Bygga och Förstå: Webbutveckling

Fabian Bakkum

Contents

Målgrupp

Detta häfte är skriven som hjälpmedel till undervisning i Webbutveckling och Webbteknik på gymnasiet, men får såklart användas av alla som vill lära sig mera om webben och webbutveckling. Genom detta häfte får man lära sig grunderna i webbutveckling. Eleven börjar läsa om historian om webben och grunderna i HTML och slutar med en enkel dock fullständig fungerande applikation skriven i PHP som interagerar med en databas.

Häftet går inte igenom varenda CSS egenskap eller funktion i PHP och satsar sig på dem praktiska tillämpningar inom applikationen som byggs i detta häfte. Genom häftet får eleven jobba och klura med praktiska exempel. Meningen är att eleven inte bara läser detta häfte, men jobbar med dem exempel som följer med detta häfte.

Historia

Internet

Efter att Sovjet Unionen lyckades lansera dess första satellit till jordens omloppsbana, såg USA nödvändigheten att starta ARPA för att forska och skapa avancerade teknologier. 1962 startade ARPA forska på ett sätt för att få flera datorer att kommunicera med varandra. Efter några år av forskning fick de det första fungerande nätverk som innehöll fyra datorer år 1969. Detta nätverk fick namnet ARPANET.

Konceptet spred sig vidare till universiteten. JANET nätverket var ett av dem första universitet nätverk som tillät de olika campus att dela filer och skicka e-post mellan varandra. Flera nätverk uppstod, men alla använde sina egna protokoll. Robert Kahn och Vinton Cerf försökte därför framställa regler för ett mera öppet nätverk. Protokollet som släpptes 1974 fick namnet Internet Transmission Control Program och gjorde det lättare för nätverk att sammanknytas med andra nätverk. Den färdiga versionen av TCP/IP specifikationen släpptes 1981. Internet som vi känner till det idag hade anlänt!

World Wide Web

Gopher var ett populärt textbaserat menysystem och nätverksprotokoll för sökning av dokument på internet. Eftersom Gopher började införa licensavgifter, började många organisationer leta efter ett alternativ. Den europeiska Council for Nuclear Research (CERN) hade ett sådant alternativ. Tim Berners-Lee hade jobbat på ett system i vilket text kunde innehålla länkar till andras verk, vilket tillät läsaren att hoppa snabbt mellan olika dokument. Han skrev en server som kunde publicera dessa dokument (kallades hypertext) och ett program som kunde läsa dessa dokument och kallade systemet för World Wide Web.

På början av 90-talet fanns det två stora aktörer på webbläsarens marknad, Microsofts Internet Explorer och Netscape Navigator. För utvecklare var det en mörk tid, eftersom det inte fanns någon standard och väsentligen fick man skapa flera separata sidor eller strunta i och stödja flera webbläsare. År 1994 grundade Tim Berners-Lee World Wide Web Consortium (W3C) med stöd från CERN, DARPA (vilket ARPA har byts namn till) och den Europeiska Kommissionen. W3C publicerade flera specifikationer inkluderande HTML 4.01, ett format för PNG bilder och Cascading Style Sheets.

En ny grupp bestående av utvecklare bildades Web Standards Project för att övertyga dem ledande företag att följa dem standarder W3C hade tagit fram. Nu för tiden följer dem flesta webbläsare specifikationen med ett fåtal undantag.

