Just Another Webshop

Marcus Börjesson

Gustav Lindqvist

David Klar

1. Metod

Projektet har utvecklats med verktygen Git och GitHub för att hålla ordning på allt som behöver göras och vad som ska bli gjort med hjälp av GitHubs issue-tracking.

Till en början togs ER-modeller och skisser på vilka funktioner och funktionalitet som den färdiga webbshoppen skulle inehålla. Utifrån det började det sedan paralellt utvecklas en databas-modul och en router-modul som senare i projektet byggdes ihop. Utifrån vad som behövdes för att få till ut presentation utvecklades fram funktioner i `functions.php`.

Kvalitétskontrollering har gjorts genom att kontinuerligt granska varandras kod och komma med kritik och kommentarer. Detta har skötts till stor del via Issues i GitHub men även via chatt och konversationer. Det mesta av utvecklandet har skett individuellt och ej i grupp och gruppen har till största del arbetat utifrån Git.

2. Resultat

Gruppen har fått fram en webbshopp som enligt vår tolkning uppfyller alla de krav som har lagts fram i kravspecifikationen. Och en hel del andra funktioner som ej stod med.

En live-variant av webbshoppen finns på http://hockeygear.lindqvist.io inloggning för test-konto är `lindqvist.gustav@gmail.com` med lösenord `test`.

Fullständig historik och resultat finns på http://github.com/reedyn/Just-Another-Webshop

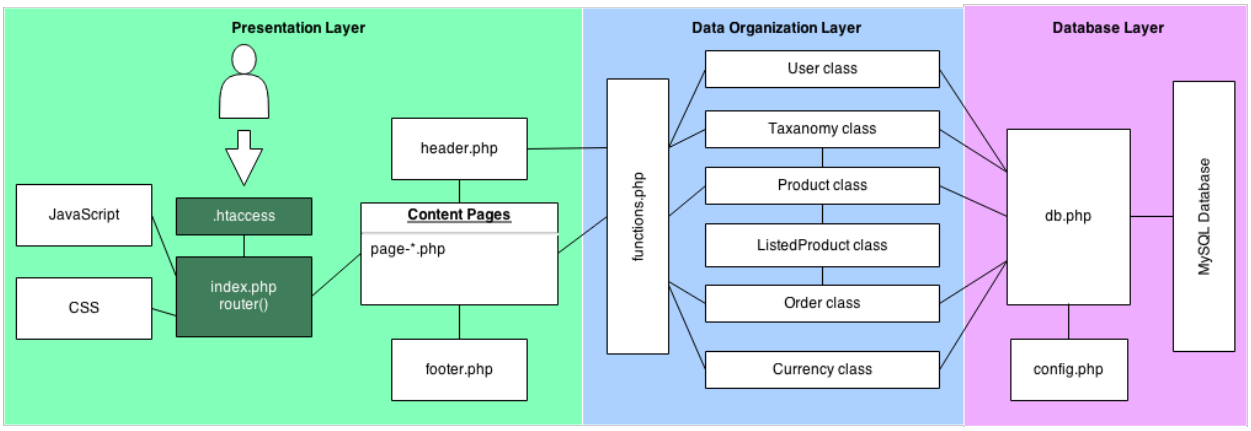
Projektet är uppdelat i två större grupper, presentation och data-hantering.

Upplägget är tänkt att göra det enkelt att utveckla mer funktionalitet senare, som ett komplett system för teman (en tanke var att göra själva presentationen i `page-*.php` filerna liknande så som WordPress gör med sin *"the loop"* vid en sådan utveckling senare).

