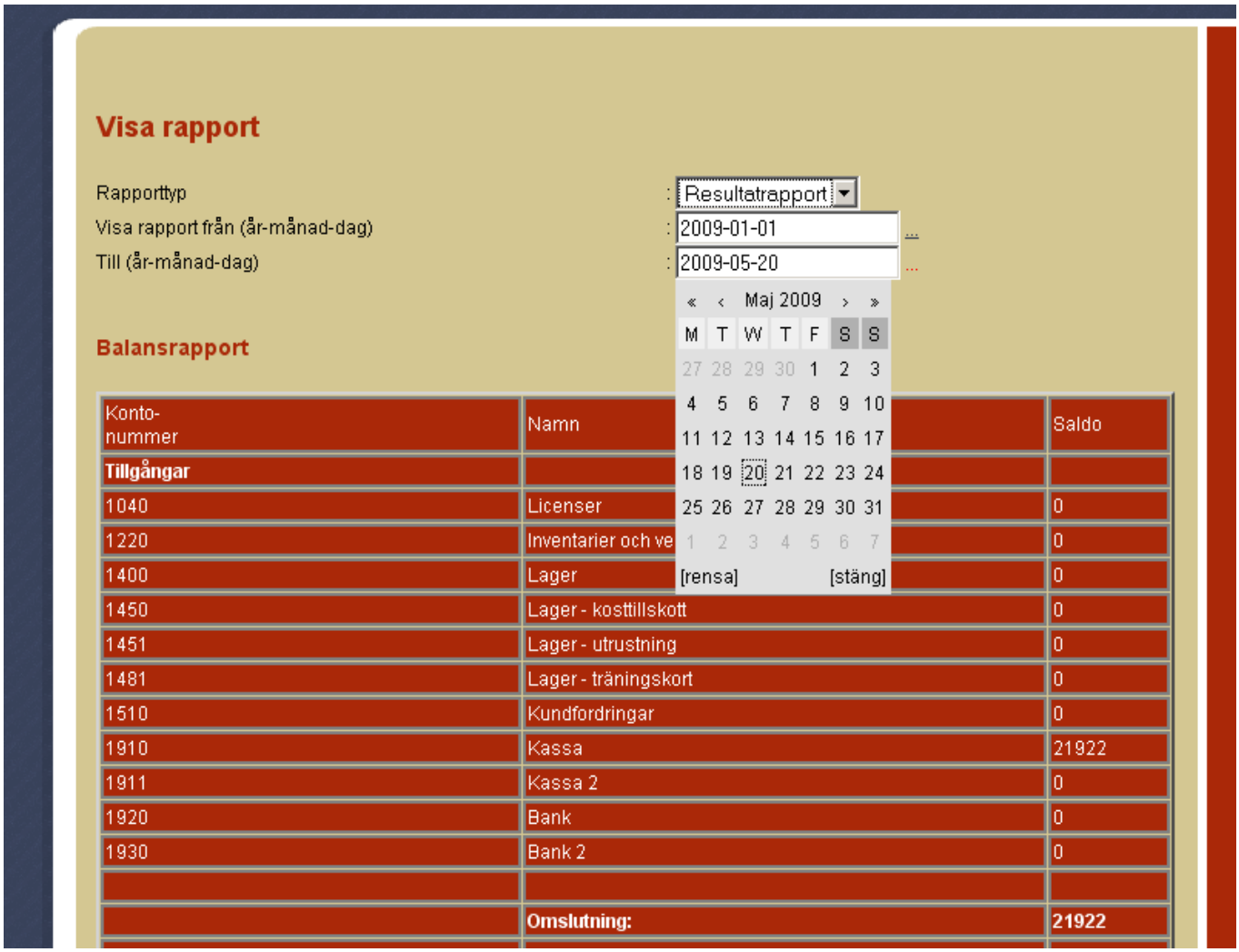
Inköp hanteras på liknande sätt som försäljning, men istället för en eventuell köparidentitet sparas alltid säljaren och medlemmen som utför köpet. De vanligaste betalningsströmmarna ut från organisationen kan hanteras via denna sida, även sponsring och ersättningar.