Hypertext Markup Language

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Test</title>
</head>
<body>
    <h1>Test</h1>
    <p style="color:red;">Bacon ipsum dolor amet.</p>
</body>
</html>
```

Ovanstående kod är skriven i HTML. Enkelt förklarat så är HTML (Hypertext Markup Language) ett språk som används för att skapa hemsidor. Man kan säga att det är hemsidans skelett.

Dokumentet är uppbyggd av *element*. I exemplet ser vi bland annat `<h1>Test</h1>` elementet. Alla element börjar med en tagg slutar med en tagg. Uppgiften av detta element är att visa innehållet ”Test” som en rubrik.

Så låt oss dyka in i koden och beskriva vad allt gör rad för rad. Vi ser att dokumentet börjar med `<!DOCTYPE html>`. Alla dokument börjar med en så kallad DOCTYPE, vilken säger åt webbläsaren vilken version av HTML dokumentet är skriven i. Det är HTML5 i detta fall.

`<html></html>` beskriver ett HTML dokument.

`<head><head>` innehåller all information som inte visas direkt på sidan. Bland annat kan det innehålla titeln på sidan och meta taggar, som hjälper sökmotorer identifiera vår hemsida.

`<meta charset="utf-8" />` För att webbläsaren ska kunna visa dokumentet korrekt, så måste den veta vilken teckenkodning som ska användas. Teckenkodningen är sättet på hur data sparas på hårddisken, medans Unicode är ett exempel på en teckentabell. Detta element säger åt webbläsaren att dokumentet ska läsas med UTF-8 teckenkodning. Du har säkert sett frågetecken på platser där egentligen ett å bör stå förut? Det är ett tecken på att sidan använder sig av fel teckenkodning.

UTF-8, den standard teckenkodningen för HTML5, har jämfört med ASCII ett stort antal fördelar. Framförallt för språk som svenska som innehåller alla dem här konstiga tecken som ääö. De icke engelska tecken visar upp sig som frågetecken i webbläsaren om man använder ASCII som teckenkodning, eftersom de tecken inte finns i ASCII tabellen. Förutom att UTF-8 är den standard teckenkodningen för HTML5 är den också standard på många moderna operativsystem som till exempel Linux.

ANSI (Windows-1252) är ett tillägg till ASCII och stöder dem icke engelska tecken. HTML 2.0 tillförde att ISO-8859-1 blev den nya standard teckenkodningen. I grund och botten är ISO-8859-1 samma sak som ANSI förutom att ANSI innehåller 32 extra tecken.

`<title>Test</title>` är sidans titel som kommer att visas på fliken.

`<body></body>` Nu kommer vi till den roligare delen av dokumentet. Menyer, tabeller, bilder, text, länkar, verkligen allt som syns på hemsidan befinner sig i bodyn.

`<p style="color:red;">Bacon ipsum dolor amet.</p>` är ett element som visar en paragraf med text. Till skillnad från `<h1>` har detta element ett attribut. Attributet `style=""` tillåter oss bland annat att ändra färgen på texten till röd.

Utvecklingsmiljön

För att kunna utveckla en webbapplikation behöver man inte många verktyg. Huvudsakligen behöver man skaffa sig en bra textredigerare. Vilken spelar ingen roll, men här är en lista på några bra textredigerare som man kan ladda ner.

Platform oberoende: Aptana Studio, Atom, VIM, Emacs

Windows: Notepad++

Linux: Gedit och Kate

Om man inte vet vilken man ska ladda ner, kan man ladda ner Atom. Slutligen behöver man en förnuftig webbläsare som stöder HTML5, Mozilla Firefox till exempel. Utöver det måste man ha tillgång till en webbserver och eventuellt ett skriptspråk och en databasserver.

Första uppgiften

Nu ska vi göra något roligt med HTML. Skolorna har ett stort problem. Att hantera alla veckorapporter från eleverna på papper är ineffektiv och därför måste vi bygga en applikation där eleverna kan göra ett inlägg varje dag med vad de har fått göra på praktikplatsen. Målet är att vi i slutändan har en simpel dock fullständig fungerande applikation, där man kan skapa nya inlägg.

Vi börjar med att skapa en ny fil med namnet `index.php`. Det är filen som servas ut automatiskt av webbservern på rooten av domänet, startsidan. Hur fungerar det? Din webbläsare skickar en HTTP **GET** förfråga till webbservern (värddatorn som serverar besökaren sidan) och svarar i sin tur med att skicka `index.php`. Det finns även andra typer av förfrågningar eller metoder som det kallas, vi kommer dock bara fokusera oss på GET och POST.

GET - Frågar servern att visa en post.

POST - Frågar servern att skapa en ny post.

DELETE - Frågar servern att ta bort en post.

PUT - Frågar servern om tillåtelse för att ladda upp data. Används bland annat för att ladda upp filer.

Efter att du har skapat filen kan du fylla den med följande innehåll. Notera att om du använder Windows så döljer filhanteraren extensionen på filen, dubbelkolla att suffixet på filen är '.html' och inte '.txt'.

```
<!DOCTYPE html>
<html lang="sv">
<head>
    <meta charset="UTF-8">
    <title>Veckorapporten</title>
</head>
<body>
    <header id="header-top">
        <h1 id="logo"><a href="index.php">Veckorapporten</a></h1>
        <nav>
            <ul>
                <li><a href="index.php">Hem</a></li>
                <li><a href="new.php">Ny</a></li>
            </ul>
        </nav>
    </header>

    <div id="page-container">
        <article>
            <h2 class="article-title">Bacon</h2>
            <p class="actions">
                <a href="delete.php?id=1">Radera</a>
                 
                <a href="edit.php?id=1">Redigera</a>
                 
                <span class="date">2015-03-17</span>
            </p>
            <p class="content">
                Idag har jag ...<br />
                Det jag även fick jobba på var ...<br />

                <br /><br />

                Slutligen gjorde jag ...
            </p>
        </article>
```

```
        <footer id="footer-bottom">
            <p>Copyright &copy; 2015 Fabian Bakkum</p>
        </footer>
    </div>
</body>
</html>
```

Som du ser finns det en del tillagd inuti `body` elementet. Låt mig bryta ner det. Du kan öppna dokumentet i Firefox genom att hålla i *Alt* och *F* samtidigt och klicka på *Öppna Fil*, välj dokumentet och klicka på *Öppna*.

`<header id="header-top"></header>` är det första elementet i bodyn. Det är en ny tag i HTML5 dit man bland annat kan placera logotypen, ett sökfält eller navigationsmenyer. Elementet innehåller också ett attribut, *id* med värdet *header-top*. Syftet med class och id attributen är att vi kan använda de för att styla dokumentet med CSS.

`<h1 id="logo"><a href="index.php">...</a></h1>` är sidans titel som innehåller en länk tillbaka till startsidan. Attributet `href="index.php"` anger platsen dit vi vill länka till. Man kan även länka till externa sidor. Till exempel så länkar följande länk till sökmotorn Google:

