Projektuppgift

DT093G

Webbutveckling II

Albin Rönnkvist

Mittuniversitetet

MID SWEDEN UNIVERSITY

Campus Härnösand Universitetsbacken 1, SE-871 88. **Campus Sundsvall** Holmgatan 10, SE-851 70 Sundsvall.
Campus Östersund Kunskapens väg 8, SE-831 25 Östersund.
Phone: +46 (0)771 97 50 00, Fax: +46 (0)771 97 50 01.

MITTUNIVERSITETET
Avdelningen för informationssystem och -teknologi

Författare: Albin Rönnkvist, alrn1700@student.miun.se
Utbildningsprogram: Webbutveckling, 120 hp
Huvudområde: Datateknik
Termin, år: VT, 2018

Sammanfattning

I detta projekt ska jag skapa en bloggportal med stöd för dynamik med hjälp av PHP och databaskopplingar.

Innehållsförteckning

Terminologi

Akronymer/Förkortningar

PHP	Hypertext Preprocessor
SQL	Structured Query Language

1 Introduktion

1.1 Bakgrund och problemmotivering

Jag ska skapa en bloggportal till ett befintligt företag som heter Vegobeauty.

På webbplatsen ska besökaren kunna registrera ett konto samt kunna göra en inloggning med det kontot. Registreringen ska innehålla inloggningsuppgifter, fullständigt namn och eventuellt övriga uppgifter.

Någonstans på webbplatsen ska det finnas en lista med registrerade användare(visade med namn). Vid klick på någon av användarna ska det visas en lista med alla inlägg som just den specifika användaren publicerat.

Det ska vara möjligt att skapa inlägg på webbplatsen men endast om besökaren är inloggad. Det ska även finnas möjlighet att ändra och radera egna inlägg.

Startsidan ska visa de 5 senaste blogginläggen från samtliga registrerade användare, sorterade på datum(senast överst). Inläggen ska visas med titel, författare, datum för publicering och själva texten/meddelandet.

Jag har även tänkt att hämta data från någon relevant webbplats och lägga dessa i en egenskapad widget.

1.2 Avgränsningar

Jag tänker gå in mest på PHP och databaser i denna rapport för att den inte ska bli för lång. Jag antar att du som läser denna rapport har kunskap inom HTML, CSS, Javascript, digital bildbehandling samt typografi och form för webb.

1.3 Detaljerad problemformulering

Först måste jag ta reda på vilken målgrupp som webbplatsen ska rikta sig till. I detta fall så har företaget(Vegobeauty) en ganska tydlig målgrupp med kvinnor mellan 15-30 som är veganer.

Därefter måste jag skapa en grafisk profil med färger, typsnitt, logga och länkar som matchar målgruppen. Den kommer presenteras med ett moodboard och sedan implementeras på webbplatsen.

Innan jag kan börja göra layouten så måste jag ta reda på vilka sidor som kommer finnas och vad de kommer innehålla. De sidor jag behöver är: start, inloggning, registrering, skapa inlägg, publicerade inlägg samt en sida för att

hantera egna inlägg. Det ska även finnas ett sidofält på startsidan med en lista på alla användare. Jag kommer lägga till en widget på ett passande ställe och eventuellt skapa en sida med en resttjänst så att andra kan hämta data och skapa en egen widget med data från min webbplats.

På **startsidan** kommer de fem senaste inläggen skrivas ut i huvudfältet. Inläggen visas med titel, användarnamn, datum för publicering samt själva texten. I sidofältet kommer en lista med registrerade användare finnas(visade med namn). Inläggen hämtas från en databas i fallande datumordning. Användarna hämtas från en databas i bokstavsordning.

På **inloggningssidan** kommer det finnas ett formulär där en befintlig användare kan logga in med användarnamn och lösenord. Användaren hämtas från en databas och lagras i en sessionsvariabel med användarnamn som variabel.

På **registreringssidan** kommer det finnas ett formulär för registrering där besökaren fyller i användarnamn, lösenord, fullständigt namn och email-address. Användaren lagras i en databas med id, användarnamn, krypterat lösenord, fullständigt namn och e-postadress samt vilket datum användaren registrerades.

På sidan för att **skapa inlägg** kommer det finnas ett formulär där användaren fyller i titel och innehållstexten/meddelandet. Inlägget lagras i en databas tillsammans med ett id, användarens information och datum för publicering. Endast inloggade besökare kommer åt denna sida.

På sidan för **publicerade inlägg** kommer en specifik användares inlägg skrivas ut. Dessa inlägg hämtas från en databas där endast de inläggen med den valda användarens namn/id hämtas.