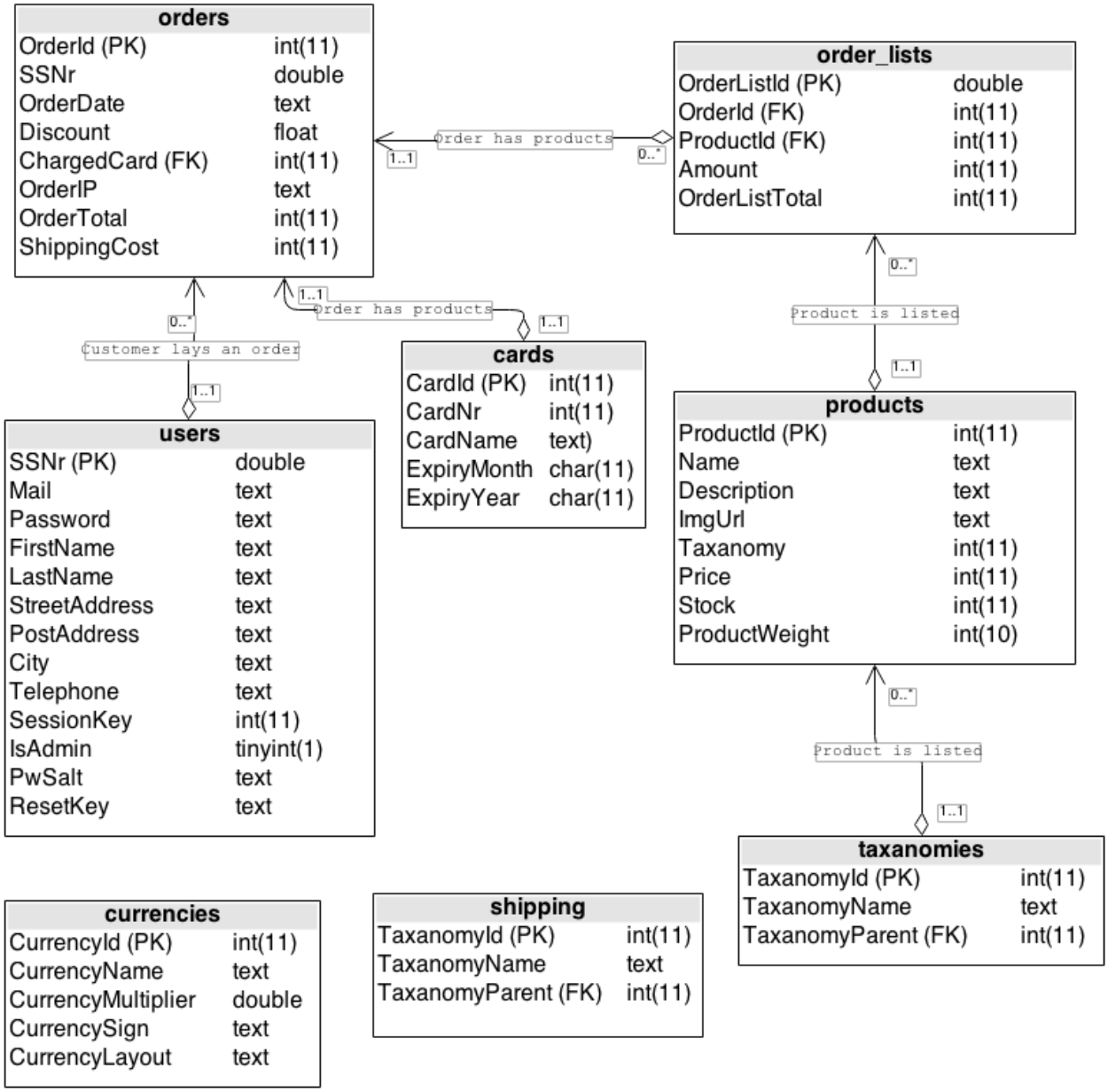
På grund av detta är koden i ett flertal filer separerade, som till exempel databas-inloggning lever i `config.php` istället för att skriva in direkt vid skapandet av *Database-objektet*.

```
index.php
.htaccess
/core/
    This folder contains the files that include the real programming, the router in
init.php, database in db.php, functions in functions.php
/content/
    All template-files that include validation and presentation are here.
    /img/
        Folder for uploaded product images
    /themes/
        Folder for themes, We only have one theme right now.
        /default/
            /css/
            /js/
            /fonts/
            /img/
```

Systemet sitter ihop på följande sätt:



Databasen är uppbyggd på följande sätt:



2.1. Resultat enligt kravspecifikation

2.1.1. Betyg 4

2.1.1.1. Struktur för produkter (Taxonomies)

Kategorier (Taxonomies) för produkter är byggda i en snygg hierarkisk struktur där i systemet inte finns begränsningar för hur många undernivåer man kan få ut (Dock begränsningar i gränssnittet). Funktionen för utskrift av kategorierna bygger på en rekursiv funktion för att skriva ut undermenyer i en struktur liknande nedan:

