INGENJÖRSHÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

Webbshop för Tyger & Ting

Tommy Svenningsson

Jonas Sanell

EXAMENSARBETE 2006
DATATEKNIK

INGENJÖRSHÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

Webbshop för Tyger & Ting

Webbshop for Tyger & Ting

Tommy Svenningsson

Jonas Sanell

Detta examensarbete är utfört vid Ingenjörshögskolan i Jönköping inom ämnesområdet datateknik. Arbetet är ett led i den treåriga högskoleingenjörsutbildningen. Författarna svarar själva för framförda åsikter, slutsatser och resultat.

Handledare: Lars-Olof Petersson

Omfattning: 10 poäng (C-nivå)

Datum: 2006-06-15

Arkiveringsnummer:

Postadress:
Box 1026
551 11 Jönköping

Besöksadress:
Gjuterigatan 5

Telefon:
036-10 10 00 (vx)

Abstract

This report describes how we made a dynamic website combined with statical web pages and a web shop for the Swedish company Tyger & Ting.

The employer Tyger & Ting wish to run a working web shop on their site but they want to keep their old system of updating pages with their old application from Intellyweb[1], which creates statical HTML pages.

The goal of this exam paper has been to learn to create a functional web shop with user friendly layout and appealing design and also to take orders and fulfill wishes from a real employer.

The report is describing how Intellyweb's application and its statical web pages has been implemented on a dynamic web page. The report does also describe how this site has been build and how the problems with the coding have been solved. The coding has been made with HTML, PHP and MySQL. CSS has also been used. The report will describe how management and storage of data in databases has been handled.

Apart from the demands the employer has given us, a complete revamp of Tyger & Ting's design has been made and the result of the exam paper is now a complete and working web site with additional web shop that supports static web pages. The website contains 20 dynamic and static pages coded in about 1900 rows of PHP and HTML. A database with five tables of varying size has been made to contain pictures and information about products, users and orders.

As soon as all the products has been put into the database the web shop will be found here: http://www.tygeroting.com

[1] http://www.intellyweb.com

Sammanfattning

Denna rapport beskriver hur vi, på uppdrag av Tyger & Ting byggt en kombinerad dynamisk & statisk webbsida med tillhörande webbshop.

Uppdragsgivaren Tyger & Ting önskar driva en webbshop på sin sida samtidigt som de vill kunna uppdatera övrigt innehåll mha. programvara, från Intellyweb, som producerar statiska sidor i HTML.

Syftet med examensarbetet har hela tiden varit att lära sig skapa en fungerande webbshop med användarvänlig layout och tilltalande design samt att ta order och uppfylla önskemål från en riktig uppdragsgivare.

Rapporten beskriver hur Intellywebs programs statiska sidor implementerades på en dynamisk webbsida. Rapporten beskriver också hur denna sida byggts upp och hur problem lösts med all kodning. Programmeringen har skett i HTML, PHP och MySQL. CSS har också använts. Här beskrivs även hur problem med hantering och lagring av data i databaser har hanterats.

Förutom de krav uppdragsgivaren ställt har en komplett designförändrig gjorts och resultatet av examensarbetet är en numera fungerande webbsida med tillhörande webbshop som stöder statiska sidor. Webbsidan består av 20 dynamiska och statiska sidor på ungefär1900 rader, kodade i PHP och HTML. En databas med fem tabeller av varierande storlek har skapats för att rymma bilder och information om produkter, användare och beställningar.

Så snart alla produkter lagts in i databasen kommer webbshoppen finnas tillgänglig på http://www.tygeroting.com

Nyckelord

Webbdesign

Webbshop

PHP - *Hypertext Preprocessor*

HTML - *HyperText Markup Language*

MySQL - *Structured Query Language*

CSS - *Cascading Style Sheets*

Innehållsförteckning

Figurförteckning

I Inledning

Det svenska inredningsföretaget Tyger & Ting hade länge velat ha en hemsida med en egen webbshop där de kunde ta hand om alla postorderkunder på ett enkelt vis samt visa nyheter och inredningstips. Efter att vi skickat E-post till ett antal företag som tordes vara intresserade av en webbshop blev Tyger & Ting de lyckliga utvalda. Med vår databas, programmering och webbdesignutbildning i ryggen hade vi en mall på hur vi tänkt bygga upp vårt arbete som vi visade för Tyger & Tings webbansvarig, Catharina Svenningsson. Hon hade sedan några invändningar och idéer som vi också tog hänsyn till innan vi gjorde klart den färdiga modellen. Vi fick en del problem med företagets befintliga webbutrustning som inte riktigt matchade våra planer, och för detta krävdes avsevärd undersökning för att lyckas lösa.

1.1 Bakgrund

Eftersom vi redan hade viss kunskap om PHP och MySQL, och ett behov av webbshop fanns hos beställaren, blev valet av uppgift lätt. Webbdesign har varit frekvent återkommande under hela vår utbildning och på senare tid har även en del SQL och databaskunskap legat på schemat. Att bygga vidare och sammanbinda dessa kunskaper känns nyttigt och intressant för kommande projekt och arbeten.

Uppgiften om att bygga en förhållandevis ganska avancerad webbsida som skall kunna samarbeta med administratörens mindre avancerade metoder att uppdatera sidan har sin grund i att administratören på Tyger & Ting nyligen köpt in ett program som är lätt att använda och lärt sig göra sidor i detta. Dessvärre utesluter det programmets mallar och guider möjligheter att skapa dynamiska sidor som en webbshop kräver. Därför tänkte vi försöka göra en dynamisk webbsida med webbshop och tillhörande funktioner som även till stor del går att uppdatera från det programmet.

1.2 Syfte och mål

Dessa mål har diskuterats fram med nuvarande webbansvarig på Tyger & Ting - Catharina Svenningsson.

1.2.1 Kravspecifikation

Den stora delen av projektet handlar om att webbshopen skall byggas. Detta är något som företaget önskat ha en längre tid men det har aldrig blivit av. Beställningarna skall skickas via postorder och betalas med postförskott vilket besparar oss och dem krångel med kreditkortstransaktioner. Webbshopen skall innehålla följande funktioner:

- Kundregistrering
  - En sida där kunden väljer ett användarnamn och lösenord samt uppger sitt namn, adress, telefonnummer och e-post. En sådan inloggning är nödvändig för att få tillgång till en kundvagn som i sin tur är nödvändig för att kunna beställa varor. Man skall givetvis kunna bläddra i sortimentet även om man inte är registrerad och inloggad, då är dock ”köp” -funktionen dold.

- Inloggning
  - Har man skapat en användarprofil ska man sedan oavsett var på sidan man befinner sig kunna logga in via ett formulär längst uppe på sidan och få tillgång till sin kundvagn. Här kan även administratören logga in för att uppdatera sortimentet, lista kunder och se ordrar samt orderhistorik.

- Användarredigering
  - Här kan både administratör och kund vid behov kunna ändra sin användarinformation som angivits vid kundregistreringen. Administratören kan även ändra andra användares information vid behov samt uppgradera vanliga användare till administratörer.

- Sortiment
  - Sortimentet är sektionen där kunden kan bläddra igenom utbudet av produkter som finns till försäljning på webben. På första sidan listas de 5 senast ändrade produkterna och där finns även en lista där man kan välja vilken produkttyp man är ute efter och lista samtliga produkter i denna kategori. Produkterna visas med en liten bild samt dess namn, artikelnummer och pris. Klickar man på produktens bild får man fram en större bild på denna i ett popup-fönster. Är man inloggad finns det även en ”köp” -knapp man kan trycka på för att lägga varan i sin kundvagn.

- Kundvagn
  - Är en liten funktion positionerad längst ner på sidan som ersätter inloggningen när man loggat in. Här läggs varorna kunden valt att köpa. Rensar man inte kundvagnen eller tar den till kassan ligger varorna kvar tills nästa gång man loggar in på sidan. Varje användare har endast en kundvagn, varför ha flera? I kundvagnen ska man kunna ta bort enskilda varor samt funktionen att rensa hela kundvagnen. Här finns även en länk till kassan.

- Kassa
  - När kunden valt alla varor och vill beställa dessa går man till kassan. Där listas alla produkter man valt och man får besked om vad allt kommer att kosta. Det finns även en meddelande ruta där man kan skriva en kommentar på ordern eller ange en leveransadress om denna skiljer sig från den man uppgett i användarinformationen. När detta är bekräftat sparas ordern i databasen som administratören sedan kan se. Användarens IP adress loggas också så en person kan bindas till ordern.

- Orderbehandlare
  - Den här sidan är endast tillgänglig då man loggat in som administratör. Här ser man vilka ordrar som lagts och kan markera dessa som behandlade när de skickats till kunden. De ligger dock kvar i databasen och kan visas via funktionen orderhistorik

- Orderhistorik
  - Här listas alla gamla ordrar. Denna sida är endast tillgänglig för administratören och om denna vill kan även specifika ordrar tas bort för gott.

- Kundlista
  - Även denna funktion är endast för administratören och listar alla kunder som registrerat sig. Klickar man på en användare kan man få fram samtliga ordrar lagda av denna. Man kan även ändra användarens information eller ta bort oseriösa användare.

- Sortimentbehandlare
  - Här lägger administratören in varor i sortimentet. Förutom information om varans namn, artikelnummer, pris, typ och eventuellt vikt kan man även ladda upp en bild på produkten. Man kan även redigera/radera befintliga produkter.

Vidare behöver även vissa justeringar ske på den befintliga sidan för att göra den mer kompatibel med webbshopen och dess funktioner.

1.2.2 Problembeskrivning

Ett problem är att administratören införskaffat ett program hon lärt sig som låter användaren skapa statiska webbsidor. Deras nuvarande webbsida är gjord i det programmet och hon vill fortfarande kunna uppdatera sidorna mha. det. Vi måste därför komma på ett sätt att både kunna ha en dynamisk sida med alla funktioner webbshopen kräver för att fungera bra samtidigt som administratören ska kunna fortsätta uppdatera övriga sidor som innan.