På sidan för att **hantera egna inlägg** kommer alla inlägg för den inloggade användaren skrivas ut tillsammans med två knappar, en för att ta bort och en för att ändra enskilda inlägg. En användare kan endast ta bort och ändra sina egna inlägg. Inläggen hämtas och sparas/raderas i samma databas.

När jag har all information så kan jag göra en layout för webbplatsen med struktur och navigation. Layouten kommer presenteras med wireframes. En för varje sida med en wireframe för desktopenheter samt en för mobila enheter.

När jag vet vilken layout jag ska ha och vilka sidor som kommer finnas så kan jag börja koda den strukturen med hjälp av PHP. Jag skapar då först alla PHP-filer som ska inkluderas på alla sidor på webbplatsen. Dessa filer är till exempel headern, footern, menyn, konfigurering, innehållet och eventuella sidofält som alltså är genomgående på alla sidor(”inkluderingssidor”). På så sätt får jag en färdigkodad struktur som jag sedan bara kan inkludera på alla sidor utan att behöva kopiera och klistra in massvis med kod på alla sidor. Sedan lägger jag till ”huvudsidorna” och inkluderar alla ”inkluderingssidor” till samtliga.

När alla element är utlagda så måste jag ge de funktionalitet och ansluta de till en databas. Men för att jag ska kunna ansluta till databasen så måste det självklart finnas en. Därför tänker jag skapa en databas med två olika tabeller där jag kan lagra användare och inlägg. För att på bästa sätt visa tabellernas värden och relationer så kommer jag skapa ett ER-diagram. I tabellen för användare ska det lagras: unikt id(med auto_increment för att slippa fylla i id), unikt användarnamn, krypterat lösenord, fullständigt namn och email-address samt vilket datum användaren registrerades. I tabellen för inlägg ska det lagras: unikt id, användarnamn, titel, innehållstext och datum för publicering.

Projektet kommer lösas objektorienterat så jag måste även göra klassdiagram för varje klass som ska ingå i webbplatsen. Det kommer finnas en klass som hanterar användare med registrering och inloggning samt en klass som hanterar inläggen med hämtning och lagring.

Sist måste jag koppla ihop klasserna med elementen för att ge elementen dess funktionalitet. T.ex. formulären.

2 Teori

Hypertext Preprocessor(PHP) är ett skriptspråk som ofta används till att driva dynamiska webbplatser. En statisk webbplats har sitt innehåll direkt i HTML-koden medan en dynamisk webbplats innehåll hämtas från t.ex. en databas eller besökarens formulärdata. Fördelen med dynamiska webbplatser är att det är mycket enklare att uppdatera och sortera innehållet. Du kan t.ex. koda en hemsida med ett användargränssnitt där andra användare utan kodningskunskaper kan både mata in data och hämta/presentera data. PHP underlättar även för dig som kodare eftersom du kan återanvända kod och inkludera de smidigt på olika webbplatser och undersidor.[1]

Objektorienterad programmering(OOP) är en typ av programmeringsmetod, ett sätt att tänka. Man delar in programmet i mindre delar för att på så sätt kunna ändra och återanvända koden mycket lättare. Du börjar med att skapa en klass. En klass är som en mall. Det är en samling av data för ett koncept, som har attribut och metoder för den datan. Ett objekt är en instans av en klass. Till exempel har klassen ”Katt” attributen ”färg” och ”namn” samt metoden ”jama()”. Ett objekt av klassen ”Katt” kan ha namnet ”Stefan” och färgen ”Blå”, vi kan även be katten jama genom att anropa metoden ”jama()”. På så sätt har vi skapat ett objekt som är en instans av klassen katt. Vi kan då återanvända denna klass till att skapa fler katter utan att behöva upprepa kod.**Structured query language(SQL)** är ett standardiserat programspråk för att lagra, hämta och ändra data i en relationsdatabas. Man använder alltså språket för att kommunicera med en databas. Några vanliga databasmotorer som stödjer SQL är: Oracle, Microsoft SQL Server, Microsoft Access, MariaDB, DB2 etc.[2] [3]

En **databas** är en organiserad samling av information som är strukturerad på ett sätt där det ska vara så lätt som möjligt att söka och hämta specifik information samt kunna ändra den. I detta projekt kommer jag jobba med en **relationsdatabas** vilket är en databas som är organiserad i relationer. Relationerna kallas även tabeller och de består av kolumner(fält) och rader(tupler). En tabell beskriver ofta en specifik typ av information, t.ex. personer. Kolumnerna kan man se som olika värden som beskriver tabellen mer specifikt. T.ex. så kanske man vill veta personens namn, personnummer och ort. De värdena beskrivs i kolumnerna. Det är i raderna som sedan data om varje enskild person lagras där man utgår från kolumnernas värden. All data i en specifik kolumn måste vara av samma typ. I en rad kan det då ha lagrats: Albin, 9712257864, Stockholm, alltså namn, personnummer och ort. Lägger jag till en till person så lagras den på en ny rad. Det går att skapa flera tabeller och koppla ihop dessa. För att hantera databasen använder man sig ofta av språket SQL. [4]

