UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Rok Avbar

Elementi implementacije spletne trgovine

DIPLOMSKO DELO VISOKOŠOLSKEGA STROKOVNEGA ŠTUDIJA

Mentor: doc. dr. Janez Demšar

LJUBLJANA, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00007/2010
Datum: 01.10.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ROK AVBAR**

Naslov: **ELEMENTI IMPLEMENTACIJE SPLETNE TRGOVINE**
ELEMENTS OF ONLINE SHOP IMPLEMENTATION

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Spletne trgovine so v zadnjih letih postale resna konkurenca klasičnim, zato predstavljajo precej pogosto spletno aplikacijo. S tehničnega vidika spletne trgovine postavljajo kup zanimivih praktičnih problemov, ki jih je potrebno pravilno rešiti za njihovo dobro delovanje.

V okviru diplomske naloge implementirajte prototip spletne trgovine in opišite različne tehnične detajle, na katere ste morali paziti.

Mentor:

doc. dr. Janez Demšar

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Zahvala

Zahvalil bi se svojemu mentorju, doc. dr. Janezu Demšarju, za njegovo prijazno pomoč in nasvete pri izdelavi te diplomske naloge. Velika zahvala gre moji Jasmini, ki me je ves čas spodbujala. Zahvala gre tudi moji družini, ki mi je v času pisanja diplomske naloge nudila potrebno moralno podporo.

Kazalo

POVZETEK

Spletne trgovine so v zadnjih letih postale resna konkurenca klasičnim trgovinam. Implementacija spletne trgovine zahteva reševanje kopice drobnih tehničnih problemov. Cilj te diplomske naloge je prikazati, za kakšne probleme gre in kako jih učinkovito rešiti. Med večje funkcionalnosti spada prijava preko tretjega ponudnika, kar izredno olajša registracijsko ter prijavno proceduro za uporabnike. Implementirali smo funkcionalnosti, kot so popusti, kuponi in plačevanje v različnih valutah. Poskrbeli smo za prepisovanje URLjev, da so le-ti lepši tako za uporabnike kot tudi za iskalnike (SEO). Omogočili smo plačilo s kreditnimi karticami. Poskrbeli smo tudi za učinkovito delo z zelo veliko podatkovno bazo (milijon zapisov) in pravilno implementacijo brskanja po straneh artiklov. Precej pozornosti smo posvetili tudi olajšanju administratorjevega dela in tehničnim podrobnostim, kot je izbira primernega mesta za uporabo AJAXa in interaktivnosti spletne strani na visokem nivoju z uporabo JavaScripta. Diplomsko delo ne kaže celotne postavitve spletne trgovine, temveč le tiste dele, ki so najzanimivejši z vidika implementacije.

ABSTRACT

Webshops have become real competitors to physical shops. Implementation of a webshop requires solving bunch of small technical problems. The main goal of this paper is to show what these problems are and how to efficiently solve them. One of them are login via external (3rd-party) provider, which drastically reduces the complexity of user registration and authentication (from the user's point of view). We have also implement functionalities like discounts, coupons and ability to pay in different currencies. We have taken care of rewriting the URLs, so they look nicer for the end user and webcrawlers (SEO). Payment can also be made with credit cards. We have efficiently dealt with database table with a lot of records (one million) and correctly implement page pagination. A lot of effort was invested into making an administrative side as simple as possible and using AJAX to make the page more interactive. The purpose of this paper is not to show how to implement a webpage from scratch, but to demonstrate the most challenging parts of its implementation.

1 UVOD

V uvodu bomo najprej predstavili tematiko same trgovine in opisali, kaj vse je potrebno implementirati.

Kot večina spletnih strani je tudi spletna trgovina razdeljena v dva dela. Prvi del je na voljo širši množici in to je uporabniški del. Drugi del pa je administracijski del in je namenjen zaposlenim oziroma „lastnikom“ spletne trgovine.

Na grobo imajo uporabniki možnost pregledovanja artiklov po kategorijah, možno je tudi iskanje določenih artiklov glede na njihov naziv. V kategorijah je možno tudi artikle filtrirati po različnih lastnostih. S klikom na določen artikel nas pripelje do podrobnejšega pregleda dotičnega artikla. Kupec lahko izbrani artikel doda v košarico, ko konča z nakupovanjem, pa odda naročilo.

Uporabniki spletne trgovine lahko upravljajo s svojim profilom, v katerem nastavijo osnovne osebne podatke, kot so ime in priimek, naslov in podobno.

Zaposlenim je omogočen dostop v administracijski del, v katerem lahko dodajajo in brišejo ponujene artikle ter spreminjajo njihove lastnosti, določajo kategorije izdelkov in podobno. Poleg tega potrebujejo seveda tudi vpogled v naročila in imajo možnost spreminjanja statusa.

Administrator spletne strani pa je oseba, ki skrbi za pravilno delovanje spletnega mesta s tehničnega vidika in ni nujno eden od oseb, ki vzdržujejo tudi sezname artiklov.

V okviru diplomske naloge smo sistematično implementirali vse potrebne funkcionalnosti spletne trgovine.

Spletno stran smo sprogramirali v jeziku PHP. Ta po mnenju mnogih programerjev velja za »grd« programski jezik. Dejstvo pa je, da je jezik uporaben, zato gre za enega najbolj uporabljanih jezikov, ki ga uporabljajo tudi najbolj popularna spletna mesta, kot recimo Facebook, Digg in WordPress.

Del kritike jezika lahko verjetno pripišemo tudi dejstvu, da večina »priučenih programerjev« začne programirati v PHPju. Večina začetnikov programira grdo in se šele z leti nauči lepega programiranja; če po nekaj letih zamenjajo svoj primaren programskih jezik,pogosto kritizirajo svojo staro kodo in njeno nepreglednost pripišejo jeziku.