En enkel lösning kunde vara att ha webbshopen på en egen sida skild från deras vanliga statiska miljö, vi känner dock att det inte skulle vara lika användarvänligt som om man hade vävt ihop allting på samma sida så att kunden kan logga in, registrera sig, se sin varukorg etc. vart man än befinner sig på sidan. Vi kommer därför att använda includes för att från det dynamiska gränssnittet anropa de statiska sidorna. Sidan blir på så sätt mer sammanhängande. Vissa modifieringar behövs då göras på de statiska sidorna men när det är gjort bedömer vi att det bör fungera bra. Målet är över huvud taget att sidan skall vara lätt att administrera även för personer med mindre vana av de olika moment detta innebär.

Ett problematiskt moment vi resonerat mycket om är även vad man ska lagra i databasen och hur detta ska göras, vilka tabeller som skall finnas osv. Samt hur behandlingen av bilder till produkterna ska göras för att det ska vara så lätt som möjligt för administratören att få in dessa på sidan.

Under projektets genomförande förutspås vissa problem uppstå. De större nyss nämnda problemen och vissa mindre problem som tas upp med dess lösning löpande i texten under genomförandet (kap. 3).

1.3 Avgränsningar

1.3.1 Endast postorder

Det ligger varken i vårt eller företagets intresse att i dagsläget erbjuda tjänster såsom Internetbankbetalning, kreditkortsbetalning eller liknande, och ordrar kan endast betalas med postförskott.

1.3.2 Begränsad säkerhet

När man loggar in eller uppger annan information upprättas ingen SSL-anslutning eller liknande säkerhetsrutin. Informationen användaren skrivit in kollas helt enkelt mot databasen och stämmer denna, loggas man in. Detta anser vi vara säkert nog då sidan inte hanterar kritisk information såsom kontokortsnummer. Vidare kontrolleras inte någon information kunden uppger, den får helt enkelt antas vara riktig. Skulle informationen vara av typen ”y54gh4r8@£€€” ser ju även administratören att detta t.ex. inte är ett giltigt postnummer och kan då ta bort den användaren.

1.3.3 Företagets uppgifter

Innan sidan kan anses värd att lägga upp offentligt har även företaget en del jobb att göra på sidan. Det gäller i huvudsak att ta bilder på produkterna och lägga upp produkter i webbshopen samt att uppdatera de statiska sidorna företaget tidigare hade. Anledningen till att inte vi gjort även dessa saker är dels för att administratören genom att göra detta lär sig hur det fungerar men även för att detta hade tagit lång tid för oss att göra. Det blir också smidigare att lösa det såhär eftersom de vet alla priser och övrig information om produkterna och kan ta bilder på dem direkt i affären.

1.4 Disposition

Vi kommer först att gå igenom den teoretiska bakgrunden som tar upp lite olika metoder som kan anses användbara i projektet. Motivationer till varför vissa metoder använts och andra inte står också här.

Sedan följer en kort beskrivning om programvaran som nyttjats och lite kort information om dessa. Därefter följer en beskrivning på hur själva användargränssnittet är uppbyggt och hur sidan fungerar i grunden och varför vi valt att göra som vi gjort. Här finns även lösningen på problemet med att kunna kombinera statiska sidor med det dynamiska gränssnittet.

När sidans grund är beskriven kommer vi gå in mer på hur vi byggt upp sidans olika funktioner som redan beskrivits kort i Syfte och Mål (kap. 1.2). Det är i dessa funktioner de flesta av problemen stötts på och lösts så vi kommer även ta upp vilka dessa problem var och hur de lösts. Allt eftersom databasen används i funktionerna kommer det även beskrivas hur vi resonerat och löst de problemen.

Exempel på hur koden för vissa funktioner ser ut förekommer frekvent. Vi har använt oss av Microsoft Words kommentarfunktion för att kommentera koden då vi tyckte det var ett bra sätt att peka ut just vilken kod kommentaren gäller. När en metod eller kod som inte tidigare använts tas upp förklaras denna lite mer i detalj för att sedan bara nämnas om den används vid senare tillfälle.

I resultatet presenteras en kort överblick på vad vi allt som allt åstadkommit.

Därpå följer slutligen diskussionen där vi jämför resultatet med vad vi ville uppnå i kravspecifikationen (kap. 1.2.1). Samt hur arbetet med sidan kommer att fortsätta när administratören på Tyger & Ting tar över.

2 Teoretisk bakgrund

På gymnasiet gick vi båda Tekniska programmet där vi fick lära oss ganska simpel och grundlig html kodning. Personligt intresse gjorde att vi sökte information på egen hand om hur man skapar intressanta och grafiskt tilltalande hemsidor. På gymnasiet valde vi även tillvalskurser inom programmering och databashantering där vi fick lära oss att använda databaser och kombinera dessa med hemsidor. Efter ett antal hobbysidor med användarregistrering, forum och filarkiv gjorde vi ett projekt inom ämnet. Vi skapade ett intranät där alla skolans teknister fick användarnamn och där vi hade tävlingar och diskussionsgrupper. Ett försök att även viga in lärare i systemet gjordes men de hade inte tillräckligt med intresse för att göra det användbart. Tanken var att vi skulle kunna kolla läxor osv. på sidan.

På Ingenjörshögskolan i Jönköping går vi båda linjen informationsteknik och har således gått mer avancerade kurser inom databashantering. Förhoppningsvis ska vi ha god nytta av dessa kunskaper när vi stöter på problem inom arbetet med webbshopen.

Vissa av metoderna som använts kommer att, i denna del av rapporten, kommenteras i texten med exempel. Senare i rapporten kommer exempel att kommenteras rad för rad.

2.1 HTML

Hyper Text Markup Language, som HTML står för, är ett programspråk som möjliggör webbsidekonstruering. Det är mycket enkelt att komma igång med och man behöver inte mycket kunskap för att göra sin egen hemsida. För att kunna lägga ut sin hemsida på Internet behöver man ha en webbserver att ladda upp den till men förutom det är en texteditor det enda som krävs för att kunna sätta igång och skriva. Vill man ha en mer avancerad hemsida är man tvungen att ha lite mer kunskap om hur språket fungerar. Det finns dock program som gör allt betydligt enklare där man inte behöver kunna någon kod för att lägga in bilder, bygga tabeller osv. HTML-språket är uppbyggt av ”taggar” som beskriver hur text och objekt ska formateras. Webbrowsern (Firefox, Internet Explorer etc.) laddar hem din sida från webbservern, läser taggarna och bygger upp din sida i browserfönstret efter hur du kodat den. En ”tagg” är ett ord eller alias, som definierats i HTML-språket, som står inom tecknen ”<” och ”>”. Allt som står mellan den taggen och dess ”sluttagg” kommer skrivas ut i webbrowsern enligt taggens definition. En sluttagg är identisk till starttaggen med enda skillnaden att ett ”/” läggs till efter < och före själva ordet/aliaset.

```
<html>
<body>
```

Html taggen måste alltid stå i början av koden och den är den sista taggen att stängas. Man kan skriva taggar inom andra taggar så länge som man stänger dem i rätt ordning.

```
</body>
</html>
```

Det finns oändliga möjligheter att utforma sin hemsida originellt mha. HTML och utan det hade vi inte kunnat göra någon webbshop alls[2].

2.2 CGI

Protokollet CGI (Common Gateway Interface) används för att kunna använda argument i hemsidor, alltså ett bra sätt att göra sidor interaktiva. CGI var ett av de första sätten att skapa dynamiska hemsidor på, men nu finns det betydligt fler sätt som både är enklare att lära sig och smidigare att använda. CGI är inget eget programmeringsspråk utan bara en sorts mall. CGI standaren (interfacet) har en del regler för hur inmatning och utmatning sker, förutom det kan vilket programmeringsspråk som helst användas[3].

[2] http://computer.howstuffworks.com/web-page.htm
[3] http://computer.howstuffworks.com/cgi.htm

2.3 IIS

Internet Information Services är Microsofts produktion. Det är ett paket tjänster skapade för servrar som kör Microsoft Windows. Stort sett är det världens näst mest använda webbserver, efter Apache. IIS stödjer bl.a. ASP vilket gör den mycket populär[4] [5].

2.4 ASP

ASP står för Active Server Pages och är precis som IIS en skapelse av Microsoft. ASP är precis som PHP ett server-side skriptspråk och används för att kunna skapa dynamiska hemsidor. Inom ASP taggarna (<% och %>) skrivs programkod som exekveras på webbservern. Webbservern genererar sedan en webbsida efter koden som skickas tillbaka till webbläsaren. Webbläsaren ser inte skillnad på vad som var ASP från början utan läser allt som om det vore i HTML. Programkoden kan skrivas i flera olika programspråk vanligast är dock VBScript, en variant av Microsoft Visual Basic[6].

2.5 Access

Access är Microsofts databashanterare från sitt Officepaket. Access är relativt lätt att lära sig och enkelt att hantera. Det har det välkända Windows utseendet och användare som är vana vid Windows design kommer känna igen sig. Det är också smidigt kompatibelt med andra Microsoft Office produkter. Access har otaliga hjälpguider vilket gör programmet utmärkt för nybörjare[7].

[4] http://news.netcraft.com/archives/web_server_survey.html
[5] http://www.microsoft.com/windowsserver2003/iis/default.mspx
[6] http://www.webwizguide.info/asp/tutorials/what_is_asp.asp
[7] http://databases.about.com/od/access/l/aaaccess1.htm

2.6 PHP

Vi har använt oss av PHP under uppbyggnaden av webbshopen. PHP är ett server-side skriptspråk, vilket innebär att koden innanför PHP taggarna exekveras på webbservern istället för i användarens webbläsare, och kan användas tillsammans med HTML. PHP har även stöd för ett antal databassystem som T.ex. MySQL som vi också har använt. PHP anses enkelt och effektivt och man kan utan problem hitta oändliga spaltmeter information och hjälp på Internet om man skulle köra fast. PHP har utmärkt kompabilitet med Apache webbserver och det är ett stort plus eftersom webbservern Tyger & Tings sida ligger på kör Linux och Apache. All PHP kodning skrivs innanför startttecknet (<?) och slutttecknet (?>)[8]. Ett litet exempel på hur phpkodning kan se ut:

```
<?php
if ($_GET['sida'] == '')
{
           include "index_forstasidan.php";
}

elseif ($_GET['sida'] == 'gardiner')
{
           include "index_gardiner.php";
}
... osv
?>
```

Om det inte skickats med någon variabel kommer index_forstasidan.php att visas. Om andra variabler skickas med kommer andra sidor visats beroende på vilken ’sida’ som angetts för den variabeln.