- Helmets
    - Hockey Helmets
    - Cool Hockey Helmets
        - Super Cool hockey helmets
    - Goalie Helmets
- Skates
    - Hockey Skates
    - Figure Skates
        - Super skates

Rekursiva funktionen ser ut på följande sätt:

```
function listAdminTaxanomies(){
    $taxanomies=$GLOBALS['db']->dbGetTaxanomyTree();
    if($taxanomies){
        echo '<div class="panel panel-primary">
                  <div class="panel-heading">Categories</div>
                  <div class="panel-body">
                    <ul class="list-unstyled">';
                        foreach($taxanomies as $taxanomy){
                            echo '<li>'.$taxanomy['TaxanomyName'].' <a
href="/admin/categories/'.$taxanomy['TaxanomyId'].'" class="btn btn-default
btn-xs">Edit</a>
                            </li>';
                            recursiveAdminTaxanomy($taxanomy['TaxanomyChildren']);
                        }
                    echo '<li><a href="/admin/categories/new" class="btn
btn-primary">Add category</a>
                    </ul>
                  </div>
                </div>';
    }
}
```

```
function recursiveAdminTaxanomy($taxanomy){
    if($taxanomy){
        echo '<ul class="list-styled">';
        foreach($taxanomy as $child){
            echo '<li>'.$child['TaxanomyName'].' <a
href="/admin/categories/'.$child['TaxanomyId'].'" class="btn btn-default
btn-xs">Edit</a>
                            </li>';
            recursiveAdminTaxanomy($child['TaxanomyChildren']);
        }
        echo '</ul>';
    }
}
```

2.1.1.2. Transaktioner i databasen

Transaktioner används primärt för borttagning av Ordrar där ListedProducts måste tas bort först.

```
public function dbDeleteOrder($OrderId) {
    $this->autocommit(false);
    if($this->query("DELETE FROM order_lists WHERE OrderId in ($OrderId)")){
        if($this->query("DELETE FROM orders WHERE OrderId in ($OrderId)")){
            $this->autocommit(true);
            return true;
        }else{
            $this->rollback();
            return false;
        }
    }else{
        $this->rollback();
        return false;
    }
}
```

2.1.1.3. Klasser

Klasser används för Database, User, Admin (som ärver User), Order, ListedProduct, Product och Taxonomy.

```
class User{
    public $SSNr;
    public $Mail;
    public $Password;
    public $FirstName;
    public $LastName;
    public $StreetAddress;
    public $PostAddress;
    public $City;
    public $Telephone;
    public $SessionKey;
    public $IsAdmin;

    public function
__construct($SSNr,$Mail,$Password,$FirstName,$LastName,$StreetAddress,$PostAddress,$Cit
y,$Telephone,$SessionKey,$IsAdmin) {
        $this->SSNr         = $SSNr;
        $this->Mail         = $Mail;
        $this->Password     = $Password;
        $this->FirstName    = $FirstName;
        $this->LastName     = $LastName;
        $this->StreetAddress= $StreetAddress;
        $this->PostAddress  = $PostAddress;
        $this->City         = $City;
        $this->Telephone    = $Telephone;
        $this->SessionKey   = $SessionKey;
        $this->IsAdmin      = $IsAdmin;
    }
}
```

2.1.1.4. Sortering och filtrering i administrations GUI

Sortering och filtrering finns tillgängligt för alla typer av enheter i administrationsgränssnittet och görs via JavaScript. På detta vis behöver sidan inte laddas om för att möjliggöra filtrering utan allt kan göras direkt i webbläsaren utan fördröjning.

2.1.2. Betyg 5

2.1.2.1. Captcha

För Captcha används Googles reCaptcha.

2.1.2.2. Bildhantering

Bildhantering används klassen ImageManipulator som är färdigskriven för att hantera all typ av bildhantering. I projektet används den för att skala ner bilder med bibehållet höjd-bredd förhållande.

2.1.2.3. Registrering och kontohantering

Registrering av ett konto kan ske på två sätt.

1. Användaren skapar sitt eget konto och har möjlighet att bestämma lösenord själv vid registrering.
2. En administratör skapar ett konto, en engångskod skickas då till e-mail addressen som angavs vid registrering som gör det möjligt för användaren att bestämma lösenord för sitt konto. På detta vis finns det aldrig någon möjlighet för någon annan än användaren (eller de som har access till mailen) att få tag på lösenordet.