```
<a href="https://www.google.se/">Klicka här</a>
```

`<nav></nav>` innehåller menyn med alla länkar. I detta fall finns det två knappar på sidan. En för att gå tillbaka till startsidan och en annan knapp för att skapa ett nytt inlägg.

`<ul></ul>` är till för att göra listor. Ett exempel:

- Handla mat
- Tvätta kläderna

`<div></div>` används för att dela upp sidan och kommer att användas i CSS sektionen av häftet.

`<article></article>` Inom detta element kommer inlägget att visas. Vi kommer att sedan att generera ett `article` element för vartenda inlägg.

`<h2></h2>` Inläggets titel.

` ` är ett speciellt HTML tecken som skapar ett tomrum mellan dem två länkar.

`<br />` bryter raden.

`<footer></footer>` innehåller ett meddelande om upphovsrätten. Elementet kan även innehålla kontakt information eller länkar till annan nyttig information.

Extra uppgift:

Placera en slumpvis bild under inlägget.

Cascading Style Sheets

Sidan vi håller på att bygga ser lite tråkig ut. För att få lite färg i det hela måste vi använda oss av CSS. Cascading Style Sheets (CSS) är ett språk för att specificera hur dokumentet ska presenteras till användarna.

Börja med att skapa en ny katalog för kommande uppgift. Skapa en ny fil i den nyss skapade katalogen med namnet `index.html` och en ny katalog med namnet `css` med däri en ny fil kallad `style.css`.

Öppna `index.html` i din favorita texteditor och fyll den med följande grundläggande mall.

```
<!DOCTYPE html>
<html lang="sv">
<head>
    <meta charset="UTF-8" />
    <title>Exempel på CSS</title>
</head>
<body>
    <h1>Ett exempel på CSS</h1>
    <p><strong>E</strong>n paragraf med text.</p>
    <p class="pretty">En till paragraf med text.</p>
</body>
</html>
```

Ladda CSS kod

Det finns tre olika sätt på hur man kan ladda CSS:

- **Inline Style Sheet** - Direkt på elementet med hjälp av `style=""` attributet.
- **Embedded Style Sheet** - Genom att inom `head` lägga till `<style type="text/css">` kan man skriva CSS direkt inuti detta element.
- **External Style Sheet** - Ladda ett externt CSS formgivningsblad.

För att ladda ett externt formgivningsblad måste man lägga in följande element i `head`.

```
<link href="css/style.css" type="text/css" rel="stylesheet" />
```

Hur fungerar CSS?

För att göra texten på sidans titel röd använder vi följande CSS:

```
h1 { color: red; }
```

Vi börjar med en *selektor*. Det kan vara en tagg, en grupp element eller ett individuellt element. I ovanstående kod använder vi endast `h1` som selektor, vilket gör att alla `h1` rubriker visar upp sig som rödfärgad. Allt inom klammerparentesen kallas för en *deklaration*. Nyckelordet `color` kallas för en *egenskap* och röd är dess *värde*.

```
selektor { egenskap: "värde"; }
```

Nu har vi använd oss av en tagg selektor, men tänk om vi bara vill formge en paragraf och inte alla? Då kan vi använda oss av ett *class* eller *id* attribut. I regel används *id* endast för att formge ett enda element, medans *class* kan återanvändas på flera ställen.

Prova följande formgivningsblad:

```
h1 {
    color: red;
}

p {
    color: green;
}

.pretty {
    font-family: sans;
    text-decoration: underline;
    text-align: center;
}

strong {
    color: blue;
    font-size: 28px;
}
```

Rubriken har fått en ny färg, röd och alla paragrafer har blivit grönfärgade. Dock har bara den andra paragrafen fått ett nytt typsnitt, ett understreck och blivit centrerad i mitten. Den andra paragrafen har en class nämnd *pretty*. I formgivningsbladet ser du att det finns en selektor som heter `.pretty`. Till skillnad från de andra regler som bara innehar tagg selektorer så har denna ett

punkt framför selektorn. Punkten i namnet tyder på att det är en class selektor. Likadant går att göra med id attributet, dock måste man byta ut punkten mot ett nummertecken.

Extra uppgift:

Försök göra sidan så fult som möjligt utan att ändra dokumentstrukturen i HTML filen. Ta hjälp av Mozilla Developer Network för att hitta fler CSS egenskaper som gör sidan ännu fulare.

https://developer.mozilla.org/en-US/docs/Web/CSS/Reference

Utforma webbapplikationen

Det finns oäntliga många möjligheter med CSS. Dags att använda våra CSS kunskaper för att designa vår webbapplikation. De flesta webbläsare tillämpar sina egna formgivningar. Till exempel så lägger den till stoppning på alla sidor. Vi använder nedanstående CSS för att ta bort stoppningen på alla taggar.

```
* {
    padding: 0;
    margin: 0;
}
```

Härnäst behöver vi formge bodyn. Där kan man ställa in bakgrundsfärgen samt standard typsnittet och text färgen. Som du ser använder vi, till skillnad från tidigare skriven kod, färgkoder istället. På så sätt kan vi använda oss av alla möjliga färger. Det är viktigt att välja neutrala färger till bakgrunden. Blå bakgrund med röd text blir inte godkänd. Inte bara är det fult, men sidan blir även oanvändbar för färgblinda och personer med nedsatt syn.

```
body {
    background-color: #F9F9F9;
    font-family: 'Roboto';
    color: #212121;
}
```

Som jag tidigare nämnde, kan man använda en id selektor för att formge ett enda element. Härnedan ser du att vi använder en id selektor, så att vi kan formge menyn. Höjden är satt till 60 pixels och bredden är satt till 100%, alltså lika bred som webbläsarens fönster. Jag la dessutom till en subtil skuggeffekt.