5.1.4 Rapporter

Rapporter är till för att klara av årsredovisningen, samt att kunna se variationer i den ekonomiska ställningen över tid. Två typer av rapporter stöds: resultatrapport och balansrapport. Rapporterna går att visa mellan enskilda datum för resultatrapporten eller vid enskild tidpunkt för balansrapporten. En resultatrapport visar hur väl det har gått ekonomiskt under en tidsperiod, och en balansrapport visar hur kapitalet i föreningen använts på ena sidan och anskaffats på andra. Rapporterna är standard inom ekonomi och krävs bl.a. i årsredovisningen för företag.

För att på ett enkelt sätt upprätta rapporter som dessa, är dubbel bokföring att föredra. Eftersom dubbel bokföring använts under flera tusen år (Perry och Schneider, 2003), har det blivit standardiserat. Det kan finnas problem med att mer minne i datorn används för att lagra alla siffror än nödvändigt, eftersom systemet kräver att varje siffra återfinns på två platser (Perry och Schneider, 2003), men fördelarna är för de flesta företag större än nackdelarna. Man kan exempelvis se varifrån pengarna har kommit, och till vad de använts (Locke, 2004). Detta är en förutsättning för att kunna upprätta en balansräkning, eftersom den visar just detta.

Dubbel bokföring förenklar även hanteringen av fordringar och skulder (Locke, 2004), men detta är inte någon uppenbar fördel i detta fall då kontantmetoden används i de flesta fall. Kontantmetoden fokuserar på betalningarna som bokförs samtidigt som de inträffar, snarare än löften om framtida betalningar (bokforingstips.se, 2009). Systemet är ändå utvecklat för att stödja faktureringar och skulder, om organisationen i framtiden skulle få behov av sådan funktionalitet.



Figur 5.2 Rapportvisning. Efter val av rapporttyp kan datum väljas med hjälp av en kalender eller skrivas in manuellt.

5.2 Organisationens mål

1. *Systemet ska vara användarvänligt (personer utan ekonomisk utbildning ska kunna hantera det) och snabbt (man ska kunna bokföra betalningar när de inträffar, som en kassaapparat).*

Genom att logga in när man ska ansvara för försäljningen en stund och fortsätta vara inloggad under hela den tiden kan man utföra försäljningar snabbt. Det finns även stöd i form av autokomplettering av textfält när man ska skriva in köparens namn eller personnummer, om det krävs. All funktionalitet går även nå direkt från huvudmenyn, för att göra systemet snabbarbetat.

För att upprätthålla prestandan i applikationen har främst databasen optimerats. Under designmomentet har Wellings och Thomsons (2005) rekommendationer beaktats. Enligt första rekommendationen är inga NULL-värden är tillåtna, datatyperna som används är så små som möjligt, och enkla nummer används som index. Andra punktens rekommendation att privilegier ska hållas så enkla som möjligt uppfylls genast, eftersom projektet inte kräver några avancerade privilegiestrukturer. Index har även lagts till kolumner som beräknas genomsökas ofta, enligt den fjärde rekommendationen. Standardvärden används även det på så många ställen som möjligt.

2. *Det ska finnas stöd för att betala medlemsavgifter med utgångspunkt från olika priskategorier, så som studerande, tävlande, arbetslös osv.*

Eftersom bara träningskort kostar olika för olika grupper, skapas olika kort för olika kategorier. Detta sparar lite arbete.

3. *Systemet ska kunna sammanställa inbetalningar och utgifter på månads- och årsbasis.*

Enkla resultatrapporter går att visa, och möjlighet att utöka med mer avancerade rapporter finns.

4. *Ett medlemsregister ska finnas, med grundläggande uppgifter om medlemmarna.*

Medlemsregistret används till snabbuppslagning av medlemmar när man säljer medlemskort, inloggning och för att spara övriga uppgifter.

5. *Genom inloggning ska administratörer kunna administrera medlemmar och försäljning, se rapporter osv.*

Rättigheter delas ut av redan priviligerade medlemmar till övriga efter behov. Vanliga medlemmar har väldigt begränsad access till databasen, för att förbättra säkerheten. Detta är i linje med vad Welling och Thomson (2005) skriver om säkerhet och olika databasanvändare.