Ändring av lösenord kan ske på två (tre) sätt.

1. Användaren går in i sina inställningar och skriver in sitt gamla lösenord och sitt nya och sparar, om det gamla stämmer överens med det i databasen ändras lösenordet till det nya.
2. Lösenordet återställs på två sätt (Admin kan reseta eller användaren kan utnyttja `forgotPassword()` funktionen). Då skickas en engångskod som gör det möjligt för användaren att skriva in ett nytt lösenord. Engångskoden gäller bara fram tills dess att det utnyttjats för att skriva in ett nytt lösenord eller det att användaren till kontot loggar in med sitt gamla lösenord. Detta ger säkerhet ifall användaren inte var den som begärde en återställning av lösenordet. Användaren får också en prompt i mailet att logga in om det var så att den en begärde återställningen.

2.1.2.4. Databasklass

Databashanteringen sköts av klassen Database som ärver mysqli. Alla funktioner är skriva på det objekt-orienterade sättet.

De funktioner som ändrar i ett element i databasen har även ett dynamiskt antal element, vilket ger större flexibilitet för hur man anropar klassen och det går att undvika att skriva flera olika funktioner för detta. Detta används till exempel när användare ska spara sin profil i `page-settings-user.php` där en variant av den dynamiska funktionen kallas när användaren inte vill byta lösenord och en annan kallas när användaren vill byta lösenord.

```
public function dbEditUser($SSNr) { // Attempts to edit a user and returns a boolean.
    // Function arguments are dynamic meaning
    // the first argument is the ID for User (SSNr).
    // The following arguments follow this pattern
    // (...,RowToChange,ValueToChangeTo...)
    // This works endlessly so as long as the
    // row to change is argument number X
    // where X%2=0 and value to change to
    // is argument number Y=X+1.

    $numargs=func_num_args();
    $arg_list=func_get_args();
    $param="";
    for($i=1;$i<$numargs;$i++){
        if($arg_list[$i]=="Password"){
            $j=$i+1;
            $result=$this->query("SELECT PwSalt FROM users WHERE SSNr='$SSNr'");
            $row=$result->fetch_assoc();
            $arg_list[$j]=$this->PasswordHash($row['PwSalt'],$arg_list[$j]);
        }
        if($i==$numargs-2){
            $param.=$arg_list[$i]."='";
            $i++;
            $param.=$arg_list[$i]."'";
        }else{
            $param.=$arg_list[$i]."='";
            $i++;
            $param.=$arg_list[$i]."',";
        }
    }
    if($this->query("UPDATE users SET $param WHERE SSNr='$SSNr'")==TRUE){
        return true;
    }else{
        return false;
    }
}
```

2.1.2.5. Valutor

Stöd för flera valutor finns för webbshoppen, dessa är även redigerbara via administrationsgränssnittet. Inställningarna ger även möjlighet att specificera hur den valda valutan ska visas (till exempel svenska kronor skrivs ut som ett suffix: 100 kr eller 100 SEK medans Euro skrivs ut som ett prefix: €10).

2.2 Individuella resultat

Gruppen har haft tydliga områden som var och en varit ansvarig för men har kodat lite på varandras under tiden, så det är inte 100% en person som gjort varje enskild modul.

2.2.1 Gustav

Gustav har till största delen skött planering och organisation av projektet. Och all dokumentation och skötseln av repositoryn på GitHub och utformning av workflows och liknande.

Eftersom Gustav gjort mycket av planering hade han bra koll på vad som behövde göras och skrev mycket pseudo-kod och skal för kommande funktioner (Output/Input för specifika funktioner)

Gustav har utvecklat följande:

- `.htaccess` med RegularExpressions och `router()` funktionaliteten.
- All validering med RegularExpressions (Använt HTML5 validering med JavaScript fallback).
- Innehållssidor (`page-*.php`) samt `header.php` och `footer.php`.
- Drygt hälften av `functions.php` *(Främst de funktioner som berör presentation som includeHeader() eller isAdmin()*.
- Bas-strukturen för User, Product och Order klasser.

2.2.2. Marcus

Marcus har varit den som programmerat och utformat grunden i hela applikationen. Ser man till rader kod *PHP* är det Marcus som programmerat mest.