```
#header-top {
    background-color: #FF5722;
    height: 60px;
```

```
    width: 100%;
    box-shadow: 0 1px 10px black;
}
```

För att placera loggan i mitten använder vi oss av padding egenskapen.

```
#logo {
    font-family: 'Roboto';
    padding-top: 11px;
    padding-left: 30px;
    color: white;
}
```

Eftersom webbläsaren lägger till ett understreck och målar texten blå på länkar behöver vi explicit berätta att den inte ska göra så. Selektorn pekar på länkar inuti `#logo`, men endast på barn till `#logo`. Om du vill att formgivningen även ska tillämpas på barnbarn bör man ta bort större än tecknet.

```
#logo > a {
    font-family: 'Roboto';
    color: white;
    text-decoration: none;
}
```

Nästa steg är att placera navigationsmenyn på rätt ställe. Eftersom vi vill placera navigationsmenyn inuti `header` menyn, kan vi använda oss av *absolut* positionering. Om man använder sig av *absolute* positionering får man tillgång till top, right, bottom och left egenskaperna.

```
nav {
    position: absolute;
    top: 21px;
    left: 300px;
}
```

Menyn ser ännu gräslig ut. Knapparna ska presenteras på en enkel rad och punkterna framför ska tas bort i det här fallet.

```
nav > ul > li {
    display: inline-block;
    list-style-type: none;
}
```

Följande kod gör att knapparna blir vita och skapar ett 25 pixels mellanrum mellan länkarna. Vi tar även bort understrecket med hjälp av `text-decoration` egenskapen.

```
nav > ul > li > a {
    color: white;
    padding: 0 25px;
    text-decoration: none;
}
```

För att få en fin effekt när vi drar över muspekaren kan man använda sig av en så kallad *pseudo klass*. Här använder vi pseudo klassen *:hover* för att ändra färgen på länkarna då man drar över muspekaren.

```
nav > ul > li > a:hover {
    color: black;
    padding: 0 25px;
    text-decoration: none;
}
```

Jag gjorde en `div` med identifierare `#page-container`, eftersom vi vill centrera artiklarna på mitten av sidan. Härnedan använder jag en egenskap kallad `max-width` för att begränsa storleken och `margin: auto;` för att centrera allt.

```
#page-container {
    max-width: 800px;
    height: 100%;
    margin: auto;
    padding-top: 50px;
}
```

Alla artiklar får en kant. Det finns olika mönster, från raka linjer till prickiga. `padding: 10px;` gör att texten inte kläms direkt på kanten och skapar ett 10 pixels mellanrum på alla sidor. `margin-bottom: 20px` skapar ett 20 pixels mellanrum så att man får lite luft mellan alla artiklar.

```
article {
    border: 1px dotted black;
    padding: 10px;
    margin-bottom: 20px;
}
```

Titeln på artikeln ska visas på den vänstra sidan.

```
.article-title {
    float: left;
}
```

Knapparna för att ta bort och redigera en artikel ska synas på högra sidan.

```
.actions {
    float: right;
    text-decoration: none;
    padding-top: 5px;
    color: black;
}
```

Själva artikelns innehåll ska placeras fint under rubriken.

```
.content {
    padding-top: 10px;
    clear: both;
}
```

Färgen på länkarna ändrar vi till svart.

```
.actions > a {
    color: black;
}
```

Inläggets datum ska visas i fet text.

```
.date {
    font-weight: bold;
}
```

Texten i foten, längst ner på sidan, ska rätas upp på högra sidan.

```
#footer-bottom {
    text-align: right;
}
```

Slutligen måste vi lägga till följande referens i `head` för att få Roboto typsnittet fungerande på datorer som inte har detta typsnitt installerat.