Res pa je, da je PHP dokaj nekonsistenten, kar je najbolj očitno iz poimenovanja funkcij. Funkcija, ki vse črke v besedah spremeni v veliko črke, se imenuje strtoupper(), če pa želimo zamenjati črko z drugo črko, uporabimo str_replace(). Drug primer je funkcija htmlentites(), katere reverzna funkcija je html_entity_decode(). Medtem ko ima, recimo, Python le dve funkciji za urejanja (list.sort() in sorted()), katerih nastavitve določimo z argumenti, ima PHP sort(), arsort(), asort(), ksort(), natsort(), natcasesort(), rsort(), usort(), array_multisort() in uksort(). Dodatna pomanjkljivost jezika je, da ne podpira imenskih prostorov, zato so nekateri imena funkcij zelo dolga, kot recimo xsl_xsltprocessor_transform_to_xml.

Osnovna odlika PHPja je, da omogoča kombinacijo prikazovalne logike z izvajalno (angl. Business logic). To je seveda odlično, če gre za majhen projekt. Takoj, ko začne projekt ter koda rasti, ta lastnost povzroča zmešnjavo. Prikazovalno logiko je strogo potrebno ločiti od izvajalne. Torej PHP kodo je potrebno ločiti od HTML kode. Tako se preglednost kode drastično izboljša. Najbolj uporabljen sistem za vodenje predlog (Template engine) je Smarty. Smarty ima svojo sintakso za implementiranje spremenljivk v HTML, podpira tudi for zanke ter pogojni stavek if.

V diplomskem delu smo se izognili opisovanju implementacije celotne funkcionalnosti spletne trgovine, saj je znaten del le-te tehnično trivialen. Namesto tega smo opisali le najzanimivejše postopke in težave, na katere naletimo pri postavljanju tovrstne aplikacije.

2 PRIJAVA, REGISTRACIJA IN SEJA

Spletna trgovina ne uporablja uporabniških imen (username), ampak namesto tega uporablja spletni naslov uporabnika. Zdelo se nam je bolj smiselno, da se uporabnik prijavi v sistem s svojim emailom ter poljubno izbranim geslom, kot pa da za prijavo uporabi uporabniško ime. Spletni naslov pa mora vpisati vseeno.

Edina slabost tega je, da imamo potem problem pri prikazovanju komentarjev oziroma mnenj o izdelku. Problem pa je v tem, da po navadi izpišemo datum komentarja, vsebino ter uporabniško ime. Verjetno večini uporabnikom ne bi bilo všeč, da izpisujemo njegov ime in priimek pri komentarjih.

V trgovino se je mogoče prijaviti na dva različna načina.

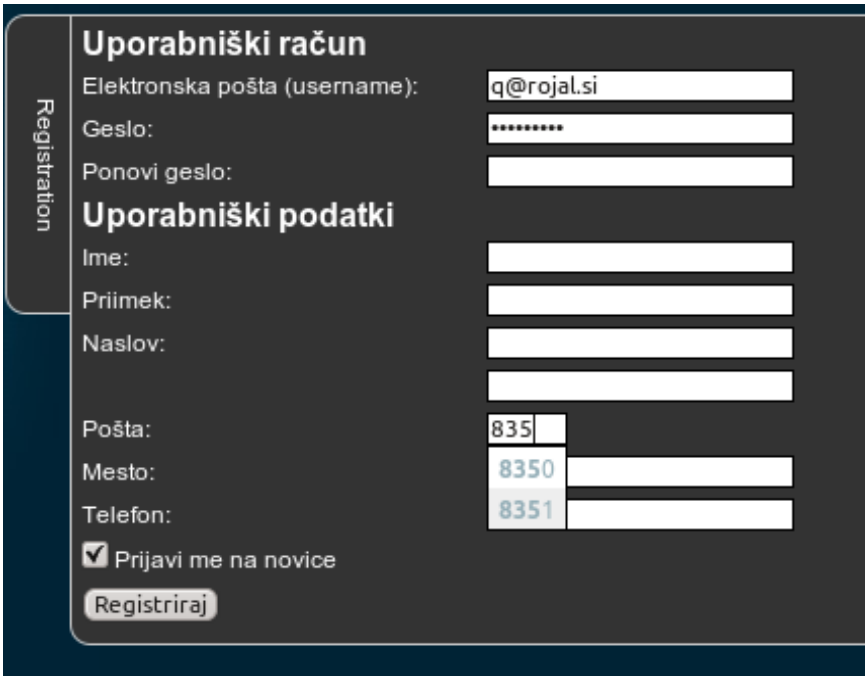
2.1 Prijava z registracijskim obrazcem

Če obiskovalec še ni uporabnik, se lahko registrira enostavno, tako, da izpolni registracijski obrazec. Izpolniti je potrebno:

- elektronski naslov
- geslo
- ime, priimek
- naslov
- poštna številka
- mesto ter
- telefonska številka

Vse poštne številke ter mesta so shranjena v bazi. Seznam le-teh smo dobili iz spletne strani pošte Slovenije. To nam je omogočilo, da med tem, ko uporabnik vpisuje poštno številko ali mesto, ponudimo možne poštne številke ali mesta, na podlagi vpisanega. Ko uporabnik izbere poštno

številko ali mesto iz ponujenega seznama, se mu mesto ali poštna številka izpolni avtomatsko (Slika 1).



Slika 1. Registracijski obrazec.

Slika 2. Primer, ko se gesla v registracijskem obrazcu ne ujemata.

Med pisanjem gesla se tudi to primerja, ali je enako (Slika 2). Dokler geslo ni enako, se rob polja obarva rdeče. To preverjanje seveda ni dovolj. Geslo je potrebno tudi preverjati na strežniški strani (PHP), ker se JavaScript izvaja samo na uporabnikovi strani v brskalniku in lahko pride do zlorab.