[8] http://www.phpportalen.net/

2.7 CSS

Med CSS (Cascading Style Sheets) kan man kontrollera flera sidors layout och stil från ett ställe. T.ex. att rubriker ska visas i fetstil och att brödtext ska använda typsnittet Verdana. Använder man inte CSS måste man ange återgivning för varje enskild sida vilket känns onödigt tidskrävande [9].

Så här ser vår body tag ut:

```
body {
     background-color: #CCCCCC;
     background-image: url('../images/skavi_bg1.gif');
     color: #666666;
     font-family: arial, verdana, helvetica, sans-serif;
     font-size: 11px;
     margin: 0;
     padding: 10px 0px 0px 0px;
}
```

2.8 MySQL

Vi valde MySQL[10] som databashanterare eftersom viss kunskap inom detta redan fanns. MySQL har fri källkod dvs. det är gratis för allmänheten att ladda hem och använda. MySQL kan också kallas server-side i vissa fall. T.ex. när man vill att något ska bearbetas på databasservern utan att behöva skickas fram och tillbaka mellan server och klient. I vårt fall används det vid lagring av användare, kunder, ordrar och för produkterna som ska säljas i webbshopen osv. Här nedan kan en av våra SQLfrågor beskådas:

```
$sql = "insert into products(pname, pprice, pstock, ptype, pdesc)
values(
'". $_POST["name"] . "',
'". $_POST["price"] . "',
'". $_POST["stock"] . "',
'". $_POST["type"] . "',
'". $_POST["desc"] . "')";

mysql_query($sql) or die(mysql_error());
```

Post hämtar information som skrivits in I ett formulär. Informationen hämtas från respektive fält och läggs in i databasen under ”samma” namn. Alltså det som skrivs in i fältet ”name” hamnar i kolumnen ”pname” i databasen. Mysql_query($sql) gör att sqlfrågan som ligger i $sql körs och informationen hamnar i databasen. Uppstår ett fel visas ett felmeddelande.

[9] http://www.westciv.com/style_master/academy/css_tutorial/
[10] http://www.mysql.com/

2.9 Cookies

En cookie är en liten textfil som lagras på webbservern. Textfilen innehåller i de flesta fall ett idnummer som används för att veta att du varit på sidan förr. När man besöker en sida, t.ex. www.hj.se, som använder cookies kommer man få ett idnummer som lagras i cookiefilen på datorn. Information som man skrivit in på hemsidan lagras i databasen på webbservern tillsammans med idnumret. Nästa gång man besöker sidan kommer webbrowsern skicka tillbaka cookien, som innehåller ditt idnummer, till webbservern och således vet webbservern att du varit på sidan innan. I vårt fall kan man välja automatisk inloggning när man loggar in och tack vare cookien behöver du inte skriva in användarnamn och lösenord nästa gång du vill handla eller bara titta om det finns några intressanta nyheter eftersom inloggningsuppgifterna redan finns lagrade. En cookie är inte farlig och kan inte göra någon skada på datorn den lagras på eftersom det bara är ett dokument med några rader text i. Alla antaganden om att en cookie skulle vara något sorts program som sparar information det hittar på datorn, som sedan kan skickas till cookiens ägare är helt **falskt**. Cookien från www.hj.se kan se ut som följande:

```
Stylecss
null
www.hj.se/
1600
319161344
29852286
2587837440
29778860
*
```

Som synes har flera idnummer lagrats och det beror på att hj.se har sparat flera sektioner. [11]

2.10 Javascript

Javascript är ett skriptspråk som har vissa likheter med C och C++. För att använda Javascript krävs att det ligger inbyggt i andra system eftersom det, precis som C, inte innehåller några input/output konstruktioner. Javascript är mest känt för sin användning i oändligt många hemsidor där det skapar vackra rörliga miljöer. Det används mest för att skriva och integrera funktioner i webbsidor som inte kan skrivas i vanlig HTML-kod. Dessa funktioner kan t.ex. vara spel eller helt nya interface för sidor som poppar upp genom ett simpelt klick med musen. Microsoft har skapat ett kompatibelt skriptspråk, med namnet Jscript, efter att ha sett hur populärt javascriptet blivit. Mer om JavaScript kan läsas på Sun Microsystems hemsida[12]

[11] http://computer.howstuffworks.com/question82.htm
[12] http://java.sun.com/products/jsp/

2.11 Apache

Som webbserver kör Tyger & Tings webbhotell (EPS Domains kan läsas mer om i Genomförandet (kap. 3.1.4)) det väldigt populära Apache som är ett open-source projekt som kan köras på de flesta operativsystemen (2003/XP/2000/NT/9x, Netware 5.x eller högre, OS/2, och de flesta Unix versionerna). Med open-source menas att det är fri programvara som vem som helst har rätt att använda utan att behöva betala för den. På Apaches hemsida går det att läsa att Apache har varit den populäraste webbservern på Internet sedan april 1996 och att 2005 års ”Netcraft Web Server” undersökning visade att mer än 70 % av Internets hemsidor använder Apache.[13]

[13] http://httpd.apache.org/

3 Genomförande

3.1 Programmeringsmiljö

Här följer en liten genomgång och lite information om de programvaror vi har använt under arbetet med exjobbet. Vi har också lagt in Referenser för var mer information kan hittas.

3.1.1 Macromedia Dreamweaver 8

Kodningen sker enbart i Macromedias Dreamweaver 8. Detta program innehåller en hel del stöd för att utveckla webbsidor. De enda vi använt oss av är dock färgkodningen vilket innebär att det visar olika element av koden i olika färger t.ex. har variabler en färg, vanlig statisk text en annan o.s.v. Det innehåller även en extremt smidig funktion som gör att så fort man sparar dokumentet man arbetat i på datorn så laddar det upp en kopia av detta på webbservern så att det bara är att gå in och testa direkt. Det finns som sagt även ett flertal andra smidiga funktioner som hjälper till med databasanslutningar och annat men det känns som man har mer koll på sådant om man skriver koden själv.

På Adobes hemsida finns information om denna och andra av deras produkter.[14]

3.1.2 Adobe Photoshop CS

En bildeditor har även krävts för att bland annat rita deras logo och andra smådetaljer på sidan. I det fallet användes det välkända programmet Adobe Photoshop eftersom det har utmärkta verktyg för att utforma grafiskt tilltalande bilder. Photoshop har använts frekvent i tidigare projekt genom åren och därför känner vi till det väl[15].

[14] http://www.adobe.com/products/dreamweaver/
[15] http://www.adobe.com/products/photoshop/

3.1.3 Intellyweb

Intellyweb är programmet Tyger & Ting använde sig av för att skapa sin hemsida, och önskar fortsätta göra så, och således anledningen för oss att göra webbshopen kompatibel med statiska sidor. Intellyweb är ett hemsideprogram för nybörjare som gör det enkelt att skapa egna hemsidor. Man kan lätt fylla sidan med bilder, text och länkar mha. mallar som finns att välja i programmet. För att lägga upp sin hemsida på Internet behöver man bara klicka på en knapp så ordnar programmets inbyggda ftp-klient så att sidan laddas upp automatiskt på ditt webbhotell. [16]

3.1.4 Testserver

Under utvecklingen av sidan har man mycket stort behov av att snabbt kunna testa om koden fungerar som den ska. Detta kräver att man har en webbserver som stöder språken man programmerar i. Vi löste det genom att använda Tyger & Tings webbhotell eftersom vi då vet att sidan kommer fungera med den serverns inställningar samt att vi kan nyttja serverns databas redan i utvecklingsstadiet. Eftersom deras befintliga sida endast använder sig av .htm filer och den nya använder .php kan vi lugnt lägga upp dessa filer utan att det stör/skriver över den befintliga sidan. Webbhotellet är EPS Domains [17] och sidan ligger på en Linux RedHat[18] server som stöder bland annat PHP, CGI, SSI, WAP och MySQL. I vårt fall nyttjar vi PHP, SSI samt MySQL. På deras sida kan man läsa att:

”Alla våra servrar består att komponenter av högsta klass, från ledande tillverkare som Intel, Seagate och Kingston för att kunna leverera en stabil servergrund. Som operativsystem använder vi uteslutande RedHat Linux samt Microsoft Windows 2003 Server, båda välkända för att vara snabba och stabila. Alla system hålls regelbundet uppdaterade med de senaste säkerhetsfixarna samt underhålls med ny hårdvara för att minimera risken för fel.”[19]

Med webbhotellet får man även grafisk statistik, e-post konton och annat trevligt vi dock inte kommer att ta upp i den här rapporten då dessa funktioner inte använts alls.

[16] http://www.intellyweb.com
[17] http://www.epsab.se
[18] http://www.redhat.com/
[19] http://www.epsab.se/om.php

3.2 Sidans struktur

Vi har använt oss av två olika metoder beroende på om det är webbshopen och funktionerna kring den som används eller om det är Tyger & Tings statiska sidor. I fallet med webbshopen och dess funktioner så skrivs sidans specifika kod i själva sidan. Som kod läggs även dessa rader till överst på sidan:

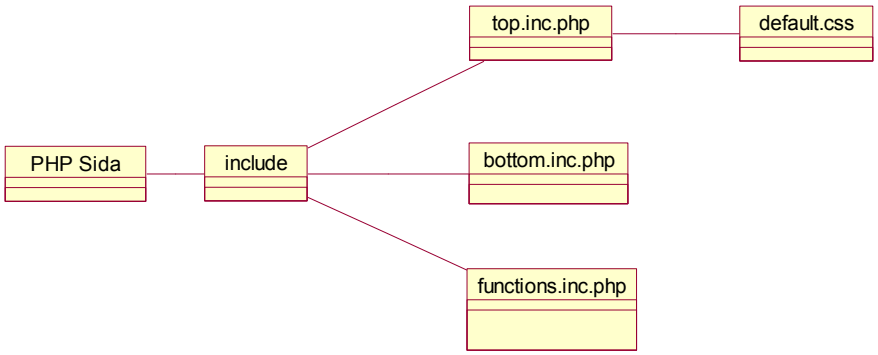
```
require "includes/functions.inc.php";
include  "includes/top.inc.php";
```

Comment [d1]: Inkluderar sidan functions.inc.php. Require ser till så att sidan bara inkluderas en gång.