```
<link href="http://fonts.googleapis.com/css?family=Roboto" rel="stylesheet"
type="text/css" />
```

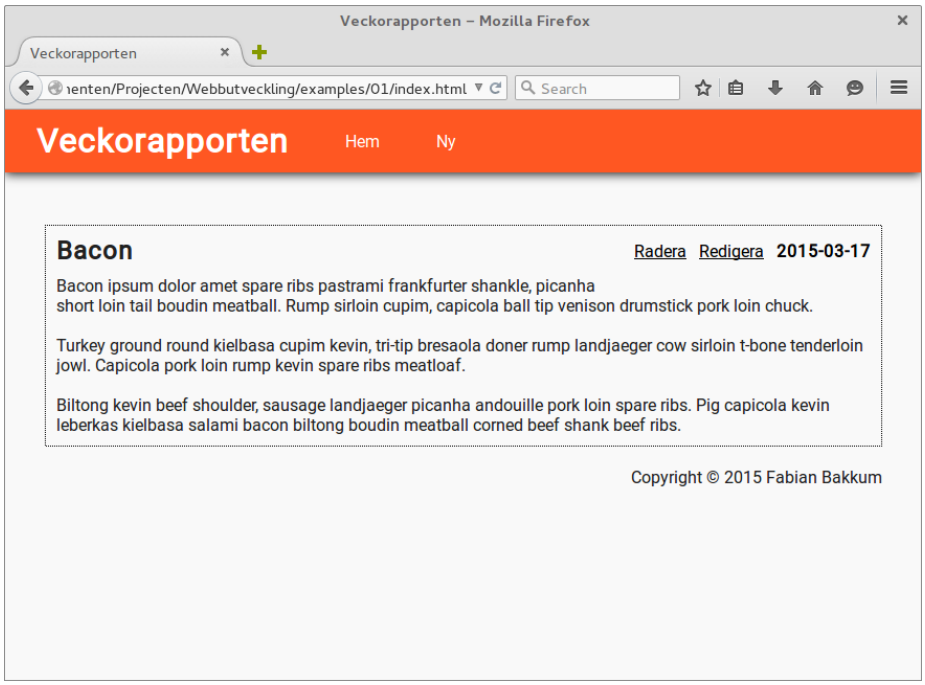

Figure 1: Formgivningens slutresultat

JavaScript

Nu då sidans skelett och hud är klar måste vi ge den en hjärna. Det går att göra på olika sätt. Antingen använder man en form av client-side skriptning eller server-side skriptning. Det går även att blanda dem två programmeringssätt. Skillnaden mellan dessa två är att med hjälp av server-side skripning genererar man utmatningen på servern och med client-side skriptning händer allt i användarens webbläsare. JavaScript är det språket som finns i alla webbläsare och är världens mest använda programmeringsspråk.

JavaScript är ett språk som utvecklades av Sun Microsystems och Netscape. Eftersom JavaScript var ett egendom från Sun Microsystems, behövde Microsoft utveckla deras eget programmeringsspråk, det blev kallad JScript. European Computer Manufacturers Association (ECMA) gjorde JavaScript till en standard och den standardiserade versionen av JavaScript blev ECMAScript.

ECMA standardiserade endast själva språket och syntaxen, men W3C håller i standarden för webb APIn. JavaScript implementerar ECMAScript. När någon pratar om ECMAScript eller ES6 (version sex av ECMAScript) satsar de oftast på grammatiken. Annars pratar man om JavaScript.

JavaScript tillåter oss skapa mera dynamiska sidor. Med hjälp av JavaScript kan man ändra komponenter på sidan utan att behöva friska upp sidan. För att demonstrera detta kan du öppna en konsol i Firefox med knappkombinationen Ctrl+Shift+K. Följande JavaScript ändrar backgrundsfärgen på vårt meny.

```
var header = document.getElementById("header-top");

header.style.backgroundColor = "#009688";
```

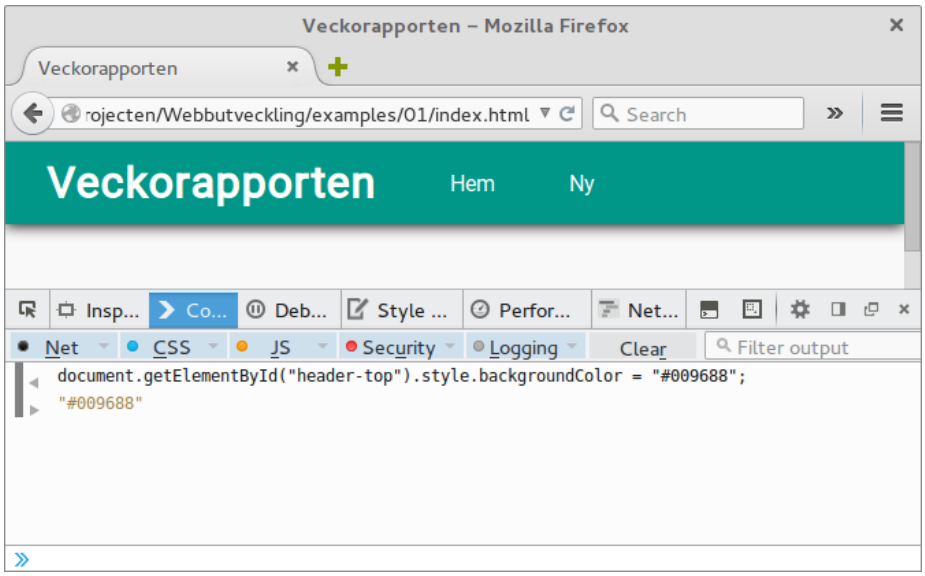

Figure 2: Firefox JavaScript konsol

Databaser

En databas är ett digitalt arkiv. Vi kommer att använda en databas för att spara alla inlägg. Ett Database Management System (DBMS) är ett program som hanterar databasen och tillåter oss att ställa frågor eller *queries* som det kallas. Databasen består av tabeller, vilka i deras tur är uppbyggda av rader och kolumner. Ett exempel på en enkel tabell härnedan.

id	forename	surname	age
1	Knatte	Anka	12
2	Kalle	Anka	18
3	Farbror	Joakim	67

Structured Query Language

Structured Query Language (SQL) är ett språk som används i kombination med *relationsdatabaser* för att förfråga data. Språket utvecklades tidigt på 70-talet av Donald D. Charmberlin och Raymond F. Boyce som jobbade på IBM och är än idag det mest populära sättet att interagera med databaser. SQL finns nästan överallt, mest troligt har du SQL i din byxficka, eftersom de flesta smartphones sparar informationen i en databas kallad SQLite.

SQL är inget svårt språk, men kan bli otroligt komplicerat när databasen innehåller många tabeller och mycket data. SQL på flera tiotals rader är ingen ovanlighet och databasdesign är därför ett yrke i sig. Om vi vill returnera alla personer ur tabellen kan vi använda oss av en SELECT query som den här.