Structured query language(SQL) är ett standardiserat programspråk för att lagra, hämta och ändra data i en relationsdatabas. Man använder alltså språket

för att kommunicera med en databas. Några vanliga databasmotorer som stödjer SQL är: Oracle, Microsoft SQL Server, Microsoft Access, MariaDB, DB2 etc. [5]

3 Metod

Jag kommer skapa mitt moodboard i Photoshop för att ha en tydlig grafisk profil att utgå från när jag sedan kodar webbplatsen.

Layouten kommer presenteras med wireframes som jag skapar på webbplatsen ”Wireframe.cc”. Med hjälp av wireframes går det mycket snabbare att koda layouten.

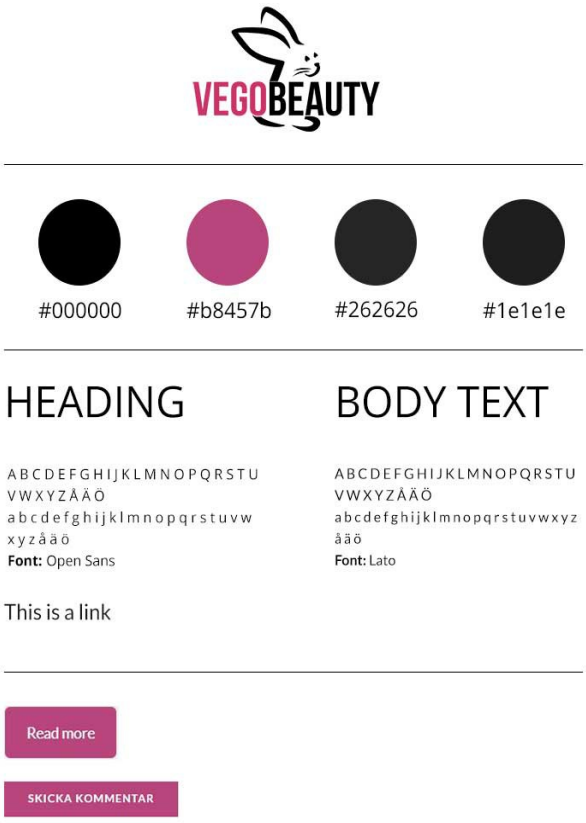
Jag kommer skapa databasen i PHPMyAdmin och tabellerna kommer presenteras med diagram som jag skapar på webbplatsen draw.io.

Kodningen kommer ske i Visual Studio Code där jag löser all funktionalitet med objektorienterad programmering och PHP. Jag kommer göra klassdiagram som jag skapar i draw.io.

4 Konstruktion

Grafisk profil

Jag skapade en grafisk profil i Photoshop där jag visar logga, typsnitt, färger och knappar. Jag ville att bloggportalen skulle likna den redan befintliga webbplatsen för företaget så att besökare skulle känna igen sig även när de besöker bloggportalen. Nedan visas den grafiska profilen i Figur 1.



Figur 1

Layout

Jag skapade en layout med wireframes. En för varje sida samt en var för mobila enheter och desktop-enheter. Alla wireframes ligger på sista sidan i denna rapport med namn för att förtydliga vad de beskriver(Bilagor(Wireframes)).

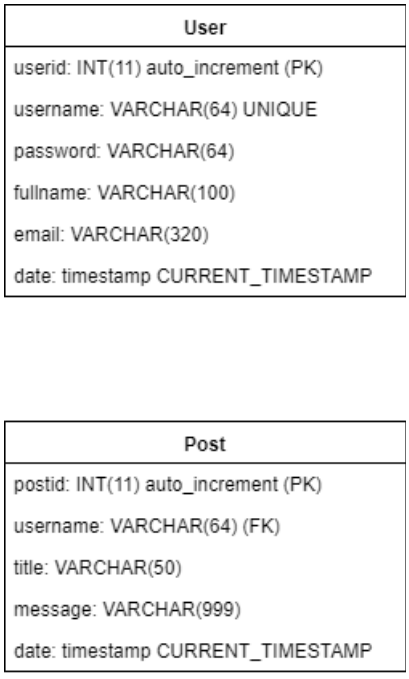
Databas

Jag skapade en databas som lagrar användare och inlägg i PHPMyAdmin. För användare skapade jag en tabell som lagrar id, användarnamn, lösenord, fullständigt namn, e-postadress och datum vid registrering. Id eller ”userid” är primärnyckel och därför unik. För att slippa fylla i ett id vid registrering så får användaren automatiskt ett id när de registrerar sig med hjälp av auto_increment. Användarnamnet är unikt så flera användare kan inte ha samma användarnamn. Datum för registrering blir automatiskt ifyllt med hjälp av CURRENT_TIMESTAMP som lagrar exakta tiden vid registrering.