Comment [d2]: Inkluderar sidan top.inc.php. Med include kan man anropa samma sida flera gånger till skillnad från require.

även raden `include "includes/bottom.inc.php";` läggs till fast längst ner i koden så denna sida utgör "foten" på sidan.

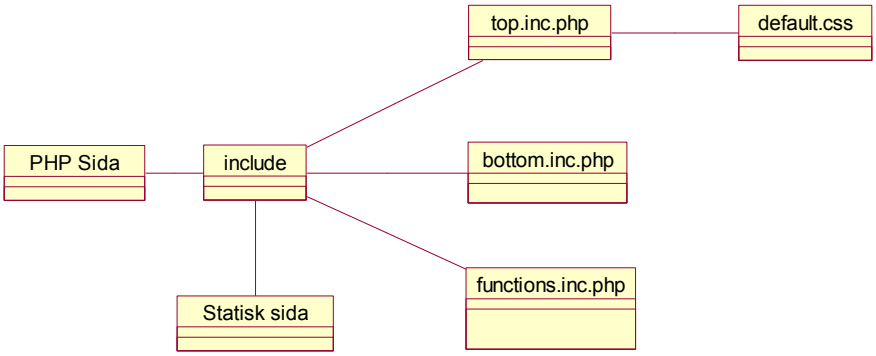
Dessa rader kod gör att de inkluderade sidornas innehåll finns med på PHP sidan som om koden stod på den sidan. Detta gör koden mer lättläst och sparar en hel del plats. Vad de olika "includesen" gör återkommer vi till längre fram i rapporten.



Figur 1. Förenklad bild över sidans uppbyggnad

Som synes i *Figur1* så anropar top.inc.php även en fil som heter default.css som är en "cascading style sheet" fil. I den finns information om hur exempelvis länkar, text och i stort sett allt annat ska se ut, vilken färg de ska ha, vilket typsnitt som skall användas etc. I vårt fall omfattar denna fil i stort sett allt som har med sidans utseende att göra.

Vi hade problem med att få sidan att fungera med de statiska sidorna som uppdragsgivaren insisterar på att behålla. Så om man surfar in på en av sidans statiska sidor har vi löst det enligt *Figur2*:



Figur 2. Förenklad bild av sidans uppbyggnad med hantering av statiska sidor

Skillnaden på detta och det tidigare diagrammet är den statiska sidan som kommit till. Själva PHP Sidan innehåller kod som gör att den även inkluderar den statiska sidans kod. Detta mha. koden nedan:

```
if ($_GET['sida'] == '')
{
    include "index_forstasidan.php";
}

elseif ($_GET['sida'] == 'hander')
{
    include "index_hander.php";
}

//Etc.
```

Comment [d3]: Om variabeln ”sida” inte innehåller någon data.

Comment [d4]: Inkluderar sidan ”index_forstasidan.php”.

Comment [d5]: Som ovan fast sidan ”index_hander.php” inkluderas om variabeln ”sida” innehåller ”hander”.

Förhoppningsvis har detta gett en tydlig bild av hur själva grunden av sidan är uppbyggd. Här följer lite mer detaljerade beskrivningar av vad de olika ”includesen” innehåller och vad de gör för nytta.

3.2.1 Top.inc.php - Inloggning & Meny

Figur 3. Sidans ”Huvud” med tillhörande logotyp, inloggning och meny

I sidans övre del (*Figur 3*) kan man logga in om man har ett användarnamn. Har man inte det och man besöker sidan för första gången kan man registrera sig via en länk till registreringssidan. Har man väl loggat in en gång kan man kryssa i en ruta så att man loggas in automatiskt nästa gång man kommer på besök. Filen innehåller också kod som skapar en ”Cookie” som möjliggör den automatiska inloggningen.

```
if (isset($_POST['autologin']))
{
        $expire = time() + (60*60*24*30);
        setcookie ("id", $userId, $expire);
        setcookie ("password", $_POST['password'], $expire);
}
else
{
        setcookie ("id", "", time()-20); //Skapar cookie
        setcookie ("password", "", time()-20);
}
```

Comment [d6]: Om autologin rutan har kryssats i körs koden under.

Comment [d7]: Ställer in hur länge cookien ska finnas kvar, detta anges i sekunder och i detta fallet stannar den en månad.

Comment [d8]: Skapar cookie med användarnamn & lösenord.

Comment [d9]: Är autologin inte ikryssat görs istället följande.

Comment [d10]: Skapar cookie.

Sidhuvudet fungerar som meny och innehåller således länkar till alla andra delar av sidan.

3.2.2 Bottom.inc.php – Sidfoten

I sidfoten finns en länk till den obligatoriska informationen angående cookies så att oerfarna användare skall förstå att det är ofarligt och endast där för att göra det bekvämt för dem. När man loggar in i webbshopen och lägger något i sin varukorg kommer denna automatiskt fram här med en lista på vad man köpt samt en länk till kassan. Man kan även välja att ta bort varor ur varukorgen direkt från sidfoten. Hur varukorgen fungerar mer i detalj beskrivs i Orderbehandlare (kap. 3.3.5).

3.2.3 Functions.inc.php - Funktioner

I den här filen finns en del funktioner som används mer eller mindre ofta som man sedan kan använda genom att bara anropa namnet på funktionen. Detta underlättar kodningen i övriga dokument eftersom man då slipper skriva samma funktion varje gång den behövs. Det gör även att koden inte ser lika krånglig ut och blir lättare att följa så länge man tänker på att vissa av funktionerna är definierade i denna fil. Funktionerna kan ta emot variabler och returnera data som i sin tur kan användas i en ny variabel om man så vill. Här har vi även funktioner som konstant används i webbshopen. Ett bra exempel på en sådan, är funktionen för att öppna databasen, denna måste alltid vara öppen när webbshopens funktioner som hanterar databasen används. Funktionen ser ut så här:

```
$link = mysql_connect("localhost", "user", "password") or
die(mysql_error());
mysql_select_db("databasen", $link);
```

Comment [d11]: Ansluter med information om server, användarnamn, lösenord.

Comment [d12]: Blir något fel stängs anslutningen och felet returneras.

Comment [d13]: Databasen väljs och anslutningen sker.

I det exemplet bör nämnas att användarnamn och lösenord bytts ut mot user och password för att undvika säkerhetskonflikter. User och pass är alltså inga variabler. Eftersom den funktionen körs på servern och inte från klientens dator blir databasens serveradress localhost då databasen ligger på samma server som sidan. På sista kodraden väljer man även vilken databas man vill komma åt och i det här fallet heter den databasen. Skulle något gå fel i anslutningen ser koden ”`or die(mysql_error());`” till så att anslutningen avbryts och felet skrivs ut.

3.2.4 Default.css

Default.css innehåller information om utseende på själva sidans layout. Här kan man själv utforma de mest använda taggarna så att man får dem som man vill ha dem genom att bara anropa taggen. Man kan även definiera hur alla textrutor, listboxar ska se ut genom att skapa egna taggar för dem. Genom att skapa t.ex. taggen `#logo` där man sätter alla värden såsom storlek, bakgrundsfärg, sökväg till bilden, position osv. så kan man sedan anropa denna tag enligt följande, `<div id="logo">`, och på så vis ge logon de förinställda värdena.

```
#logo {
        background-color: #000000;
        background-image: url(../images/logo.jpg);
        background-repeat: no-repeat;
        background-position: left;
        height: 94px;
        padding: 0;
        position: relative;
        text-align: right;
        width: 100%;
        }
```

Comment [d14]: Namn på stilen anges

Comment [d15]: Sätter bakgrundsfärgen. Denna kommer inte dock inte synas eftersom logon täcker hela ytan så den kunde lika gärna utelämnas i detta fall.

Comment [d16]: Lägger in en bakgrundsbild. I detta fallet är det logon (se Figur 3).

Comment [d17]: Anger om bakgrundsbilden ska upprepas, det gör den inte i detta fallet.

Comment [d18]: Anger höjden på #logo-området.

Comment [d19]: Anger bredden på #logo-området.

Man kan även skapa egna taggar där man får definiera vad som ska hända när de används, som t.ex. kan jag skapa en tag som heter `<custom1>` där jag vill att all text ska vara fet, kursiv med fonten Verdana och ha storleken 12 pixlar.

3.3 Sidans funktioner

Efter att sidans grund byggts upp implementerades alla övriga funktioner och delar av sidan som utgör webbshopen och dess nödvändiga komponenter såsom Kassan, Sortimentlistan osv. Funktionerna tas upp på följande sidor i den ordning vi skrev dem för att man enklare ska kunna förstå hur vi kom fram till lösningarna.

3.3.1 Kod för samtliga sidor

Som tidigare nämnts anropar samtliga sidor top.inc.php, bottom.inc.php samt functions.inc.php. Förutom detta har även sidorna denna kod:

```
$strPagename = "Registrera Kund";
echo "<div class=\"pagename\">". $strPagename ."</div>";
```

Comment [d20]: strPagename får värdet Registrera Kund

Comment [d21]: klassen pagename anropas från default.css och strPagename skrivs ut med den klassens egenskaper.

Denna kod skriver ut vilken sida man befinner sig på i en liten ruta under menyn längst upp som kan ses på bilden nedan:

Figur 4. Flik som visar vart på sidan man befinner sig

$strPagename innehåller sidans namn, klassen pagename står definierad i default.css förutom färg och typsnitt etc. för denna klass specificeras även står att en bakgrundsbild ska användas. Denna bakgrundsbild utgör den lilla fliken där sidans namn står (*Figur 4*).

Kommandot ”echo” kommer återkomma frekvent i rapportens kodexempel och dess funktion är att det som står innanför ” ”; efter echo kommer skrivas ut som html. Skriver man t.ex. echo ”hej”; inom PHP klamrar (`<?php ”PHPkod” ?>`) kommer ordet hej skrivas ut i webbläsaren.

3.3.2 Användarbehandlare

Användarbehandlaren består av tre olika funktioner. Registrering där användaren skapas, redigering där den registrerade användaren kan ändra sin information samt en separat redigering där administratören kan se och ändra samtliga användares information.