Ko uporabnik zaključi z izpolnjevanjem registracijskega obrazca, pritisne gumb „Registriraj“, na dnu obrazca. Ob pritisku na gumb, se najprej požene koda v JavaScriptu, ki preveri, ali so vsi podatki izpolnjeni pravilno. Ta obrazec je potem poslan skripti, ki ponovno preveri poslane podatke in če so vsi podatki v redu jih zapiše v bazo nato pošlje uporabniku spletno pošto z aktivacijsko povezavo, katero mora uporabnik klikniti v roku, ki ga je določil administrator. Če povezavo klikne prepozno, sistem izbriše uporabnika ter ga o tem obvesti.

Primer povezave:
shop.php?group=register&email=rok@rojal.si&hash=99d1507b3857590ce2b5e2adef02c5ed596fcdd1ba7d1cbc12559516a581b407

Hash se izračuna iz elektronskega naslova in soli po algoritmu SHA256. Hash se ne shrani v bazo, temveč se pošlje skupaj z emailov uporabniku. V bazo se shrani samo čas registracije.
Ko uporabnik obišče povezavo, se pokliče metoda, ki sprejme posredovan elektronski naslov ter hash. Nato se izračuna hash na isti način kot ob registraciji in se preveri ali se ujema s posredovanim hashom. Če je prišlo do spremembe, pomeni, da je uporabnik spremenil email ali hash, zato končamo z izvajanjem metode.
Nato preverimo, če je uporabnik obiskal povezavo v določenem roku; če ni, uporabnika izbrišemo, drugače pa aktiviramo račun.

```
public function userActivate($mail,$hash) {
	global $db;

	$emailhash=Stuff::hashit($mail);
	if($emailhash!=$hash) return false;
	$mtime=$db->selectField("users",array("email"=>$mail),"registered");

	if(!$mtime) return false;
	if(time()-$mtime>$GLOBALS['registrationConfirmTime']*60){
		global $user;
		$userdata=$user->getUserDataFromEmail($mail);
		$db->delete("addresses",array("user_id"=>$userdata['id']));
		$db->delete("users",array("id"=>$userdata['id']));
		return false;
	}
	return $db->update("users",array("isactive"=>1),array("email"=>$mail));
}
```

Če je metoda vrnila „false“, potem obvestimo uporabnika o problemu, drugače pa ga avtomatsko prijavimo.

2.2 Prijava prek tretjega ponudnika

Drugi način pa je prijava oziroma registracija, ki poteka preko tretjega ponudnika. To smo dosegli z implementacijo OpenIDja *[2*] ter OAuthja *[10*].

2.2.1 OpenID

OpenID omogoča, da obstoječi uporabniški račun uporabijo na različnih spletnih straneh, brez ustvarjanja novega gesla.

Veliko uporabnikov spleta že ima OpenID, ne da bi se tega zavedali. OpenID ponudnikov je ogromno. Eden iz verjetno največjih je Google, a seznam ni kratek: Yahoo, LiveJournal, MySpace, WordPress, Blogger, Verisign, Typepad, MyOpenID, Steam, Orange, TonidoOpenID, Launchpad...

Z uporabo OpenIDja geslo dobi le ponudnik OpenID (recimo Google) in ta potrdi identiteto uporabnika spletni strani, ki jo obiskuje (recimo našo spletno trgovino). Ker spletna stran ne vidi uporabnikovega gesla in ne more priti do kraje ali celo do zlorabe vašega računa, razen nekaterih primerov, ki jih bomo opisali kasneje.

OpenID se hitro širi, ima že več kot 1 bilijon OpenID podprtih računov ter več kot 50.000 spletnih strani, ki sprejemajo OpenID za prijavo.

Vsak uporabnik ima unikatni OpenID URL, ki se imenuje *identifier*. Primer Googlovega identifierja je https://www.google.com/accounts/o8/id?id=AItOawkOms9n3DWY75th9euDOpb91cXFFqbuGs4 in primer Steam identifierja http://steamcommunity.com/openid/id/76561198002361060

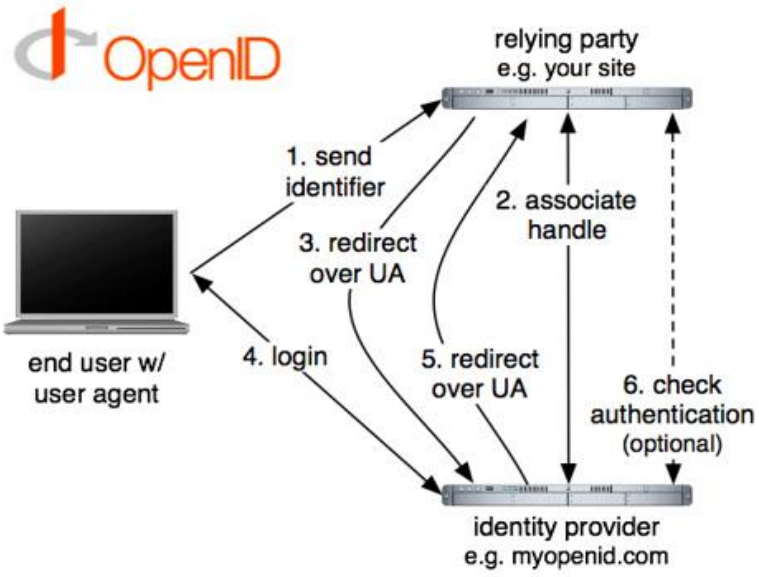
Spletna stran lahko od uporabnika oziroma od njegovega OpenID ponudnika zahteva različne atribute. Naša spletna trgovina zahteva ime, priimek ter email. OpenID ponudnik uporabnika obvesti, če spletna stran zahteva dodatne informacije o njegovem računu, in ponudi uporabniku možnost potrditve teh zahtev.