För inlägg skapade jag en tabell som lagrar id, användarnamn, titel, meddelande och datum vid publicering. Id för inläggen eller ”postid” är pirmärnyckel och lagras på samma sätt som userid. Datum lagras även på samma sätt som i user-tabellen. I denna tabell är användarnamn eller ”username” ett främmande nyckelattribut med ett referensintegritetsvillkor som kopplas till username i user-tabellen. Det betyder att det inte går att skapa ett inlägg i post-tabellen med ett användarnamn(username) som inte finns user-tabellen.

Jag har använt mig av dataintegritetsvillkor, alltså vilken typ av data som får fyllas i(t.ex. INT/VARCHAR) och hur många tecken det får lagras i varje kolumn. I PHP kommer jag sedan kryptera lösenord och göra en koll så att inga tecken i något fält stör SQL-koden mm.

I Figur 2 visas ett ER-diagram som beskriver dessa tabeller med deras dataintegritetsvillkor, referensintegritetsvillkor och samband.



Figur 2

Struktur – kodning

Jag strukturerade webbplatsen med PHP. I include-filen ligger alla PHP-filer som ska inkluderas på alla sidor. Dessa filer är: config, footer, header, mainmenu, sidebar och sidebarempty.

I config startar jag en session, inkluderar alla klasser, sätter på error-rapportering och ställer in vilken databasserver jag vill ansluta till. Denna fil läggs längst upp i headern som i sin tur läggs längst upp på alla sidor. Detta är viktigt eftersom session_start()-funktionen mm måste inkluderas först i koden, innan HTML-koden.

I footer-filen lägger jag footern med dess innehåll samt scripten och avslutande taggar för containern, body och html.

I header-filen inkluderar jag config och startar min HTML-kod med head, container, header och huvudsektionen för innehåll. I headern inkluderas mainmenu-filen.

I mainmenu-filen ligger huvudmenyn som visar olika länkar beroende på om besökaren är inloggad eller inte. En koll görs om en sessionsvariabel existerar eller inte, om den existerar så är besökaren inloggad vilket ger den alternativ som att t.ex. logga ut. Jag har gjort en meny för desktopenheter och en för mobila enheter. Dessa visas och döljs med hjälp av media queries.

I sidebar-filen skriver jag ut alla användare som hämtas från metoden getUsers() i Users-klassen. Jag går in på klasserna och dess funktionalitet senare i rapporten men i stora drag så returnerar getUsers() en associativ array med alla användare i databasen. För att skriva ut denna array så måste jag först skapa en instans av klassen och sedan anropa metoden som returnerar arrayen till den instansen. När jag nu har arrayen så kan jag skriva ut den med en foreach-loop. Jag skriver ut varje användare som en länk, länken tar användarnamn som unikt länknamn för varje användare. Denna länk används senare på userposts-sidan.

I sidebarempty-filen skriver jag ut en länk för att logga ut. Länken visas endast om besökaren är inloggad och detta görs med samma teknik som i mainmenu-filen. Vid klick på länken så anropas metoden logOutUser() från Users-klassen som loggar ut användaren och därefter hänvisas besökaren till startsidan med hjälp av kodsnutten: `header("Location: index.php");`.

Huvudfilerna eller alla webbplatsens sidor är: createpost, handlepost, index, login, register, update och userposts. På alla dessa sidor inkluderar jag include-filerna med include()-påståenden.

På index-sidan skrivs de fem senaste inläggen ut, sorterade efter datum. Arrayen med inlägg hämtas från metoden getPosts() i klassen CreatePost och skrivs ut med en foreach-loop.

På login-sidan görs det först en koll om besökaren är inloggad eller inte. Är besökaren inloggad så skrivs det ut ett meddelande som säger att den redan är inloggad. Annars om besökaren inte är inloggad, alltså om det inte existerar någon sessionsvariabel (`if(!isset($_SESSION["username"]))`) så skrivs ett formulär ut för inloggning. Vid klick på logga in så görs sedan en koll om uppgifterna är korrekta och existerar i databasen. Detta görs i metoden loginUser() i klassen Users. Metoden returnerar true om uppgifterna stämmer och då loggas användaren in, annars skrivs ett felmeddelande ut.