3.3.2.1 Användarregistrering

En förhållandevis enkel sida som är uppbyggd på följande vis:
När man kommer in på sidan visas ett formulär (`<form>`). Användaren fyller i all nödvändig information och skickar denna. Informationen skickas tillbaks till sidan som sedan kontrollerar eventuella fel i formuläret. Här följer ett exempel på hur felbehandlingen sker:

```
if (isset($_POST["submit"])) {

            if (empty($_POST["ffname"])) {
                $arrError[] = "Du måste fylla i ditt förnamn.";
              }
            //Etc..
}
```

Comment [d22]: Om användaren skickat formuläret

Comment [d23]: Om fältet ffname (förnamn) är tomt läggs felet ”Du måste fylla i ditt förnamn” in i arrayen arrError

Arrayen arrError skrivs sedan ut och användaren ser felen i formuläret och kan sedan rätta till dem. När formuläret fyllts i korrekt läggs informationen från formuläret in i databastabellen users och användaren skickas vidare till en annan sida som berättar att man nu är registrerad och kan logga in och handla.

De problem som stötts på under utvecklingen av denna funktion har varit att vi ansett det vara viktigt att se till så att samtliga av de nödvändiga fälten fyllts i korrekt och att om detta inte gjorts kunna meddela kunden om vad som inte stämmer. Detta för att viktig information om kunden, såsom adress inte ska ha utelämnats vid en beställning. Först använde vi en metod som kollade av om ett fält t.ex. var tomt och skrev sedan ut felet att det fältet var tomt direkt, på så sätt godkänns formuläret inte förrän allt är korrekt ifyllt. Den här funktionen skrev dock bara ut ett fel åt gången och om formuläret innehöll flera fel skulle kunden få ändra i formuläret upprepade gånger. Vi kom sedan på att vi kunde använda oss av en array där samtliga fel matas in och sedan skrivs ut alla på en gång så att kunden direkt kan se alla fel som gjorts och åtgärda dessa. En annan detalj som först felade var att om formuläret innehöll fel så tömdes samtliga fält och kunden var tvungen att fylla i allt på nytt. Detta löstes dock lätt genom att i varje fält sätta in koden:

```
value="". $_POST['fältnamn'. ]""
```

Comment [d24]: Fältets startvärde = Texten som nyss skickades med formuläret från det fältet.

Som gör att informationen som skickats tidigare med formuläret hamnar i fältet den kom ifrån och det lilla problemet var således löst.

3.3.2.2 Användarredigering

Först kontrolleras att man är inloggad, detta görs på följande sätt:

```
if (!isset($_SESSION['id'])) {
   header("location: login.php");
   exit();
} else {
            //Kod för resten av sidan
}
```

Comment [d25]: Kollar om sessionsvariabeln ”id” inte finns. Finns inte denna innebär det att man har inte har loggat in.

Comment [d26]: Om ”id” inte finns hänvisas man till sidan login.php där man ombedes logga in.

Comment [d27]: Lämnar din befintliga sidan och går till header.

Comment [d28]: Finns ”id” så visas resten av sidan.

När kollen utförts och godkänts hämtas information från databasen. Eftersom vi tidigare inte visat hur vi löst detta tar vi med ett exempel:

```
$sql = "select uid, ucity, ustreet, upostcode, uusername, ufname,
ulname, uemail, uphone
from users where uid = '". intval($_SESSION['id']) ."'";

   $result = mysql_query($sql) or die(mysql_error());

   if (mysql_num_rows($result) > 0) {

      $array = mysql_fetch_array($result);
      $array = unescape($array);

            //Visa formulär för redigering

}
```

Comment [d29]: Väljer vilka fält vi vill hämta information ifrån, dessa stoppas i $sql.

Comment [d30]: Väljer att hämta informationen från tabellen ”users” och bara från raden som har den inloggade användarens id så att bara den inloggade användarens information hämtas.

Comment [d31]: Kör SQL-Frågan som läggs i result.

Comment [d32]: Om raderna som hämtats från tabellen är fler än 0 körs koden under.

Comment [d33]: Lägger in innehållet i result i en array.

Comment [d34]: Funktionen unescape från functions.inc.php körs på arrayens innehåll och tar bort slashes (\) som följer med datan från databasen.

3.3.2.3 Administratörens användarredigering

Här får administratören först fram en lista på samtliga registrerade användare inklusive administratörer. Man kan sedan välja en specifik användare och redigera dennes information. Man kan även ta bort användare genom ett enkelt klick i användarlistan. Administratören rekommenderas dock inte att ändra andra användares information då tanken är att användarna själva ska göra detta vid behov. Dock kan det vara bra för administratören att kunna ta bort användare om missbruk misstänks. Därför följer här en förklaring på hur vi gjorde ”ta bort” funktionen. En liknande funktion används även för att ta bort ordrar, produkter och kategorier.

Först har vi i formuläret med en rad kod som visar en länk administratören klickar på för att ta bort användaren:

```
echo "<a href=\"delete_user.php?id=". $array['uid'] ."\"
onclick=\"return confirm('Är du säker på detta?');\">Ta Bort</a>";
```

Comment [d35]: Länk till sidan som tar bort användaren. En variabel som heter ”id” skickas med och innehåller id för användaren som ska tas bort.

Comment [d36]: Klickar man på ”Ta bort” Skickas ett varningsmeddelande där administratören får bekräfta att man vill ta bort användaren.

Därefter skickas man som synes ovan till sidan delete_user.php där användaren tas bort. Där kollas först att man är administratör och får ta bort användare. Är man det körs följande kod:

```
if (isset($_GET["id"])) {
$sql = "delete from users where uid = '". intval($_GET['id']) ."'";
     if (!mysql_query($sql)) {
          $arrError[] = mysql_error();
     }
}
else {
     $arrError[] = "Du valde ingen användare att ta bort.";
}

if (!empty($arrError[0])) {
     showErrors($arrError);
}
else {
     echo "Den valda användaren är borttagen.";

}
```

Comment [d37]: Om den medskickade variabeln ”id” innehåller ett värde tas raden där ”uid” = ”id” bort från tabellen ”users”. Användaren är då borttagen.

Comment [d38]: Uppstod något fel läggs det i arrayen ”arrError” som sedan skrivs ut.

Comment [d39]: Skulle den medskickade variabeln ”id” av någon anledning inte innehålla något värde läggs felet ”Du valde ingen användare..” in i arrayen som innehåller felbeskrivningarna.

Comment [d40]: Innehåller arrError några felbeskrivningar skrivs dessa ut.

Comment [d41]: Uppstod inga fel får man veta att den valda användaren är borttagen. Vilket den då är.

3.3.3 Produktbehandlare

Produktbehandlaren består i huvudsak av fyra olika funktioner: Lägg till produkt, redigera produkt, lägg till kategori samt redigera kategori. Vi tänkte först endast ha kategoriinformationen som ett fält i produkttabellen i databasen, då får administratören manuellt skriva in vilken kategori produkten ska tillhöra vilket vi senare kom på kunde medföra problem eftersom det är osmidigt att behöva kolla och ha i huvudet vilka kategorier som finns. Därför implementerade vi funktionen att skapa kategorier så man i produktregistreringen kan få fram en lista på kategorier man tidigare valt att dela in produkterna i. Skulle rätt kategori saknas kan man då enkelt bara lägga till den en gång för alla. Här följer beskrivningar på hur vi löste dessa fyra delar.

3.3.3.1 Lägg till & redigera produkt

Den här delen är i grunden väldigt lik användarregistreringen som beskrevs i kap. *3.3.2.1* vad gäller formuläret och felkontrollen. Därför kommer endast det som skiljer sig från denna att beskrivas här.

Det första som händer när man kommer in på sidan är att en koll görs så att det säkert är en administratör som är inloggad. Denna koll är ganska simpel men samtidigt effektiv och görs på följande sätt:

```
if ($_SESSION['status'] == 'a') {
            //Här inne finns koden för resten av sidan
}else{
            Echo “Du har inte rättigheter att utföra denna åtgärd”;
}
```

Comment [d42]: Kollar så att sessionsvariabeln ”status” är ”a” som står för administratör och kör i så fall koden på resten av sidan.

Comment [d43]: Om ”status” inte är ”a” skrivs texten ”Du har inte rättigheter att utföra denna åtgärd” ut istället för sidans riktiga innehåll.

Felkontrollen är inte lika sträng i den här delen eftersom den är gjord för administratören som själv bör veta vilken information som är vital.

En skillnad från användarregistreringen var även att produkterna skulle kunna ha bilder knutna till sig. Detta var ett problematiskt moment då vi aldrig råkat ut för detta tidigare. För att hålla löftet om att sidan ska vara lätt att administrera och uppdatera kändes det uteslutet att administratören via ftp skulle sitta och ladda upp bilderna. En smidig uppladdningsfunktion behövdes. Efter att ha kollat runt på lite på php.net kom vi dock fram till att det fanns en funktion för att göra detta i PHP. Ett problem kvarstod dock. Produkten behövde bindas till bildfilen. Först tänkte vi lösa detta genom att lagra den uppladdade filens namn i en egen kolumn i produktdatabasen. Vi kom dock fram till att det både var smidigare och sparade plats i databasen om bildens namn istället blev samma som produktens id. Lösningen på ovanstående problem blev alltså att använda PHP för att ladda upp filen och att ge filen samma namn som produktens id.

För att undvika att någon skriver ett script som laddar hem alla bilder från servern valde vi även att koda namnen med funktionen md5 som kalkylerar en strings MD5 hash genom att använda RSA Data Securitys ”MD5 Message-Digest Algorithm” som returnerar ett 32 tecken långt hexadecimalt hashvärde.[20] Filnamnet ”1.jpg” kan då alltså kodas om till 9bf31c7ff062936a96d3c8bd1f8f2ff3.jpg. Detta känns dock inte så nödvändigt att använda men vi tyckte det kunde vara kul att prova på så därför tog vi med den funktionen.

Ytterligare ett problem uppstod när vi insåg att en produkts id sätts automatiskt av databasen när produkten läggs till och att man från början inte kommer veta vilket värde detta blir. Då vet man inte heller vilket namn bilden ska erhålla om filnamnet ska vara lika med produktens id eftersom man lägger till denna samtidigt som produkten läggs till. Vi löste det genom att man lägger till bilden i efterhand i ”redigera produkt” funktionen. Då är produkten redan skapad och har ett id. Vi insåg dock att det skulle gå att möjliggöra filuppladdningen redan vid ”lägg till produkt” funktionen men valde att ha kvar det som ovan då vi ansåg det tillräckligt och för tidsödande att börja ändra på allt igen. Eventuellt kommer detta rättas till vid ett senare tillfälle.