Ponudniki OpenID se razlikujejo v malenkostih, večji del pa je enak pri vseh. MySpace zahteva, da v URL podamo uporabniško ime, Steam pa ne omogoča zahteve po dodatnih atributih in vrne samo uporabniško ime ter identifier ter vedno zahteva prijavo, tudi, če je uporabnik že prijavljen..

2.2.1.1 Implementacija OpenIDja v spletno trgovino

Knjižnica za delo z OpenIDjem je na voljo na domači strani ponudnika in je implementirana v različnih programskih jezikih, PHP, Java, C#, Haskell, Perl, Python, Ruby, Smalltalk...

Naša implementacija OpenIDja je za uporabnika izredno praktična. Uporabniku na spletni strani ni potrebno izpolnjevati registracijskega obrazca, zato mu tudi ni potrebno potrjevati registracije.. Vse, kar mora narediti, je izbrati prijavo preko Googla, nato ga spletna trgovina preusmeri na Google, kjer se prijavi, nato pa ga Google preusmeri nazaj na spletno trgovino in uporabnik je prijavljen v sistem. Če pa je uporabnik v Google že prijavljen, potem je pa prijava v spletno trgovino še krajša. Ko uporabnik klikne na prijavo preko Googla, se mu spletna stran samo osveži in že je prijavljen. Seveda se v ozadju zgodi več korakov (Slika 3), ki pa jih uporabnik ne vidi.



Slika 3. Potek tokov OpenIDja

Prijava je razdeljena v dva dela. Prvi del je priprava zahteve ter preusmeritev uporabnika na OpenID ponudnika, v tem primeru gre za Google.

```
$identifier="https://www.google.com/accounts/o8/id";
$openid = new LightOpenID;
$openid->identity = $identifier;
$openid->required=array("contact/email","namePerson/first","namePerson/last");
header('Location: ' . $openid->authUrl());
```

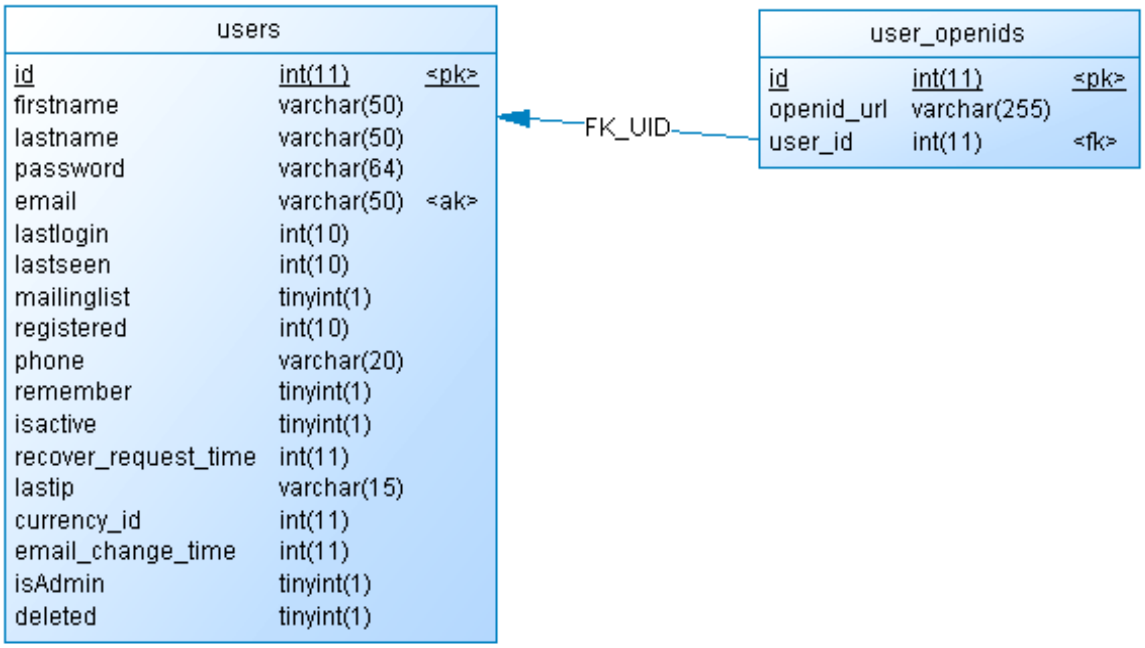
Drugi del pa poskrbi za vpis uporabnika v bazo. Najprej preverimo, ali so vsi potrebni atributi vrnjeni od OpenID ponudnika pravilni; če niso, uporabnika preusmerimo na domačo stran trgovine.

Nato preverimo, ali se je uporabnik v preteklosti že prijavil v spletno trgovino preko tega OpenID ponudnika. To storimo tako, da pogledamo, če njegov identifier že obstaja v tabeli „user_openids“. Če ga najdemo, ga enostavno prijavimo, drugače pa preverimo, ali je uporabnik že registriran, kar naredimo tako, da sestavimo poizvedbo za bazo in sicer iščemo uporabnika s takšnim elektronskim naslovom, kot ga ima trenutni uporabnik. Če ga najdemo, vpišemo uporabnikov identifier v tabelo ter uporabnika prijavimo. Če pa uporabnik še ne obstaja, ga ustvarimo, vpišemo njegov identifier v tabelo „user_openids“ ter ga prijavimo v sistem (Slika 4).

```
$openid = new LightOpenID;
if(!$openid->validate()){
	$login->logout();
	header("Location: ".$GLOBALS['url']);
	exit;
}
$me=$openid->getAttributes();

if(!$userid=$login->checkOpenId($openid->identity)){
	//pogledamo, ce je user lokalno registriran
	if($userdata=$user->getUserDataFromEmail($me['contact/email'])){
		$userid=$userdata['id'];
		$login->insertOpenId($userid,$openid->identity);
		$login->loginUser($userid,"openid");
	}
	//user se ne obstaja, naredimo novega
	else{
		$data=array("firstname"=>$me['namePerson/first'],
		   "lastname"=>$me['namePerson/last'],
```

```
                    "email"=>$me['contact/email'],
                    "mailinglist"=>"1",
                    "isactive"=>1);
                $register=new Register();
                if($newuserid=$register->addUser($data,false)){
                        $login->insertOpenId($newuserid,$openid->identity);
                        $login->loginUser($newuserid,"openid");
                }
        }
}
else{
        $login->loginUser($userid,"openid");
}
$referer=$_SERVER['HTTP_REFERER'] ? $_SERVER['HTTP_REFERER'] : $GLOBALS['url'];
header("Location: ".$referer);
exit;
```