På register-sidan finns ett formulär för registrering, alla fält måste fyllas i och det kollar jag med ”required”-attributet i input-fälten. Om besökaren fyller i alla fält och trycker på registrera-knappen samt godkänner att dess data lagras så görs det först en koll om användarnamnet redan finns. Det görs med metoden takenUsername(). Jag kollar även om e-postadressen är giltig med ”FILTER_VALIDATE_EMAIL”-filtret. Finns inte användarnamnet och alla fält är korrekt ifyllda så lagras användaren med metoden storeUser() i Users-klassen. Jag meddelar användaren om lagringen gick igenom eller om den misslyckades.

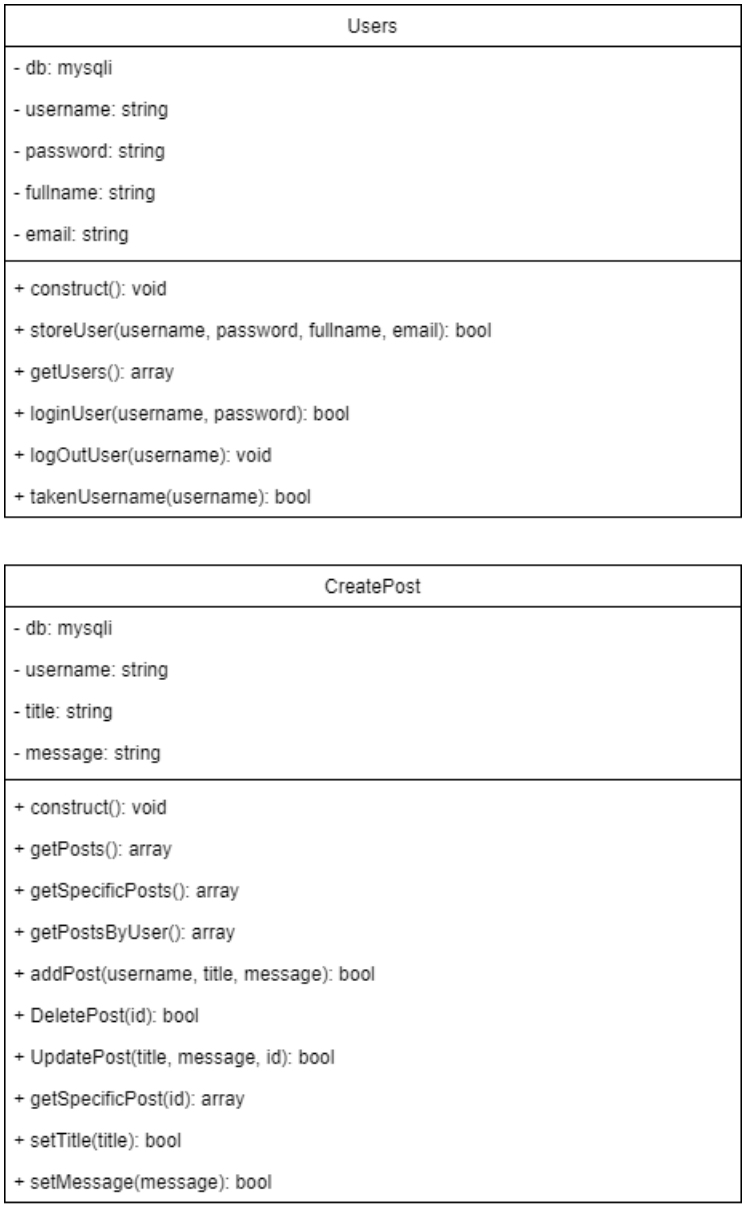
På createpost-sidan görs först en koll om besökaren är inloggad eller inte och det görs med samma teknik som på login-sidan. Är besökaren inloggad så skrivs ett formulär ut för att skapa inlägg. Inläggen lagras med hjälp av metoden addPosts() i klassen CreatePost. Eftersom det bara kan lagras 999 tecken i message-kolumnen i post-tabellen så sätter jag en gräns på textarean med attributet ”maxlength”.

På handlepost-sidan kan användare radera eller uppdatera sina inlägg. Först görs en koll om besökaren är inloggad eller inte. Är besökaren inte inloggad så skrivs ett inloggningsformulär ut. Är besökaren inloggad så skrivs alla dess inlägg ut i datumordning. Inläggen hämtas från getSpecificPosts()-metoden i klassen CreatePost och skrivs ut med en foreach-loop. Även två knappar(länkar) skrivs ut till varje enskilt inlägg, en för att radera och en för att uppdatera. Vid klick på radera-länken så lagras det inläggets unika ”postid” i variabeln $id som sedan raderas med hjälp av DeletePost()-metoden i klassen CreatePost. Vid klick på redigera-länken så skickas användaren vidare till update-sidan där inläggets unika postid lagras i variabeln $id. Inlägget hämtas sedan med hjälp av getSpecificPost()-metoden och skrivs ut i ett formulär för redigering. Titeln skrivs ut i value-attributet i input-fältet på detta vis: `value="<?= $title; ?>"` . Meddelandet skrivs ut med samma teknik fast i textfältet. Användaren kan nu se exakt hur inlägget ser ut och redigera det för att sedan ladda upp det igen. Vid klick på uppdatera så uppdateras inlägget med hjälp av UpdatePost()-metoden.