```
SELECT id, forename, surname, age
FROM persons;
```

Om man dock bara vill få ut alla personer som är arton eller över arton kan man använda en WHERE sats som den här.

```
SELECT id, forename, surname, age
FROM persons
WHERE age >= 18;
```

Och med hjälp av ett INSERT uttryck kan vi lägga till nya rader.

```
INSERT INTO persons (forename, surname, age)
VALUES ("Fnatte", "Anka", 12);
```

PHP: Hypertext Preprocessor

PHP är ett populärt skriptningsspråk som finns på nästan alla webbhotell. Initialt hette språket Personal Home Page och bestod endast av ett par skript skrivna i Perl språket, men växte snart till världens mest populära skriptningsspråk för webben. Facebook, Wikipedia och Yahoo! är allihoppa skrivna i PHP. Till skillnad från JavaScript, är PHP ett server-side skriptningsspråk, vilket betyder att allt genereras på servern. Det som är unikt med detta språk är att man kan inpränta PHP kod i ett HTML dokument.

```
<?php
    echo "<p>Bacon ipsum</p>";
?>
```

Ovanstående kod skriver ut *Bacon ipsum*. PHP skrivs inom `<?php ?>` taggar och notera att alla regler slutar med ett semikolon. Till skillnad från tidigare skriven kod måste man ladda upp koden till en webbserver som stöder PHP. Säkerställ att suffixet på filens namn slutar på `.php`. När du har gjort det, kan du surfa till sidan och när allt fungerar som det ska bör du se texten "Bacon ipsum" som en paragraf.

Variabler

Senare kommer vi bland annat att jobba med användarens inmatningar och eftersom vi inte vet vad personen kommer att mata in, behöver vi spara informationen någonstans i minnet så att vi sedan kan använda informationen. Det är vad *variabler* är till för.

Härnedan deklarerar jag en variabel med namnet *name*. Alla variabelnamn börjar med ett dollartecken och man behöver inte deklarera någon datatyp. I nedanstående exempel ser du att jag nu använder mig av enkla citattecken. PHP tillåter både enkla och dubbla citattecken, men de beter sig olika. Punkten används för att *konkatenera* strängarna. Det vill säga att man lägger ihop två strängar.

```
<?php
    $name = 'Fabian Bakkum';
    echo 'Mitt namn är ' . $name;
?>
```

Ibland behöver man spara en lista. Detta kan man göra med så kallade *arrays*. Nedanstående kod skapar en ny array med ett flertal olika frukter och skriver sedan ut alla frukter.

```php
<?php
    $fruits = array("äpple", "banan", "melon");
    echo $fruits[0] . "\t";
    echo $fruits[1] . "\t";
    echo end($fruits);
?>
```

Villkorssatser

if $\overbrace{(\$x == 5)}^{\text{sant}}$ { echo '$x är lika med 5'; }

Den ovannämnda koden är ett exempel på en if-sats och används för att kolla om ett uttryck är sant eller inte och exekverar passande kod.

Läs koden så här: Om $x är lika med 5, skriv ut att $x är lika med 5.

Man kan expandera if-satsen med en else-sats, i vilket man kan skriva koden som ska köras om uttrycket är falskt.

```php
<?php
    $x = 5;
    $y = 10;

    if ($y > $x) {
        echo '$y är större än $x';
    } else {
        echo '$y är inte större än $x';
    }
?>
```

Det finns en del *operatörer* man kan använda sig av:

- == - är lika med
- === - är identiskt med (datatypen är likadan)
- != - är inte lika med
- > - större än
- < - mindre än
- >= - större än eller lika med
- <= - mindre än eller lika med

Funktioner

Även fast du kanske inte har märkt, så har du använd dig av *funktioner*. En av dem funktioner du har stött dig på var `end()` funktionen, vilken *returnerade* det sista elementet i arrayn. Vi kan även skapa våra egna funktioner.

I följande exempel ser du att jag skapade en funktion kallad `sum`, vilken tar två *argument* och returnerar summan av dem två tal.

```
<?php
    function sum($x, $y) {
        return $x + $y;
    }

    echo sum(1, 3);
?>
```

Objektorienterad Programmering

Objektorienterad programmering är en programmeringsmetod, vilket har som mål att organisera koden. Tänk på ett objekt i det verkliga livet, allt är ett objekt, du är ett objekt, din dator, maten du äter, allt. Vi tar en hund som exempel. En hund kan ha flera egenskaper, den kan vara snäll, korthårig, svart och den kan göra olika saker, den kan gå, den kan skälla och den kan äta.

Nu ska vi göra om hunden till kod. En *klass* är en slags mall för att beskriva objektet.