Slika 4. Pogled strukture baze za OpenID implementacijo. V tabelo „user_openids“ zapišemo uporabnikov identifier ter ID uporabnika.

2.2.1.2 Zlorabe OpenID

Z uveljavitvijo nove tehnologije je seveda možnost po novih problemih in zlorab. Ena izmed pogostih problemov OpenIDja je izredno znani Phishing. Phishing na kratko pomeni, da napadalec poskuša pretentati napadenega z imitacijo prave spletne strani in mu na tak način

ukrade podatke.

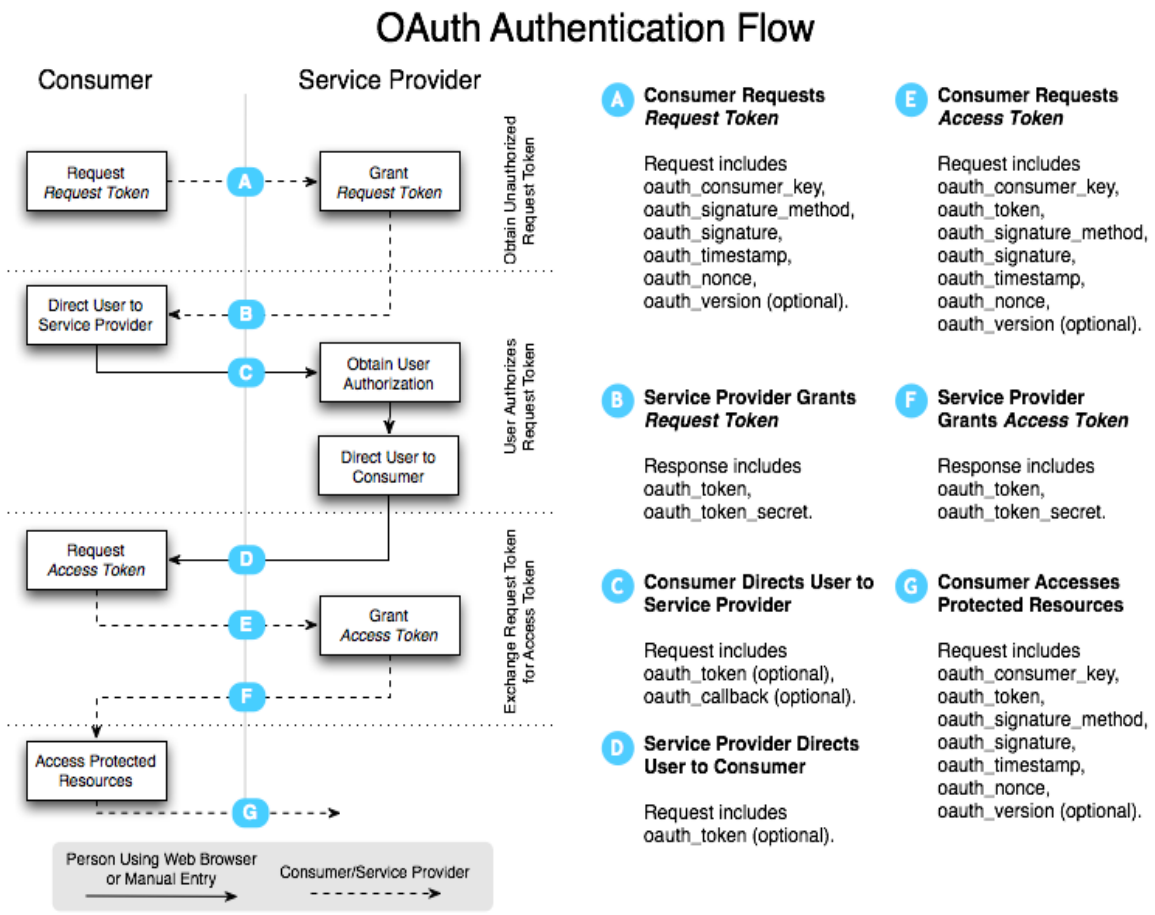
Torej napadalec lahko naredi spletno trgovino, kot je naša, nato klonira Googlovo prijavno stran. Ob kliku na prijavo preko Googla, uporabnika preusmeri na to klonirano Googlovo stran, v katero uporabnik vpiše svoje podatke in tako je napadalec dobil vse potrebne podatke.

Rešitev je dokaj enostavna. Uporabnik se pred prijavo na spletno trgovino preko Googla, prijavi v Google, recimo na Gmail. V tem primeru spletna trgovina od vas ne bo zahtevala nobenih podatkov, oziroma če bi jih, potem je možno, da gre za zlorabo.

Resna rešitev problema pa bi bila striktna uporaba protokola https in certifikatov, certificiranih s strani agencij, ki jim brskalniki zaupajo.

2.2.2 OAuth

OAuth, krajše za Open Authorization je odprt standard za avtorizacijo. Uporabniku omogoča deljenje privatnih virov (slike, filmi, kontakti) shranjenih na eni strani z drugo spletno stranjo, ne da bi moral ponovno pošiljati prijavne podatke (ponavadi uporabniško ime ter geslo). Oauth omogoča uporabniku, da „izda“ žeton (*token*) namesto uporabniškega imena ter gesla. Vsak žeton omogoči dostop do določene strani in to za določen čas (recimo eno uro) (Slika 5).