[20] http://se.php.net/manual/sv/function.md5.php

För att både kunna behandla filer och vanlig formulärdata måste `<form>` taggen se lite annorlunda ut:

```
<form method="post" action="". $_SERVER['REQUEST_URI'] .""
enctype="multipart/form-data">";
```

Comment [d44]: Talar om att formuläret innehåller olika sortes data.

När formuläret är ifyllt och en felkoll kontrollerat att formuläret inte innehåller några kritiska fel körs följande kod:

```
$sql = "update products set pname = '". $_POST["name"] ."', pprice =
'". $_POST["price"] ."', pstock = '". $_POST["stock"] ."', ptype =
'". $_POST["type"] ."', pweight = '". $_POST["weight"] ."' where
pnumber = '". $_GET['number'] ."'";

mysql_query($sql);

$arrExtension = explode(".", $_FILES['file']['name']);
$strExtension = ".". $arrExtension[1];

$strUploaddir = "images/products/";
$strFilename = $strUploaddir . md5($_GET['number']) . $strExtension;

if ($_FILES['file']['type'] !== 'image/pjpeg' &&
$_FILES['file']['type'] !== 'image/jpeg') {
          $arrError[] = "Du får bara ladda upp bilder med filtypen
          jpg!";
}

else{
          if (!move_uploaded_file($_FILES['file']['tmp_name'],
          $strFilename)){

                    $arrError[] = "Något blev fel, försök igen.";

          }

}
```

Comment [d45]: Uppdaterar informationen i tabellen ”products” där fältet ”pnumber” är samma som numret som skickats med i variabeln ”number”.

Comment [d46]: Delar upp filen i namn och filändelse och lgger dessa i arrayen ”arrExtension”.

Comment [d47]: Hämtar filändelsedelen från ”arrExtension” och lägger den i ”strExtension”

Comment [d48]: Sökväg till mappen där filen som laddats upp ska ligga.

Comment [d49]: strFilename får värdet för filens namn. Md5funktionen som nämnts i 3.3.3.1 används för att koda filnamnet.

Comment [d50]: Kollar om den uppladdade filen är av typen jpg då endast detta format tillåts.

Comment [d51]: Kontrollerar så filen laddats upp, om inte läggs felet ”Något blev fel, försök igen” in i ”arrError” som sedan skrivs ut.

3.3.3.2 Lägg till & redigera kategori

Dessa två funktioner är förhållandevis simpla. Syftet med dessa är att skapa fasta kategorier som man sedan mha. en rullista ska kunna välja när man lägger till eller redigerar en produkt. Ett alternativ var att låta administratören skriva in vilken kategori produkten hör till manuellt men då måste administratören hålla koll på vilka kategorier som finns och se till så att stavningen blir samma som i övriga produkter med samma kategori, ett problem som elimineras med fördefinierade kategorier. Här matar administratören in kategorins namn samt en kort beskrivning om vad kategorin innehåller. Koden för detta är snarlik den för att registrera (kap. 3.3.2.1) och redigera användare (kap. 3.3.2.2).

Ett lite problematiskt och intressant moment är dock hur man sedan ska få in de inmatade kategorierna i en rullista i lägg till respektive redigera produkt. Därför kommer här ett kodexempel på hur vi löst detta.

Först skrevs en funktion i ”functions.inc.php” vid namn ”writeSelBox”. Detta är en typ av mall för rullistan och innehåller väldigt många variabler som ska skickas med när funktionen anropas, därför kan det vara lite svårt att förstå vad allt innebär. Den något komplicerade och svårförklarade koden ser ut som följande:

```
function writeSelBox($tbl_table, $fld_id, $fld_name, $sel_name,
$emptyallowed = "yes", $pre_sel_id = 0, $emptyname = "Tom"){

    echo "<select name=\"". $sel_name ."\">";
    if ($emptyallowed == "yes") {
        echo "<option value=\"\">- ". $emptyname ." -</option>";
    }

    $sql = "select ". $fld_id .", ". $fld_name ." from ". $tbl_table
    ." order by ". $fld_name ." asc";

    $result = mysql_query($sql);
    while ($array = mysql_fetch_array($result)) {

        $id = $array["". $fld_id .""];
        $name = stripslashes($array["". $fld_name .""]);

        if ($id == $pre_sel_id){
                $selected = " selected = \"selected\"";
        }
        else { $selected = "";
        }
        echo "<option value=\"". $id ."\"". $selected .">". $name
        ."</option>";
        $selected = "";
    }

    echo "</select>";

}
```

Comment [d52]: Definierar funktionens namn samt lägger in en del variabler och konstanter för listans egenskaper.

Comment [d53]: Anger vad variablen där den valda kategorin ska läggas ska heta.

Comment [d54]: Får listan vara tom ska värdet i ”emptyname” visas (exempelvis ”tom”).

Comment [d55]: Hämtar information från databasen som variablerna anger och lägger in den i arrayer och variabler.

Comment [d56]: Om man vill att ett val ska vara förvalt sätts detta in här.

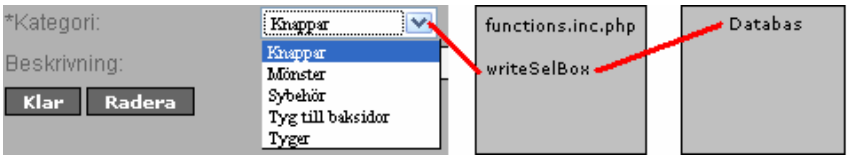
Comment [d57]: Ska inget vara förvalt blir ”selected” tom.

Comment [d58]: Fyller i alternativen i rullistan.

Comment [d59]: Avslutar rullistan.

Rullistan skapas sedan genom att anropa funktionen writeSelBox samtidigt som man skickar med argument som ska sättas in i funktionens variabler:

```
writeSelBox("categories", "cid", "ccategory", "type", "no");
```



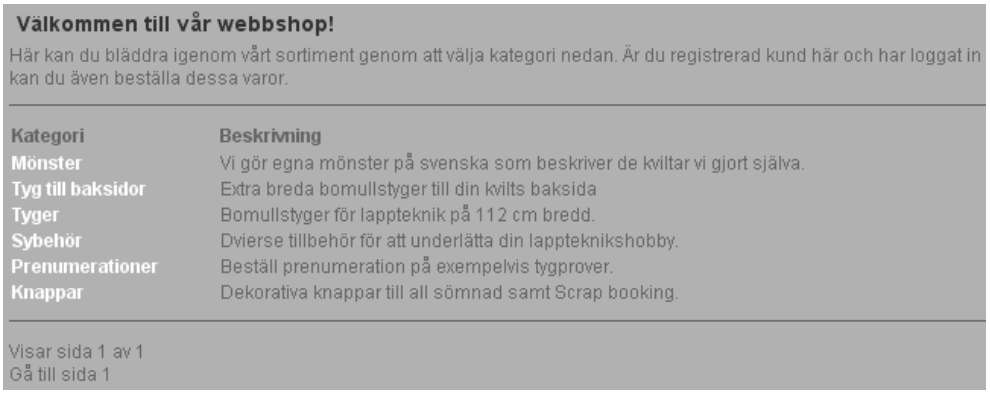
Comment [d60]: Argumenten som skickas med väljer (i ordning från vänster):
1. Tabellens namn (tbl_table).
2. Fältets id (fld_id).
3. Fältet som innehåller kategorins namn (fld_name)
4. Vad variablen där den valda kategorin ska läggas ska heta (sel_name).
5. Om alternativet ”tom” får visas (emptyallowed).

Figur 5. Beskrivning av funktionen writeSelBox

När funktionen anropats och korrekta argument skickats med körs funktionen som då hämtar data från fälten i databasen man angett och skapar rullistan som då innehåller kategorierna som finns i databasen (se *Figur 5*). Man kan också radera kategorier, detta fungerar ungefär som att radera en användare eller produkt.

3.3.4 Sortimentlista

Sortimentlistan utgör själva webbshopen där kunderna kan se vilka olika varor som finns till försäljning och om man är inloggad även lägga dessa i kundvagnen.



Figur 6. Webbshopens förstasida

När man först kommer in i webbshopen visas en lista på de olika kategorier som finns samt en beskrivning till dessa (se *Figur 6*). Kategorierna är länkar som ser ut som följande: `webbshop.php?id=5`. Man skickas alltså till samma sida fast variabeln ”id” får kategorins IDnummer (i detta fallet 5). Detta nummer används sedan för att lista samtliga produkter som har samma IDnummer, alltså produkterna som tillhör kategorin man valt.

Comment [d61]: Kategorins idnummer.

Figur 7. Produkter listade i webbshopen

Information om produktens id, namn, pris och beskrivning hämtas sedan från databasen och samtliga produkter med samma kategoriID listas (se *Figur 7*). Eftersom bildernas namn är kodade med md5 som nämndes i kap. 3.3.3.1 måste de avkodas för att sedan matchas med produktens IDnummer. Detta på följande vis:

```
$strImage = "images/products/". md5($array['pnumber']) .".jpg";
```

Comment [d62]: strImage får sökvägen till bilden. Bildens namn fås genom att koda produktens ID (pnumber) med md5.

Saknas en bild för produkten visas istället en tom bild. Detta kontrolleras såhär:

```
if (file_exists($strImage)) {
     strImage = "<a href=\"". $strImage ."\"><img
     src=\"thumbnail.php?image=". $strImage
     ."&height=75&width=60\"/></a>";
} else {
     $strImage = "<img src=\"images/nopic.gif\"alt=\"Ingen bild\"/>";
}
```

Comment [d63]: Kollar om filen med namnet som ligger i strImage finns.

Comment [d64]: Finns bilden visas den med hjälp av en ”thumbnal”-funktion som beskrivs på nästa sida.

Comment [d65]: Finns ingen bild för produkten visas bilden ”nopic.gif” istället.