```
<?php
    class Dog {
        private $name;
        private $color;
        private $age;

        function __construct($name, $color, $age) {
            $this->name = $name;
        }

        public function bark() {
            echo 'bark bark, my name is ' . $this->name;
        }
    }

    $scooby = new Dog("Scooby-Doo", "Brown", 13);
    $scooby->bark();
?>
```

Klassen innehåller tre egenskaper som har synligheten *private*. Det innebär att vi endast kan komma åt variablerna inuti själva klassen, men inte utanför. För alla egenskaper (variabler inom klassen) och metoder (funktioner inom klassen) gäller att man måste specificera synligheten. Det finns tre synligheter i PHP.

- **public** – man får komma åt variabeln utanför klassen.
- **protected** – man får inte komma åt variabeln utanför klassen, men föräldraklasser får komma åt variabeln.
- **private** – man får endast komma åt variabeln inom klassen.

Tidigare nämnde jag att alla metoder måste ha en synlighet. Det gäller dock inte för en *konstruktor*. En konstruktor är en metod i klassen som alltid körs när man skapar en ny instans av objektet. Egenskaper inuti klassen kommer man åt med variabeln *this*. Det finns en publik metod `bark` som vi kan anropa utanför klassen. Vi skapar en ny instans av objektet, `scooby` och kastar in tre argument. Slutligen anropar vi bark metoden på objektet.

Webbapplikationens logik

Innan vi börjar programmera behöver vi göra ordning databasen. MySQL erbjuder en textbaserat klient som du kan använda, men de flesta webbhotell ger dig valet att använda PHPMyAdmin, ett grafiskt gränssnitt till MySQL.

När du har hittat konsolen kan du skapa en databas och en tabell med fyra kolumner, id, title, body och date genom nedanstående SQL. Den första kolumnen `id` kommer att få ett unikt värde så att vi kan särskilja posterna och den kommer att räkna upp automatiskt.

Alltså den första posten kommer att ha ett id lika med ett och den andra lika med två osv. Datatypen är satt till *int* med en storlek satt till elva, det vill säga att vi endast får mata in heltal i detta fält. `NOT NULL` säger till att detta fält inte får vara tomt. `AUTO_INCREMENT` gör att den automatiskt räknar upp, vilket gör att vi inte behöver mata in ett id när vi skapar en ny post. `PRIMARY KEY` säger till att detta fält ska användas för att särskilja posterna.

Den andra kolumnen har sin datatyp satt till *varchar* och storleken på den är satt till 128, som gör att titeln är begränsad till 128 tecken. `COLLATE utf8_bin` sätter teckenkodningen till UTF-8.

Den tredje kolumnen har sin datatyp satt till *text*. Den är till för lite längre texter och man behöver inte specificera någon storlek. Samma gäller för datatypen *date*. På den sista raden säger vi till att vi vill använda InnoDB som databasmotor.

```
CREATE TABLE IF NOT EXISTS `journals` (
  `id` int(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  `title` varchar(128) COLLATE utf8_bin NOT NULL,
  `body` text COLLATE utf8_bin NOT NULL,
  `date` date NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
```

Det första vi behöver göra är en sida där man kan skapa nya poster. Därför måste vi skapa en ny fil med namnet `new.php`. Det fullständiga innehållet finns i exempelfilen. Som du ser har jag lagt till en ny HTML form, där användaren kan mata in sin data. Attributet `method` specificerar att data ska skickas i en POST förfrågning. Elementet `label` har fått attributet `for` vilket används för referera till inmatningsfältet. Om man klickar på texten hamnar man i inmatningsfältet.

Elementet `input` används för att skapa fältet där användaren kommer att mata in sin data. Den har fått ett `id` för att kunna referera till den med `for` attributet i labeln. Utöver har den fått `name` attributet, som skickas som nyckelpar i POST förfrågan tillsammans med det inmatade data. Elementet `textarea` används för inmatning av längre texter. Sist i formuläret finns en Spara-knapp.

```
<form method="post">
    <label for="title">Titel:</label>
    <br />
    <input id="title" name="title" type="text" />
    <br /><br />
    <label for="body">Idag har jag:</label>
    <br />
    <textarea id="body" name="body"></textarea>
    <br />
    <input name="submit" type="submit" value="Spara" />
</form>
```

Nu måste vi kunna hantera POST förfrågan som skickas när användaren klickar på Spara-knappen. PHP sparar all data skickat i POST förfrågan i en speciell variabel (array) nämnd `$_POST`. Med `isset` funktionen kan vi kolla om variabeln är satt och har ett värde knutet till sig.

```
<?php
    if ( isset($_POST['submit']) ) {
        // Gör någonting med förfrågan ...
    }
?>
```

Vi kommer att ha några krav, ett av dem är att formuläret ska vara ifylld helt. Här kommer `empty` funktionen till hands. En funktion som returnerar *sanning* om variabeln är tom. Om användaren dock matar in mellanslag kommer funktionen att returnera *falskhet*, alltså kommer variabeln inte att vara tom. Funktionen `trim` kan då användas för att ta bort mellanrum på början och slutet av strängen. Så här kommer villkorssatsen att se ut.

```
if(empty(trim($_POST['title'])) || empty(trim($_POST['body']))) {
    // Informera användaren om problemen
}
```

Inuti if-satsen finner vi tre till if-satser som kollar vad problemet är och informerar användaren sedan om problemet. Om allting är korrekt kan vi lägga in posten i databasen. Innan vi kan göra det, måste vi skapa en databasanslutning. Det finns flera sätt att interagera med MySQL databaser.

- **mysql** – gränssnittet som användes till 2012. Den bör inte användas längre och kommer att tas bort i nyare versioner av PHP.
- **mysqli** – uppföljaren till det föråldrade mysql gränssnittet. Man får interagera med den på ett objektorienterad eller processuellt sätt.
- **PDO** – gränssnittet som vi kommer att använda. Den stöder tolv olika databaser inkluderande MySQL och har sitt fokus på objektorienterat programmering samt att den stöder *nämnda parametrar*.

Nedanstående kod skapar en ny anslutning till databasen genom att skapa ett nytt PDO objekt. Troligtvis kommer databasservern att köras på samma server som webbservern, om det inte är sant bör du byta ut `localhost` mot databasserverns IP adress. Ävenledes måste du byta ut `databasnamn`, `användarnamn` och `lösenord` mot dina egna uppgifter.