6. *Man ska kunna se hur mycket pengar varje medlem erhållit i sponsring från klubben.*

Enklaste sättet att se detta är att söka upp medlemmen i registret och läsa informationen.

7. *Man ska kunna skapa och ladda ner backup-kopior av databasen från webbgränssnittet.*

Denna funktionalitet uppnåddes genom att integrera en backup-lösning (PHPMyBackupPro) med öppen källkod i systemet. Denna lösning har stöd för backup via mail, komprimering av backupfiler, schemaläggning av backup osv.

8. *Säkerhetsåtgärder som kryptering av medlemmars lösenord måste vidtas.*

För att göra systemet ännu säkrare krypteras inte bara lösenordet, utan andra variabler som är unika per användare med och sparas tillsammans.

6 Slutsats och diskussion

Problemställningarna från *1.2 Syfte och mål* besvaras först. Därefter följer reflektioner om projektet och uppsatsen. Avslutningsvis finns förslag på vidareutveckling av projektet och ytterligare vetenskapligt arbete.

6.1 Slutsats

Syftet med detta arbete var att undersöka utvecklingen av ett Internetbaserat ekonomiprogram för användning av utvalda i en liten organisation samt utreda vilka metoder och verktyg som är lämpligast för uppgiften utifrån de krav som finns.

Att göra en enkel variant av ekonomiprogram och anpassa det för webben innebar många utmaningar som ingen verkar ha dokumenterat innan. Både ekonomiprogram och webb-baserade butiker är ofta komplexa system, och därför är uppgiften att kombinera de två och samtidigt göra det enklare inte lätt. Jag kan dra några slutsatser utifrån utvecklingen av ett system som detta:

1. Planera och avgränsa rejält. Systemet kommer inte kunna ersätta vare sig webbshop eller ekonomiprogram till fullo, men organisationens krav är det viktiga.

2. Försök använda befintliga lösningar så långt som möjligt. Ibland är det värt att ändra hur någonting ska fungera för att spara tid.

3. Var beredd på att prioritera. Säkerhet, prestanda och användbarhet tävlar om utvecklingstiden och i de flesta fall är vissa aspekter viktigare än andra.

Genom att besvara de akademiska frågorna, uppfylls även resten av syftet med arbetet.

1. Hur uppnås tillräcklig prestanda, säkerhet och användbarhet för ett system som detta?

För att veta vilka val som behöver göras är första och andra fasen i OOS/UML-modellen mycket viktiga, och då i synnerhet kravspecifikation och databasdesign. Under utvecklingsprocessens gång fick fler och mer omfattande avgränsningar göras för att inte misslyckas med att uppfylla de grundläggande kraven från organisationen. Vissa krav prioriterades också ner, eftersom de inte var strikt nödvändiga för att uppnå tillräcklig användbarhet.

Prestandan i några webbprogrammeringsspråk undersöktes och PHP visade sig vara snabbt nog. Både säkerhet och användbarhet beror till stor del på hur många och vilka användarna av systemet kommer vara, och anpassningar har gjorts därefter. Welling och Thomson (2005) skriver att man kan se användbarhet, prestanda, kostnad och säkerhet som olika intressen som tävlar om samma resurser. I detta projekt har prestanda och kostnad varit viktiga poster, medan säkerhet och användbarhet kunnat prioriteras ned något och anpassats till typen och mängden av användare.

2. Vad i utvecklingsprocessen skapar svårast problem och var är tidsåtgången som störst?

Som metod valdes en anpassad variant av OOS / UML, eftersom bara undertecknad arbetat med projektet och en flexibel modell antogs vara bäst. Hade en mindre flexibel modell valts istället, kunde tidsåtgången för att uppfylla varje steg i processen blivit opraktisk.

De mest tidskrävande stegen i utvecklingsprocessen visade sig vara planering och databasdesign, följt av felsökning i den mer avancerade AJAX-koden. Planering sker främst under fas ett, och databasdesign främst under fas två enligt OOS/UML-modellen (prototyputveckling). Trots att planeringen tilldelats veckor fick databasen ändras flera gånger under arbetsprocessens gång. En orsak kan ha varit svårigheterna i att kombinera ett system för en webbplats med inloggning och produktförsäljning med ett traditionellt ekonomisystem, samtidigt som komplexiteten inte fick bli för hög. Avgränsningarna fick också de ändras och stärkas eftersom projektets karaktär gjorde att arbetsbördan kunde blivit i princip hur stor som helst.