Slika 5. Potek tokov pri OAuth.

2.2.2.1 Prijava preko Facebooka z uporabo OAauth

Facebook je maja 2009 začel sprejemati račune, podprte z OpenID [11]. To pomeni, da se sedaj lahko prijavite na Facebook z Googlovim računom. Žal pa Facebook ni ponudnik OpenID, kar pomeni, da se na druge spletne strani ne morete prijaviti s Facebookovim računom prek OpenID.

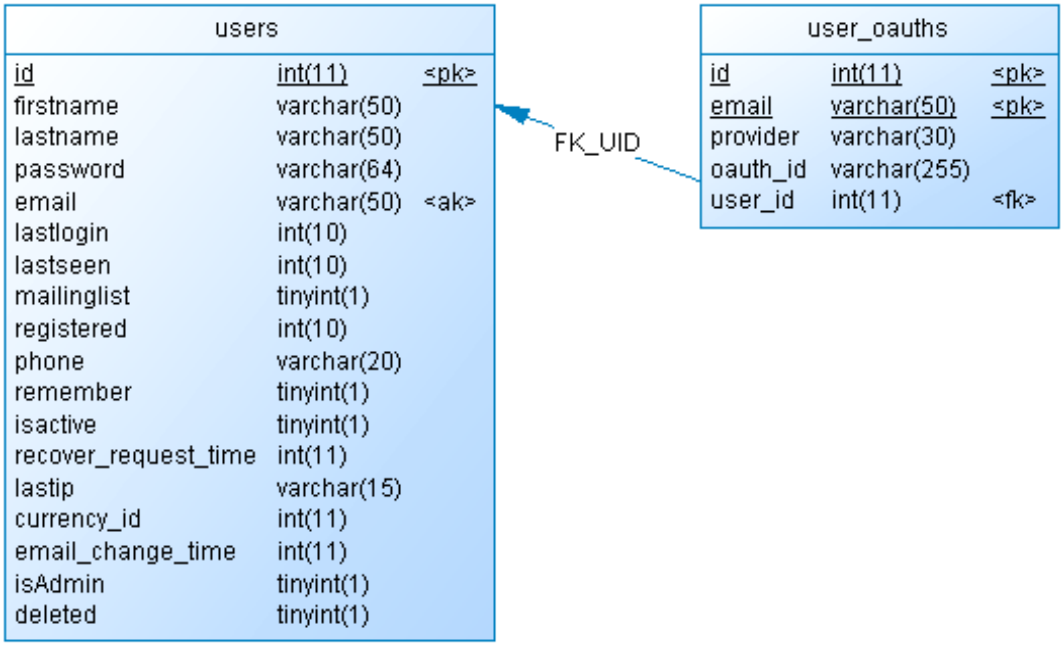
Facebook tudi ponuja svoj sistem za prijavo, ki se imenuje „Login Button“ in spada v sistem „Connect“, ki so ga razvili sami.

Facebookov Oauth ponuja veliko stvari, predpogoj za to pa je, da spletna stran preveri, ali je uporabnik prijavljen v Facebook ali ne. Šele potem lahko dostopa do njegovih podatkov.

2.2.2.2 Implementacija OAuthja v spletno trgovino

Knjižnica za delo z Oauthjem je na voljo na domači strani projekta v mnogih programskih jezikih, PHP, DotNET, C#, ColdFusion, Java, JavaScript, Objectiv-C, Ocalm, Perl, Python, Ruby...
Naša implementacija OAuthja je za uporabnika podobna implementaciji OpenIDja. Ko uporabnik klikne, da se želi prijaviti preko Facebooka, se zgodi podobno kot pri prijavi preko Googla: če še ni prijavljen, ga spletna stran ga preusmeri na Facebookovo vstopno stran, kjer se mora vpisati. Tu se mu tudi izpiše, kakšne podatke spletna trgovina zahteva od uporabnika. V primeru, da gre samo za osnovne podatke, se mu ne izpiše nič.

Sestava tabele za shranjevanje Oauth je podobna kot pri OpenIDju (Slika 6). Razlika je le v tem, da v tem primeru ne moremo shraniti identifierja, ker ga ni, shranimo pa lahko uporabnikov ID na Facebooku. Ta pa je lahko isti kot ID drugih ponudnikov, zato, je potrebno ob vsakem zapisu v bazo, zapisati tudi OAuth ponudnika, v tem primeru Facebook. Privzeli pa smo, da je Facebookov ID vedno naravno število (Unsigned Integer). Če bi se kasneje odločili za podporo še kakšnega drugega OAuth ponudnika, bi bilo mogoče potrebno to spremeniti v varchar (niz znakov).



Slika 6. Sestava tabele za implementacijo OAuth ter povezava z uporabniki.

Klic APIjev je precej enostaven. Ko uporabnika, na spletni trgovini klikne prijavo preko Facebooka, moramo seveda prvo ustvariti nov objekt, nato pa uporabnika preusmerimo na vpisno stran Facebooka.