```
$dbh = new PDO('mysql:host=localhost;dbname=databasnamn',
               'användarnamn', 'lösenord');
```

Prepared statements

Användaren kan mata in vad som helst, även SQL kod. Essentiellt betyder det att användaren kan göra vad som helst med vår databas. Detta fenomen kallas för *SQL-injektion* och är en av dem vanligaste attacker. Vi kan motverka dessa attacker genom att använda oss av *prepared statements*.

```
$stmt = $dbh->prepare("INSERT INTO journals (title, body, date)
                       VALUES(:title, :body, NOW())");
```

Istället för att konkatenera användarens inmatningar direkt med satsen, så förbereder vi satsen först så att vi sedan kan sitta ihop användarens inmatningar med vår förfråga. På så sätt är vi skyddat mot attacken SQL-injektion.

Nästa steg är att sitta ihop användarens inmatningar med förfrågan. Här säger vi till PDO att byta ut parametern *:title* mot värdet av variabeln *$title*.

```
$stmt->bindParam(':title', $title);
```

Såklart behöver vi sätta variabeln också. Notera återigen att vi använder `trim` funktionen för att ta bort mellanrummet på början och slutet av strängen.

```php
$title = trim($_POST['title']);
```

Slutligen kan vi exekvera queryn och återvända användaren till startsidan om allt gick som det skulle.

```php
$stmt->execute();
header('Location: index.php');
```

Undantagshantering

```php
try {
    $dbh = new PDO('mysql:host=localhost;dbname=veckorapporten',
                   'användaren', 'lösenord');
} catch ( PDOException $e ) {
    echo 'Detta fel inträffade: ' . $e;
}
```

Vad händer om vi inte kan ansluta mot databasservern? Jo, PDO kastar då ett så kallad undantag (*exception*). Undantagshantering görs med ett try-catch block. Vi kan omge koden som kastar ett undantag med try och sedan kan vi fånga undantaget med ett catch block. Notera att PDO som standard endast kastar ett undantag om anslutningen misslyckades. Troligtvis vill du inte skriva ut informationen till sidan, eftersom undantaget innehåller information om databasanslutningen inkluderande användarnamn och lösenord. Om man vill få information om en query som misslyckades måste man sätta ett attribut.

```php
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Visa rapporterna

Nu ska vi göra ordning startsidan (index.php) som kommer att lista alla veckorapporter. Det här blir lite enklare, eftersom vi inte behöver använda oss av prepared statements. För att ta ut all data ur databasen kan vi använda följande SQL.

```php
$sql = "SELECT id, title, body, date FROM journals";
```

Varje rad i tabellen journals är en artikel. Det betyder att vi för varje rad behöver generera en artikel. Vi kan använda oss av en foreach slinga för att generera artiklarna. I koden ser du att vi stänger PHP taggen och fortsätter med lite HTML kod efteråt.

```php
foreach($dbh->query($sql) as $row):
```

På några ställen öppnar vi PHP taggarna igen och skriver ut värdena som erhölls från databasen. Vi gör det för titeln, länkarna, datumet och själva innehållet.

```
<h2 class="article-title"><?php echo $row['title']; ?></h2>
```

Tips: I PHP 5.4+ kan du istället använda `<?= $row['title']; ?>`.

Slutligen stänger vi slingan.

```
<?php endforeach; ?>
```

Redigera rapporter

Som du märkt la vi till rapportens id i redigera och ta bort länken. Skapa en ny fil kallad `edit.php`. Vi kommer att använda id:n för att hämta rapporten från databasen så att vi kan fylla alla fält med nuvarande innehåll. Här kommer vi återigen använda oss av en prepared statement. Den kommer att vara lik sidan där man skapar rapporter.

```
$stmt = $dbh->prepare("SELECT title, body, date
                       FROM journals
                       WHERE id = :id");
```

Rapportens id som skickades med i GET förfrågan kan vi extrahera från `$_GET` variabeln och därefter koppla till :id parametern.

```
$stmt->bindParam(':id', $_GET['id']);
```

Eftersom queryn returnerar endast en rad, behöver vi till skillnad från tidigare inte göra någon slinga. Utan kan vi hämta ut det första värdet med `fetch` metoden.

```
$row = $stmt->fetch();
```

Vi använder attributet `value` för att sätta in texten i fälten. För att uppdatera posten i databasen använder vi en UPDATE sats. Glöm inte WHERE satsen, annars kommer alla rapporter att uppdateras och få samma innehåll.

```
$stmt = $dbh->prepare("UPDATE journals
                       SET title = :title, body = :body
                       WHERE id = :id");
```

Radera rapporter

Sist av allt ska användaren kunna ta bort rapporter. Detta kommer att göras i filen med namnet `delete.php`. Jag gjorde ett enkelt formulär som endast innehåller en Radera-knapp. Om användaren klickar på knappen tar vi bort rapporten och återvänder användaren till startsidan.

```
$stmt = $dbh->prepare("DELETE FROM journals
                       WHERE id = :id");
```