I föregående kod användes en sida som heter thumbnail.php. Detta är en lösning på ett problem vi hade med bildbehandlingen. Ett första alternativ var att ta den stora bilden som laddats upp och sedan sätta in den där de små bilderna vid produkterna är och bara skriva in att den ska vara mindre. Då laddas dock hela den stora bilden även om den är liten, detta sker även för samtliga bilder som visas och förhöjer laddningstiden för sidan drastiskt. Därför har vi i thumbnail.php en funktion som använder det så kallade ”GD-biblioteket” i php. Detta bibliotek innehåller funktioner för att behandla bilder på servern innan de visas för klienten. I vårt fall använde vi det för att förminska bilden för att sedan visa bilden i full storlek när användaren klickar på den. Koden i thumbnail.php har vi inte skrivit själva utan endast redigerat lite i. Grovjobbet är gjort av Mårten Andersson[21]. Den fullständiga koden för bildredigeringen finns i bilaga 1 (kap. 7.1).

Ett av problemen vi funderat mest över var hur vi ska lösa hanteringen av produkter kunden valt att lägga i varukorgen. Vår första tanke var att ha en tabell i databasen för varukorgen och att i den lägga in rad för rad varans id, id för kunden som beställt varan, en flagga som talar om ifall kunden valt att beställa dessa varor samt en flagga som talar om att Tyger & Ting skickat varan till kunden. Detta skulle dock innebära en hel del SQL-frågor under tiden kunden surfar på sidan vilket vi försökt undvika. Efter att ha diskuterat problemet med en kompis och expert (Linus Bäckman[22]) på ämnet kom vi fram till att man skulle kunna lägga varorna till varukorgen i sessionsvariabler, dessa har tidigare använts tillsammans med cookies för att minnas om kunden är inloggad samt att kolla dennes status.

```
$_SESSION['id']
```

Comment [d66]: En liten minnesuppfriskare på hur en sessionsvariabel ser ut. Här hämtas sessionsvariabeln ”id” som innehåller den inloggade kundens unika IDnummer.

I föregående fall har vi dock bara lagt in någon enstaka siffra i variabeln. Nu behöver vi först och främst få in flera produkter i samma variabel. Därtill måste både produktens id och antal beställda enheter finnas med. Det visade sig att den som kommit idén med sessionsvariabler redan tänkt på detta och att det gick och göra ungefär som innan:

```
$_SESSION['products'][$_POST['number']] = $_POST['ammount'];
```

Comment [d67]: I Variabeln products lagras de valda produkternas ID. De får även ett värde knutet till sig, i detta fallet ”ammount” som innehåller antalet produkter med samma id som köpts.

På så vis är halva problemet löst. Det som nu återstår är att packa upp den sessionslagrade informationen och göra den synlig vid behov. I den här delen finns dock inget behov av detta så nästa halva av problemlösningen behandlas under rubriken varukorg (kap. 3.3.5.1) där behovet av att visa de valda varorna från kunden uppstår.

[21] marten@odg.nu
[22] lijjnus@hotmail.com

3.3.5 Orderbehandlare

Orderbehandlaren innefattar en hel bunt olika element som rör beställning av varor. På följande rader kommer varans väg från varukorgen till kunden beskrivas och väsentliga delar av koden för de olika momenten kommer beskrivas precis som i föregående rubriker.

3.3.5.1 Varukorg

I beskrivningen av sortimentlistan (kap. 3.3.4) behandlade vi på de sista raderna ett problem med hur kundens varor skulle lagras och sedan kunna visas i exempelvis varukorgen. Problemet förblev till hälften olöst då andra halvan av problemet passade bättre att lösa under denna rubrik. I sortimentlistan lagrade vi kundens valda varor i sessionsvariabeln ”products”. Här hamnade information om varornas id och antalet beställda varor. Sista halvan av problemet består av att ”packa upp” informationen i sessionsvariabeln och sedan lista informationen i en rullista ungefär som i kategoribehandlaren (kap. 3.3.3.2). Problemet med att plocka ut informationen från sessionsvariabeln löstes med funktionen ”`implode`” som tar informationen från arrayen med produkter och lägger den i en sträng. Varukorgen som vi, efter lite ändringar i planerna, valt att lägga i bottom.inc.php (sidfoten) fungerar då enligt följande kommenterade kod:

```
if (isset($_SESSION["id"]) && (count($_SESSION['products']) > 0)) {

    echo "<form method=\"post\"
    action=\"". $_SERVER['REQUEST_URI'] ."\">";

    echo "<table>";
    echo "<tr><td> Varukorg:<br></td></tr>";
    echo "<tr><td>";

    $productlist = implode(",", array_keys($_SESSION['products']));

    $sql = "select pnumber, pname from products where pnumber IN(".
    $productlist .") order by pname asc";
    $result = mysql_query($sql);


    if (mysql_num_rows($result) > 0) {
          echo "<select name=\"item\">";

          while ($array = mysql_fetch_assoc($result)) {
                $array['pname'] = stripslashes($array['pname']);
                echo "<option value=\"". $array['pnumber'] ."\">".
                $_SESSION['products'][$array['pnumber']] ."st ".
                $array['pname'] ."</option>";
          }
          echo "</select>";
    }
```

Comment [d68]: Kollar om man är inloggad och om kunden valt att lägga några varor i varukorgen. Är det så körs koden nedan som visar varukorgen.

Comment [d69]: Skapar ett formulär (behövs för att kunden ska kunna ta bort varor ut varukorgen).

Comment [d70]: En tabell ritas upp.

Comment [d71]: Lägger med hjälp av implode in produkternas ID i en sträng.

Comment [d72]: Hämtar information om de valda produkternas namn från databasen.

Comment [d73]: Ritar upp rullistan om det ligger produkter i varukorgen (vilket vi även kontrollerat på första raden).

Comment [d74]: Lägger in alla valda produkter i varukorgens rullista.

Till varukorgen finns även en funktion som tillåter användaren att ta bort markerade produkter ur varukorgen. Denna funktion fungerar dock ungefär likadant som den som tar bort användare (kap. 3.3.2.3) och kommer inte beskrivas ytterligare. Vid varukorgen finns även en länk till kassan som beskrivs härnäst.

3.3.5.2 Kassa

Kassan är den sista hållplatsen innan beställningen skickas iväg till Tyger & Ting för att sedan skickas till kunden. Här får kunden en chans att verifiera så allt från kundinformationen stämmer. Koden för denna del använder olika metoder som vi redan beskrivit, därför kommer inga kodexempel tas upp här.

Genom att först hämta informationen från databastabellen användare från fältet som har samma id som den inloggade kunden tas all vital information om kunden fram och visas upp så kunden kan bekräfta att den stämmer innan ordern skickas. Detta ser ut på följande sätt:

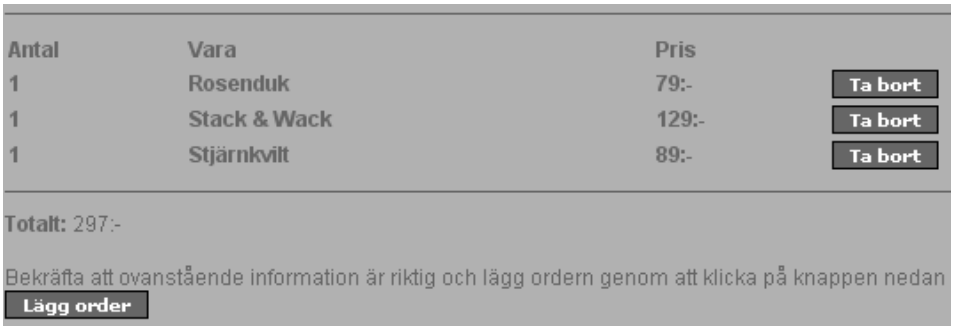
Väljer du att beställa dessa varor kommer de skickas till följande adress:
<förnamn> <efternamn>
<gata>
<postnummer> <postort>

Skulle vi behöva kontakta dig angående denna order kommer det ske antingen via:
Telefon: <telefonnumret> eller **Mail:** <mailen>

Genom att lägga denna order bekräftar ni även att ovanstående information är riktig, är det något som inte stämmer går detta ändra här.

Comment [d75]: Länk till sidan där kunden kan redigera sin egen information (3.3.2.2).

Under kundens information kommer sedan en lista på vad som är beställt, priset för varje produkt (antal x pris) samt orderns totala kostnad (se *Figur 8*).



Figur 8 – Exempel på sammanställning av order i kassan

Längst ner i *Figur 8* finns också en knapp som låter kunden lägga sin order. Trycker kunden här läggs ordern in i databasen och en orderbekräftelse visas enligt nedan:

Din order är nu lagd. Du kan se om din order skickats under "konto" sektionen och "mina ordrar".

Ditt ordernummer är: <ordernummer>
Referera till detta om du har frågor angående din order.

I databasen läggs ordern in på följande sätt:

- En rad i tabellen ”orders” läggs till innehållande kundens id, orderns id, datumet ordern lades samt en flagga som sätts om ordern har behandlats och skickats till kunden.

- En rad för varje produkt som finns i ordern innehållande information om antalet likadana produkter, produktens id och vilken order produkten hör till.

Ett litet problem som ännu inte fått någon riktig lösning här är hur fraktkostnad ska behandlas. Vi har diskuterat detta lite och kommit fram till att man antingen kan lägga på lite på produkternas pris och sedan låta frakten ingå eller att man har en fast frakt för beställningar under en viss summa och om priset överstiger den summan blir frakten gratis. Hur detta kommer att bli är upp till Tyger & Ting och skulle lösningen innebära att ändringar i sidans kod behöver göras har vi lovat att ställa upp när de beslutat sig.

3.3.5.3 Obehandlade ordrar & Orderhistorik

Dessa två funktioner nås endast av administratören och är till för att behandla ordrar som lagts av kunderna.

Obehandlade ordrar består av en lista över ordrar som lagts men ännu inte skickats till kunden. En lista med samtliga obehandlade ordrar skrivs först ut. Under varje order finns tre knappar. En knapp för att ta bort en order om denna tycks vara oseriös eller på annat sätt felaktig, en för att markera när ordern är skickad till kunden och en knapp som visar all information om ordern. Väljer man att visa all information om ordern ser det ut enligt följande:

Order# <ordernummer> info

Beställd <datum orden lagts> av:
<förnamn> <efternamn>
<gata>
<postnummer> <postort>

Beställda varor:
<antal> st <varans namn> <varans pris (antal x pris)>
Totalt: <totala summan för hela ordern>

Med ovanstående information kan sedan ordern skickas till kunden. Administratören markerar därefter att ordern har skickats och ordern hamnar då i orderhistoriken.