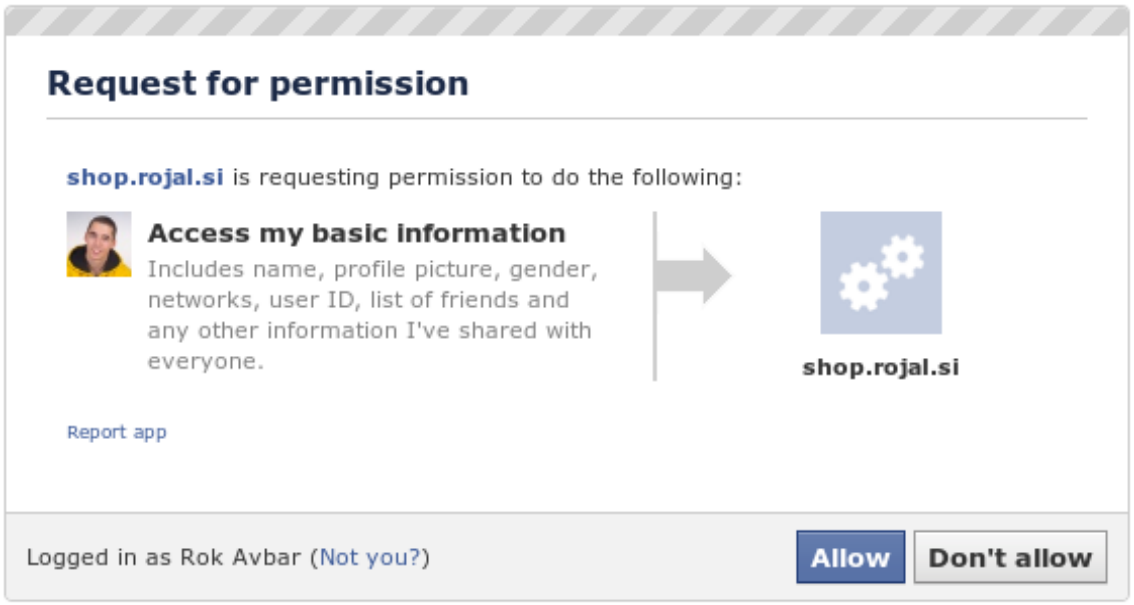
AppID je ID aplikacije, *secret* pa je skriti ključ, ki ga zgenerira Facebook.

```
$facebook = new Facebook(array(
  'appId'  => '543234564534234',
  'secret' => 'klajsdfk734kljhaslkjas',
  'cookie' => false,
));
```

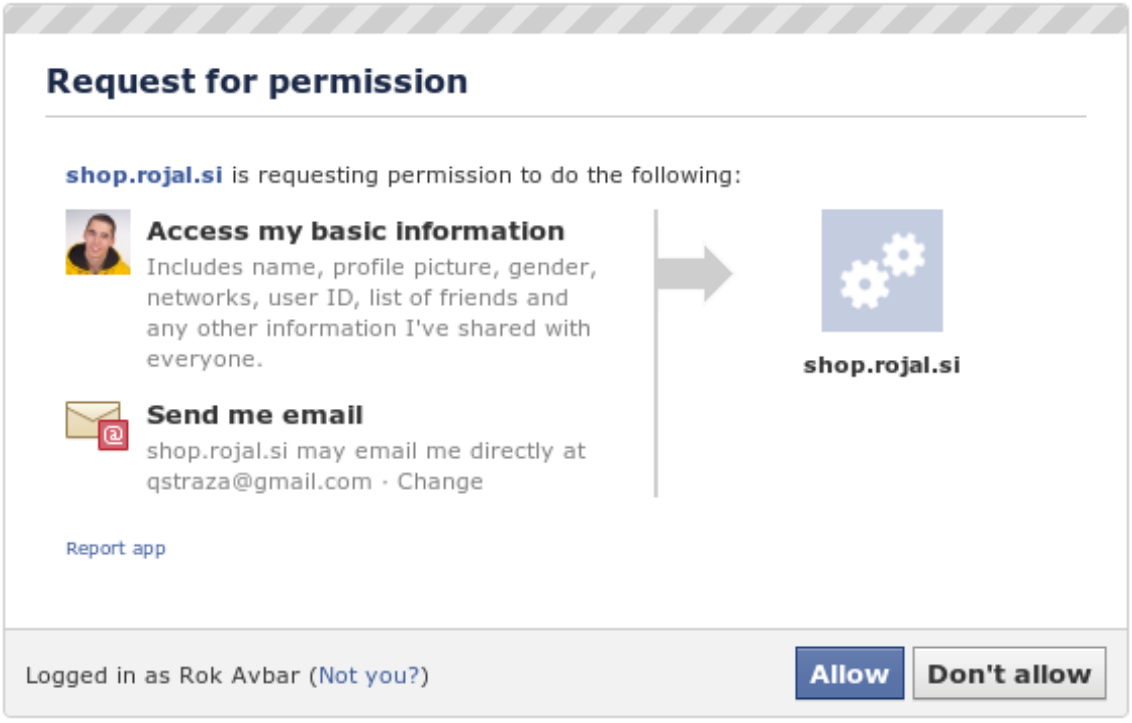
Nato preusmerimo uporabnika ter zaključimo izvajanje.

```
header("Location: ".$facebook->getLoginUrl(array('req_perms' => 'email')));
exit;
```

Metoda »getLoginUrl« sprejme več argumentov, noben pa ni obvezen. Tabela z indeksom »req_perms« sprejme polja, ki jih spletna trgovina zahteva od Facebooka *[6]*. V našem primeru spletna trgovina zahteva, da Facebook, poleg prevzetih podatkov, vrne še uporabniški email. O tem je seveda uporabnik tudi obveščen(Slika 7, Slika 8).



Slika 7. Primer, ko spletna trgovina zahteva samo osnovne podatke.



Slika 8. Primer, ko spletna trgovina zahteva še uporabnikov email.

Če spletna stran ne zahteva nobenih podatkov od Facebooka, potem dobi sledeče prevzete podatke.