3. Varför är de valda verktygen och metoderna att föredra framför alternativen i det här fallet?

De viktigaste kraven var att systemet skulle vara snabbarbetat och kostnaderna låga. Låga kostnader ledde automatiskt in på att använda kostnadsfria verktyg, vilket de flesta med öppen källkod är. Hade projektet varit kommersiellt och arbetsinsatsen kostat pengar, hade detta val inte varit lika självklart. Användandet av fri mjukvara med öppen källkod har i detta arbete sannolikt inte minskat utvecklingstiden, eftersom alternativen till Microsofts utvecklingsmiljö och annan mjukvara inte ingick i utbildningen. Kostnaderna för uppdragsgivaren blir dock små, eftersom inga pengar behöver spenderas på operativsystem och serverprogramvara.

6.2 Diskussion och förslag på uppföljning

Tidsåtgången och arbetsbördan som detta projekt har krävt var svåra att förutse. Hade projektet startat i dag, med den kunskap jag har nu, skulle jag jobbat hårdare på avgränsningarna från början och möjligtvis gjort en seriösare utvärdering av de kommersiella alternativen som Microsofts .net-plattform. Jag känner ändå att detta varit mycket givande och nyfikenheten väckts på de nya initiativen bland projekt med öppen källkod.

Förutom de givna utvecklingsmöjligheterna som öppnar sig om avgränsningarna lättas på, skulle ett system som detta kunna anpassas till andra verksamheter. Mindre företag kan mycket väl ha nytta av ett sådant här projekt, som inte kräver mycket kunskap om vare sig ekonomi eller datorer för användning.

Ur ett akademiskt perspektiv skulle det vara intressant med en uppföljande studie av dels hur systemet används, och dels vilka styrkor och svagheter som det visar sig ha på längre sikt. Jämförelser med befintliga kommersiella alternativ bör utföras. Även den fortsatta utvecklingen av alternativen på AJAX-området är intressant. Det finns i dag en snabb utveckling av webbapplikationsservrar som är skräddarsydda för att köra avancerade användarinteragerande program på webben. Vart leder denna utvecklingen?

7 Referenser

Apelkrans, M. och Åbom, C. (2001). *OOS/UML: En objektorienterad systemutvecklingsmodell för processorienterad affärsutveckling.* Studentlitteratur, Lund

Bessen, J. (2005). *Open Source Software: Free Provision of Complex Public Goods.* http://www.researchoninnovation.org/opensrc.pdf (Hämtad 2009-01-27)

Bokföringstips.se (2009). *Kontantmetoden.* http://www.bokforingstips.se/kontantmetoden.htm (Hämtad 2009-03-13)

Connolly, T. och Begg, C. (2005). *Database systems: A practical approach to design, implementation and management.* Essex: Pearson Education Limited

Cullen, K. (2002). PHP: An open source solution for web programming and dynamic content. *Information technology and libraries*, 21 (3): 116-120

Jacobsen, Dag I. (2002). *Vad, hur och varför: Om metodval i företagsekonomi och andra samhällsvetenskapliga ämnen*. Studentlitteratur. Lund

Locke, J. (2004). *OPEN SOURCE SOLUTIONS for Small Business Problems.* Massachusetts: Charles River Media, Inc.

Lakhani och von Hippel, (2003). How open source software works: "free" user-to-user assistance. *Research Policy*, 32 (6): 923-943

Olson, S.D. (2007). *AJAX on Java.* Sebastopol, CA: O'Reilly Media, Inc.

Perry, J. och Schneider, G. (2003). *Building Accounting Systems Using Access 2002.* Ohio: South-Western

Svenning, Conny (2003). *Metodboken*, 5:e upplagan. Lorentz förlag. Eslöv.

Unger, R. (2005). *Eclipse vs. NetBeans *yawn** http://weblogs.java.net/blog/richunger/archive/2005/07/eclipse_vs_netb.html (Hämtad 2009-01-27)

Welling, L. och Thomson, L. (2005). *PHP and MySQL Web Development.* Indianapolis: Sams Publishing

8 Sökord