I Orderhistoriken listas alla gamla ordrar som skickats. Denna del är i stort sett identisk med obehandlade ordrar. Den enda skillnaden bortsett från att bara de skickade ordrarna visas är att man istället för att markera en order som skickad kan få ordern oskickad. Ordern hamnar då åter igen i obehandlade ordrar.

3.3.5.4 Mina ordrar

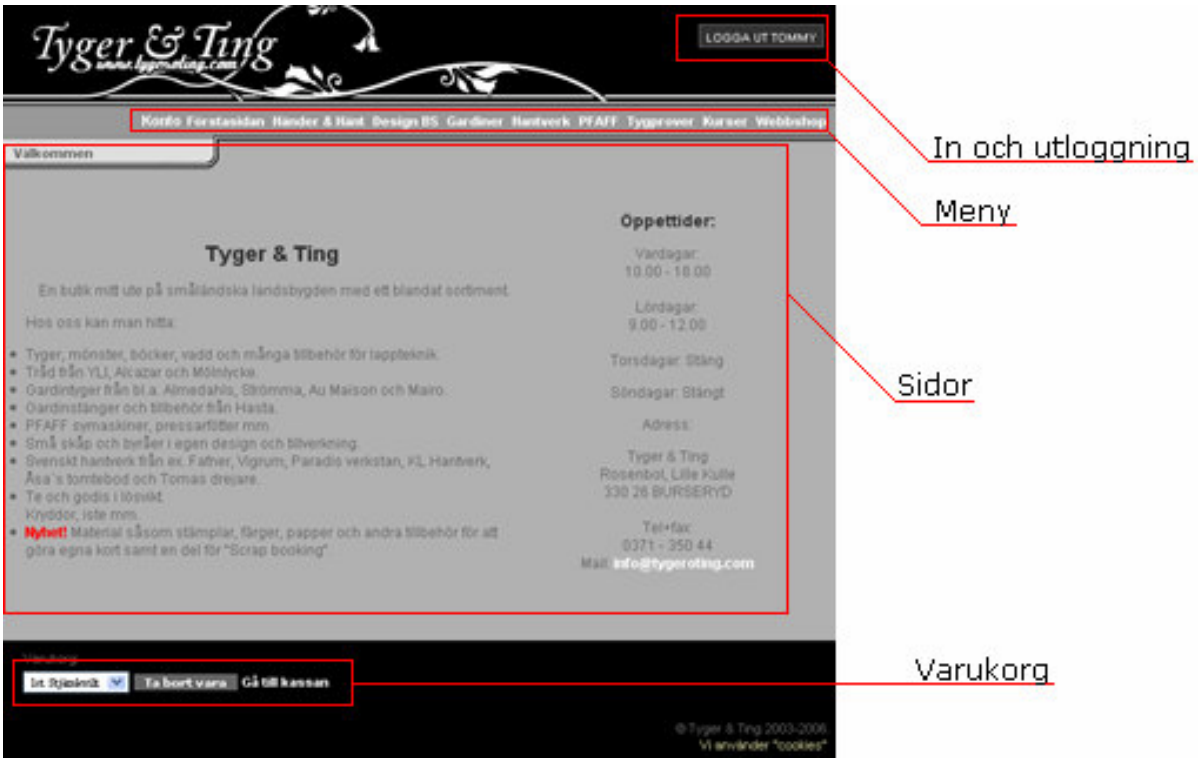
Även denna del är mycket lik ovanstående delar. Här är det emellertid kunden som kan gå in och kolla sina ordrar. Här listas både de ordrar som skickats och de som inte ännu skickats. Idén är att kunden här ska kunna se om ordern skickats samt att kolla upp övrig information om ordern vid behov. Kunden kan inte själv ta bort ordrar. Vill kunden annullera en order måste denne kontakta företaget innan ordern skickats.

4 Resultat

Resultatet består sammanlagt av 20 dynamiska PHPsidor som sammanlagt motsvarar ca: 1900 rader kod samt en statisk sida innehållande information om cookies och säkerhet riktad till kunden. Utöver detta har även en databas med 5 tabeller av varierande storlek skapats. Tabellerna vi slutligen kom att använda i databasen var:

- Categories – Lagring av produktkategorierna
- Order_items – Produkter knutna till en order
- Orders – Alla ordrar som lagts
- Products – Information om produkterna
- Users – Information om de registrerade kunderna

Tillsammans bildar dessa sidor och databasen en lättadministrerad och kundvänlig företagssida med webbshop som även stöder statiska sidor (se *Figur 8*).



Figur 9 – Sidan i sin helhet

5 Slutsats och diskussion

Allt som allt är vi både nöjda och stolta över det slutgiltiga resultatet. Jämför man resultatet med kravspecifikationen (kap. 1.2.1) finner man att resultatet skiljer sig lite från denna. Detta beror dock inte på att vi inte klarat lösa problemen enligt kravspecifikationen utan att vi helt enkelt kommit på bättre lösningar under projektets gång. Exempelvis tänkte vi först att kundvagnen skulle komma fram i sidans topp efter man loggat in. Efter att ha undersökt lite andra webbshopar kom vi dock fram till att de flesta hade sin kundvagn i sidans fot vilket också gjorde toppen med logon i renare. Dock har inte många undantag från kravspecifikationen gjorts vilket känns bra för det visar att resultatet blivit som vi tänkt oss.

Tester att uppdatera sidan med statiska sidor genererade av företagets program Intellyweb (kap. 3.1.3) har också gjorts och detta verkar fungera precis som vi tänkt oss. Dock har administratören på Tyger & Ting inte haft tid att testa själv hur det fungerar så vi har inget utlåtande från dem.

Den allmänna användarvänligheten på sidan har testats av både mer och mindre datavana användare och de har inte haft några problem med att leta upp produkterna de vill ha och beställa dessa. Administratören har även provat på att behandla dessa beställningar och är efter viss hjälp från oss väl införstådd och nöjd med hur detta fungerar.

Hanteringen av databasen och vad som skall lagras i denna känns också lyckad. Vi försökte att bara lagra det nödvändigaste i databasen och på så sätt hålla ner antalet tabeller i den för att undvika allt för många SQL-frågor och därmed göra sidan snabbare samtidigt som koden inte blir lika tung.

Det fortsatta arbetet på sidan kommer göras av administratören på Tyger & Ting innan den görs tillgänglig för offentligheten. Det som ska göras är att ta bilder på produkterna de avser sälja i webbshopen och även lägga upp produkterna med bilder och information i databasen. De statiska sidorna ska också uppdateras och läggas upp. När de är nöjda med produktsortimentet och sidorna kommer vi hjälpa dem att ersätta deras standardsida (index.html) med en sida som automatiskt skickar kunden till den nya (index2.php) och resultatet blir därmed tillgängligt för allmänheten.

6 Referenser

1 http://www.intellyweb.com (2006-06-07)
2 http://computer.howstuffworks.com/web-page.htm (2006-06-07)
3 http://computer.howstuffworks.com/cgi.htm (2006-06-07)
4 http://news.netcraft.com/archives/web_server_survey.html (2006-06-07)
5 http://www.microsoft.com/windowsserver2003/iis/default.mspx (2006-06-07)
6 http://www.webwizguide.info/asp/tutorials/what_is_asp.asp (2006-06-07)
7 http://databases.about.com/od/access/l/aaaccess1.htm (2006-06-07)
8 http://www.phpportalen.net/ (2006-06-07)
9 http://www.westciv.com/style_master/academy/css_tutorial/ (2006-06-07)
10 http://www.mysql.com/ (2006-06-07)
11 http://computer.howstuffworks.com/question82.htm (2006-06-07)
12 http://java.sun.com/products/jsp/ (2006-06-07)
13 http://httpd.apache.org/ (2006-06-07)
14 http://www.adobe.com/products/dreamweaver/ (2006-06-07)
15 http://www.adobe.com/products/photoshop/ (2006-06-07)
16 http://www.intellyweb.com (2006-06-07)
17 http://www.epsab.se (2006-06-07)
18 http://www.redhat.com/ (2006-06-07)
19 http://www.epsab.se/om.php (2006-06-07)
20 http://se.php.net/manual/sv/function.md5.php (2006-06-07)
21 marten@odg.nu (2006-06-07)
22 lijjnus@hotmail.com (2006-06-07)

7 Bilagor

Bilaga 1 Thumbnail.php – Script som skapar en mindre temporär bild.

7.1 Bilaga 1 - Thumbnail.php

```
<?php
/*
-----------------------------------------------------------------------
By Mårten Andersson, marten@odg.nu
Makes a resized temporary image.
This script can only handle jpg and png imagetypes.
INPUT: imagename and the size of the new image in percent.
EXAMPLE: <img src='changeImgSize.php?userfile=image.png&procent=200'>
The example will create a copy of the image with the double size
-----------------------------------------------------------------------
*/

$userfile = $_GET["image"];
$size = GetImageSize($userfile);

if ($size[2] == '2') {
   $src_img = imagecreatefromjpeg($userfile);
   $headertype = "jpeg";
} elseif($size[2] == '3') {
   $src_img = imagecreatefrompng($userfile);
   $headertype = "png";
}

if (isset($_GET["width"])) {
           $new_w = $_GET["width"];
} else {
           $new_w = 133;
}

//Räkna ut skalenlig bredd också
if (isset($_GET["height"])) {
           $new_h = $_GET["height"];
} else {
           $scale = $new_w / imagesx($src_img);
           $new_h = $scale * imagesy($src_img);
}

$dst_img = imagecreatetruecolor($new_w, $new_h);
imagecopyresampled($dst_img, $src_img, 0, 0, 0, 0, $new_w, $new_h,
imagesx($src_img), imagesy($src_img));
header ("Content-type: image/$headertype");

if ($size[2] == '2') {
           imagejpeg($dst_img, '', 100);
} elseif($size[2] == '3') {
           imagepng($dst_img, '', 100);
}

imagedestroy($dst_img);
?>
<html>
<head>
<title>Variabler</title>
</head>
<body>
</body>
</html>
```