På userposts-sidan visas alla inlägg från en specifik användare. Vid klick på en länk i sidebar-filen så tas användarnamnet som länknamn. Länken som är användarnamnet används senare för att hämta inlägg med hjälp av metoden getPostsByUser().

Klasser

Nästan all funktionalitet kommer hanteras av klasser som sedan instansieras på ”huvudsidorna”. De klasserna jag skapat är ”CreatePost” som hanterar alla inlägg och ”Users” som hanterar alla användare. I dessa klasser finns metoder som utgör olika funktionaliteter. Alla metoder visas i klassdiagrammen i Figur 3.



Figur 3.

I Users-klassen finns det 6 metoder. *Construct()*-metoden ansluter till databasen på MySQL-servern.

storeUser()-metoden lagrar alla uppgifter som besökaren fyller i registreringsformuläret. Först tar metoden bort alla specialtecken från strängarna i input-fälten så att de kan användas i en SQL-fråga. Sedan krypteras lösenordet med hjälp av Crypt()-funktionen som returnerar en hashad sträng. Jag har använt mig av hashingalgoritmen Blowfish som hashar lösenordet med en saltsträng. Om man inte krypterar lösenordet så ligger det synligt i databasen vilket inte är säkert. Skulle någon lyckas ta sig in i databasen så kan den se alla användares lösenord. När metoden har krypterat lösenordet så förs alla värden in i user-tabellen med en INSERT-fråga. Metoden returnerar sedan en bool, alltså om värdena lagrades eller inte.

getUsers()-metoden hämtar alla användare i namnordning med en SELECT-fråga. Metoden returnerar sedan en associativ array med alla rader som hämtades i SELECT-frågan.

loginUser()-metoden loggar in en befintlig användare. Först tas alla specialtecken bort. Sedan hämtas lösenordet från user-tabellen med en SELECT-fråga där användarnamnet är samma som besökaren fyller i. Sedan kollar jag om användarnamnet och lösenordet finns i databasen med hjälp av num_rows()-funktionen som returnerar antalet rader som SELECT-frågan gav. Är det fler rader än 0 så finns användaren i databasen, annars inte. Finns det fler än 0 rader så lagrar jag raden som en associativ array i en variabel med hjälp av fetch-assoc()-funktionen. Sedan lagrar jag lösenordet från den arrayen i en egen variabel. Om lösenordet i databasen är samma som användaren fyller i så skapas en sessionsvariabel med användarnamnet som värde. Metoden returnerar sedan en bool, alltså om användaren loggades in eller inte.

logOutUser()-metoden tar bort sessionsvariabeln som användaren använder med hjälp av unset()-funktionen. På så sätt loggar jag ut användaren.

takeUsername()-metoden kollar om besökarens angivna användarnamn vid registrering redan finns i databasen. Detta kollas med samma teknik som i *loginUser()*-metoden. Först hämtar jag alla användarnamn med en SELECT-fråga, sen kollar jag om några rader returneras av den frågan. Om fler än 0 rader returneras så finns användarnamnet, annars inte. Metoden returnerar sedan en bool, alltså om användarnamnet finns eller inte.

I CreatePost-klassen finns det 10 metoder. *Construct()*-metoden ansluter till databasen på MySQL-servern.

getPosts()-metoden hämtar de fem senaste inläggen från post-tabellen med en SELECT-fråga. Med hjälp av mysqli_fetch_all()-funktionen hämtas sedan alla rader som SELECT-frågan gav och returnerar de som en associativ array.

getSpecificPosts()-metoden hämtar alla inlägg från en inloggad användare med en SELECT-fråga. Alla inlägg där användarnamn är samma som sessionsvariabelns värde hämtas. Sessionsvariabeln värde är användarnamnet. Raderna som hämtas returneras sedan som en associativ array likt metoden ovan.

getPostsByUser()-metoden hämtar inlägg från en specifik användare. Metoden är lik ovanstående med en skillnad i SELECT-frågan. Alla inlägg där användarnamn är samma som `$_GET['userID']` hämtas. I userID ligger användarnamnet.

setTitle()-metoden kollar om titeln är en tom sträng eller inte. Om det inte är en tom sträng så tas specialtecken bort så att den kan användas i en SQL-fråga. Metoden returnerar en boolean, alltså om det är en tom sträng eller inte. *setMessage()*-metoden fungerar på exakt samma sätt fast den kollar meddelandet.

addPost()-metoden lagrar ett inlägg i post-tabellen med hjälp av en SELECT-fråga. Först kollar metoden om titel och meddelande är ifyllda med hjälp av de två ovanstående metoderna. Sedan lagras de ifyllda värdena i ”skapa inlägg”-formuläret tillsammans med användarnamn som är sessionsvariabeln. Alla värden lagras med en INSERT-fråga. Metoden returnerar sedan en boolean, alltså om inlägget lagrades eller inte.