Marcus har utvecklat följande:

- `config.php`, `db.php` samt skapandet av SQL-databasen utifrån den struktur som togs fram gemensamt.
- Drygt hälften av `functions.php` *(Alla de funktioner som skriver ut, samt flertalet Add/Delete/Edit-funktioner)*.
- Förändringar i de klasser Gustav skapat och klasser för de objekt som kom till senare i projektet.

2.2.3. David

David har tagit fram design och struktur för Front-End i HTML, CSS och JavaScript.

2.3. Extra (saker som ej stod med i kravspec)

2.3.1. Snygga URIs

Webbshoppen har snygga sökmotorsoptimerade URIs som är lätta att utläsa och som är beskrivande. Till exempel `/products/17-figure-skates/16-figure-skates-2000` för produkten *Figure Skates 2000* som är i kategorin *Figure Skates* eller `/settings/user/` för användarinställningarna. Snygga URIs är implementerat genom RegularExpressions i `.htaccess` kombinerat med funktionen `router()` i `init.php` som utifrån vilka *get* variabler som skickas med, laddar olika sidor.

```
RewriteRule ^admin/(shipping)/(\w+)*/*$          /index.php?admin=$1&package=$2 [QSA]
RewriteRule ^admin/(products)/(new|\d+)+/*$      /index.php?admin=$1&product=$2 [QSA]
RewriteRule ^admin/(categories)/(new|\d+)*/*$    /index.php?admin=$1&category=$2 [QSA]
RewriteRule ^admin/(orders)/(\d+)*/*$            /index.php?admin=$1&order=$2 [QSA]
RewriteRule ^admin/(users)/(new|\d+)*/*$         /index.php?admin=$1&user=$2 [QSA]
RewriteRule ^admin/(currencies)/(new|\d+)*/*$    /index.php?admin=$1&currency=$2 [QSA]
```

2.3.2. Validering

Validering utnyttjar HTML5 validering med JavaScript som fallback på klientsidan och Validering med PHP på serversidan. Valideringen utnyttjar RegularExpressions för att matcha input, anledningen till detta är att RegularExpressions är enkelt för att åstadkomma exakt samma

validering på klientsidan som på serversidan med hjälp av `pattern="^[A-ZÅÄÖa-zåäö]+$"` i *HTML* och `preg_match("$[A-ZÅÄÖa-zåäö]+$",string)` i PHP.

2.3.3. Omdirigeringar

Webbplatsen använder sig frekvent av omdirigeringar. Efter varje post sker en redirect mha redirect() funktionen vilket gör att användaren slipper se "Vill du återsända formulärdata", det ger även möjlighet (och som vi använder oss av) till kreative omdirigeringar. Till exempel om användaren har lagt till i kundkorgen men ej loggat in så skickas användaren till logga-in sidan och när användaren loggat in skickas hen till sidan som hen försökte besöka, i detta fallet kundkorgen.

2.4. Avgränsningar

Mycket funktionalitet som var påtänkt från början har lagts på hyllan för att göra det möjligt att få färdigt projektet i tid. Bland annat:

- Statistik i administrationsgränssnittet.
- Funktionalitet att dynamiskt ändra teman.
- Bygga presentationen av data på ett sådant sätt att man enkelt kan använda samma metod för att skriva ut flera olika typer av data. Som det är nu finns en funktion för varje specifik typ av data till exempel `listAdminProduct()`. En tanke var att göra liknande WordPress med en generell loop som skriver ut det som en viss sida är till för. Tanken var även att göra `router()` funktionen på samma sätt så att den utifrån en viss struktur som `page-*.php` sidorna var uppbyggda på, skicka användaren till rätt sida istället för den mer statiska variant som används nu.

3 Summering && Reflektion

Projektet i sin helhet har gått bra. Ribban sattes något högt initialt vilket gjorde att vi ej klarade första deadline. Samma ribba behölls dock och det blev färdigt med den funktionalitet som gruppen ville ha inbyggd till deadline för komplettering.
Till nästa gång ska det tänkas på att även om man har en hög ribba, se till att få färdigt den funktionalitet som krävs för lägsta betygen och sen arbeta sig uppåt, för att undvika stress över att inte bli färdiga.
Projektet började även i fel ändå, Front-End skulle gjorts i ett tidigt skede för att enkelt kunna se vad som behöver göras. När man börjar utveckla med databasen blir det ett väldigt abstrakt projekt vilket gör det svårare att få överblick.