DeletePost()-metoden raderar specifika inlägg med hjälp av en DELETE-fråga. Endast inlägg med ett unikt postid raderas. Alla inlägg lagras med ett postid, $id är en variabel som håller `$_GET['deleteID']` som i sin tur håller postid. Metoden returnerar sedan en boolen, alltså om inlägget raderades eller inte.

UpdatePost()-metoden uppdaterar ett specifikt inlägg med hjälp av en UPDATE-fråga. Först görs en koll om titel och meddelande är ifyllda. Sedan uppdateras inlägget och nya strängar från input-fälten lagras tillsammans med datum vid uppdatering. Metoden returnerar en boolean, alltså om inlägget uppdaterades eller inte.

getSpecificPost()-metoden hämtar ett specifikt inlägg med hjälp av en SELECT-fråga. Raden med samma postid som inläggets $id hämtas, som sagt innan så ligger inläggets postid i variabeln $id. Metoden returnerar sedan en array.

Resttjänst

Jag har skapat en resttjänst som utvecklare kan använda sig av om de vill skapa en widget eller liknande till en annan hemsida. Först skapade jag en simpel klass(RestPost) med en konstruktor och en metod. Konstruktorn ansluter till databasservern. Metoden getWidgetPosts() hämtar inlägg från post-tabellen med en SELECT-fråga där antal inlägg som ska skrivas ut begränsas av en variabel som anges i URL:en. Variabeln som anges måste vara en siffra. Anges

inget i URL:en så läses alla inlägg ut(max 999). Resultatet av frågan, alltså alla rader som hämtades sparas i en associativ array. Den arrayen konverteras sedan till JSON-format och returneras som en JSON-sträng.

På restservice-sidan instansierar jag RestPost-klassen och skriver ut JSON-strängen som getWidgetPosts()-metoden returnerar. `header('content-type: application/json; charset=utf-8');` anger att innehållet ska vara i JSON-format.

På developers-sidan finns en genomgång på hur man använder resttjänsten och det finns även ett exempel där den används i en widget. Jag har använt mig av ett AJAX-anrop för att hämta data från servern som sedan skrivs ut på HTML-sidan.

Övrig funktionalitet

Jag har skapat funktionalitet med Javascript som visar hur många tecken användaren kan skriva. Det är en räknare som returnerar längden på textareans sträng. Är längden på strängen längre eller lika med längden som är angivet i ”maxlength”-attributet i textarean så meddelas besökaren att den inte kan skriva fler tecken. Annars skrivs längden på strängen ut bredvid max antal tecken som får skrivas.

5 Resultat

Jag har nu skapat en bloggportal med stöd för dynamik som riktar sig mot en specifik målgrupp.

Den grafiska profilen är skapad utifrån det redan befintliga företaget Vegobeauty och deras målgrupp. Layouten är optimerad för både stora och små skärmar(desktop/mobil).

Webbplatsen är responsiv och fungerar på både mobil- och desktop-enheter samt i de mest använda webbläsarna. Både HTML och CSS validerar korrekt utan fel.

All funktionalitet som nämndes i problemformuleringen är implementerad och fungerar korrekt. Det finns möjlighet att registrera ett konto och logga in samt logga ut. Inloggad användare kan skapa inlägg och redigera samt radera sina egna inlägg. De fem senaste inläggen skrivs ut på startsidan och i sidofältet skrivs det ut en lista med användare. Vid klick på en användare visas dess inlägg. Även extra funktionalitet finns på webbplatsen.

6 Slutsatser

Detta var ett av de roligare projekten jag gjort och jag tycker det har flutit på ganska bra utan några större bekymmer. Webbplatsen fungerar felfritt enligt de tester jag gjort och uppfyller de krav som ställdes i problemmotiveringen. Jag är ganska nöjd med resultatet men sen finns det saker som kan vidareutvecklas.

Jag tycker webbplatsen är för simpel just nu och har inte tillräckligt mycket funktionalitet. Jag skulle kunna bygga vidare på den och lägga till mer saker som gör webbplatsen roligare och bättre för användare, t.ex. kommentarsfält och gilla-knappar. Med kunskapen jag fått från detta projekt så känns det som att sådan funktionalitet inte skulle vara någon större svårighet att få till.

Jag är inte helt nöjd med designen. Den skulle kunna vara mer tilltalande men jag valde att lägga mer fokus på funktionalitet i detta projekt.

Sammanfattningsvis så har arbetet gått bra och resultatet blivit hyfsat. Webbplatsen kan byggas vidare och designen kan vara mer tilltalande. Projektet har givit mig många nya kunskaper och det har varit kul att jobba med en dynamisk webbplats.

Källförteckning

[1] Wikipedia, ”PHP” https://sv.wikipedia.org/wiki/PHP Publicerad 2018-02-09. Hämtad 2018-03-05

[2] Databasteknik, ”Grunder om objektorientering” http://www.databasteknik.se/webbkursen/oo-grunder/index.html Publicerad 2005-07-16. Hämtad 2018-03-05

[3] Wikipedia, ”Objektorienterad programmering” https://sv.wikipedia.org/wiki/Objektorienterad_programmering Publicerad 2016-09-12. Hämtad 2018-03-05

[4] Databasteknik, ”Introduktion till databaser och databashanterare” http://www.databasteknik.se/webbkursen/databaser/ Publicerad 2015-07-18. Hämtad 2018-03-05

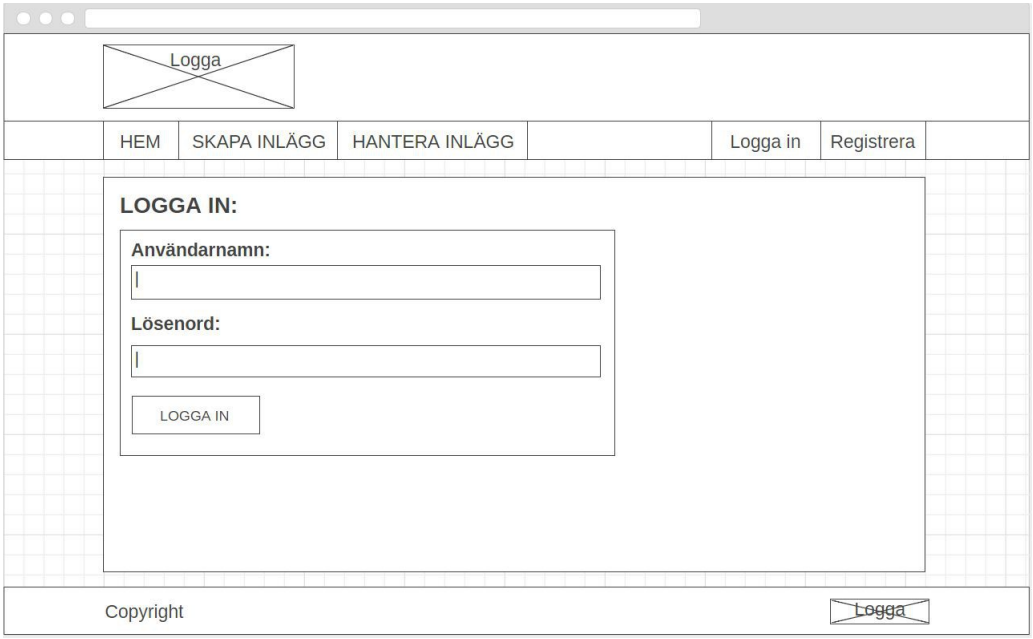
[5] SQLCourse, ”What is SQL?” http://www.sqlcourse.com/intro.html Hämtad 2018-03-05

Bilagor(Wireframes)

Wireframes:

Startsida – Desktop.

Registrera – Desktop.

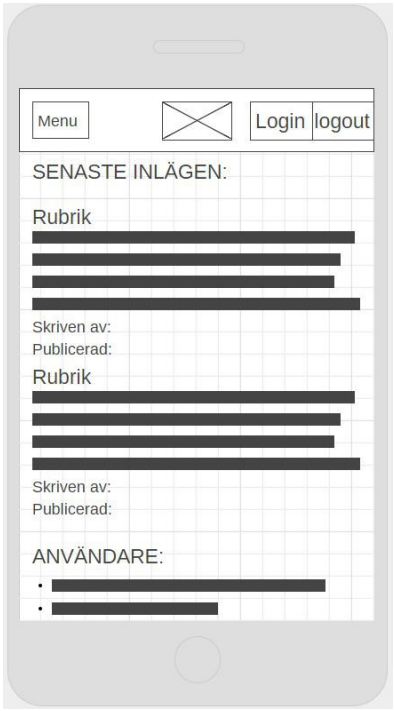


Logga in – Desktop.

Skapa inlägg – Desktop.

Hantera inlägg – Desktop.

Publicerade inlägg av specifik användare – Desktop.



Startsida – Mobil. Alla sidor använder sig av denna layout. Formulär och inlägg fyller ut fulla bredden och sidofältet flyter in under huvudsektionen.