Klic brez argumentov:

```
header("Location: ".$facebook->getLoginUrl());
```

In vrnjeni podatki:

```
Array
(
    [id] => 1049823886
    [name] => Rok Avbar
    [first_name] => Rok
    [last_name] => Avbar
    [link] => http://www.facebook.com/profile.php?id=1049823886
    [hometown] => Array
        (
            [id] => 115057721840110
            [name] => Straza, Slovenia
        )
    [location] => Array
        (
            [id] => 115057721840110
            [name] => Straza, Slovenia
        )
    [work] => Array
        (
            [0] => Array
                (
                    [employer] => Array
                        (
                            [id] => 110252765664505
                            [name] => Fakulteta za računalništvo in informatiko
                        )
                    [start_date] => 0000-00
                    [end_date] => 0000-00
                )
        )
    [education] => Array
        (
            [0] => Array
                (
                    [school] => Array
                        (
                            [id] => 110252765664505
                            [name] => Fakulteta za računalništvo in informatiko
                        )
                    [year] => Array
                        (
                            [id] => 116137395080252
                            [name] => 2006
                        )
                    [type] => College
                )
            [1] => Array
                (
                    [school] => Array
                        (
                            [id] => 102130683162388
                            [name] => Ekonomska Šola Novo mesto
                        )
                    [year] => Array
                        (
                            [id] => 139397649432038
                            [name] => 2002
                        )
                    [type] => High School
                )
        )
```

```
    [gender] => male
    [timezone] => 1
    [locale] => en_GB
    [verified] => 1
    [updated_time] => 2010-11-06T14:30:14+0000
)
```

Če pa bi poklicali metodo getLoginUrl z dodatnim argumentom in sicer tako:

```
header("Location: ".$facebook->getLoginUrl(array('req_perms' => 'email')));
```

pa bi poleg prevzetih polj dobili še polje email.

Ko se uporabnik prijavi v Facebook, pa ga le-ta preusmeri nazaj na spletno trgovino, kjer je potrebno preveriti, če se je uporabnik uspešno prijavil.

Najprej ponovno ustvarimo objekt.

```
$facebook = new Facebook(array(
  'appId'  => '146829528671307',
  'secret' => '33a54a25b93428ed8bcb266a3dcbe579',
  'cookie' => false,
));
```

Pokličemo metodo getSession. S klicem te metode smo preverili, če je seja pravilno podpisana s „secret“. Ne vemo pa, če je seja še veljavna.

```
$session = $facebook->getSession();
```

Sedaj pa pokličemo metodo „api“ z argumentom „/me“, ki bo vrgla izjemo, če seja ne bo veljavna.

```
$me = null;
if ($session) {
	try {
		$me = $facebook->api('/me');

	} catch (FacebookApiException $e) {
		error_log($e);
	}
}
```

Sedaj pa pride na vrsto del, ki poskrbi za uporabnikovo prijavo oziroma registracijo. Ta del kode pa je podoben kot pri OpenIDju.

Preverimo, če se uporabnik že nahaja v tabeli „user_oauth“, če se, uporabnika samo prijavimo, drugače pa preverimo, če se je uporabnik že registriral, če se je, vpišemo v tabelo „user_oauth“ Facebookov ID ter njegov „user_id“ in ga prijavimo. V nasprotnem primeru pa moramo uporabnika registrirati ter prijaviti.

```
if($me){
        //preveris ce je ze zapis v bazi
        if(!$userid=$login->checkOAuth($session['uid'],"facebook")){
        //pogledamo, ce je user lokalno registriran
        if($userdata=$user->getUserDataFromEmail($me['email'])){
              $userid=$userdata['id'];
              $login->insertOAuth($userid,"facebook",$me['id']);
              $login->loginUser($userid,"oauth");
            }
        //user se ne obstaja, naredimo novega
        else{
               $data=array("firstname"=>$me['first_name'],
                      "lastname"=>$me['last_name'],
                      "email"=>$me['email'],
                      "mailinglist"=>"1",
                      "isactive"=>1);
               $register=new Register();
               if($newuserid=$register->addUser($data,false)){
                      $login->insertOAuth($newuserid,"facebook",$me['id']);
                      $login->loginUser($newuserid,"oauth");
               }
        }

        }
        else{
               $login->loginUser($userid,"oauth");
        }
        $referer=$_SERVER['HTTP_REFERER'] ? $_SERVER['HTTP_REFERER'] : $GLOBALS['url'];
        header("Location: ".$referer);
        exit;
}
```

Facebook omogoča veliko stvari, ki bi jih lahko uporabil v marketinške namene. Primer takšne uporabe je na primer pisanje na uporabniški zid *[8]*.

Enostavno pokličemo metodo „api“, ki je izredno bogata metoda, saj omogoča tako-rekoč vse.

```
$facebook->api('/me/feed', 'post', array('message'=> "Pravkar sem se prijavil na spletno trgovino..."));
```

Od uporabnika lahko tudi zahtevamo permenantni žeton in s tem lahko potem dostopamo do uporabniškega profila vedno. To pomeni, da bi lahko, denimo, kadarkoli pisali po uporabniškem zidu.

Če bi mogoče želeli izpisati zadnji status od vsakega prijatelja, ki jih ima naš prijavljen uporabnik, bi naredili tako. Najprej moramo zahtevati dodatne parametre in sicer „read_friendlists“, ki nam omogoča, da dobimo vse prijatelje prijavljenega uporabnika in „friends_status“, ki omogoča, da preberemo njihov status. Dobimo lahko vse statuse, ki so bili objavljeni. Facebook ne vrne vsega v enem zahtevku, ampak jih vrne samo zadnji 25, če želimo več, pa nastavimo še argument „until“, ki je formata „date“.

```
header("Location: ".$facebook->getLoginUrl(array('req_perms' => 'read_friendlists,friends_status')));

friends=$facebook->api("/me/friends");
foreach($friends['data'] as $f){
	$statuses=$facebook->api($f['id']."/statuses&limit=1");
	foreach($statuses['data'] as $s)
		echo $s['message']."¥n";
}
```

2.3 Seje

Ker protokol http nima stanj, se pojavi problem, kako ohranjati sejo. Pri vsaki poslani zahtevi strežniku, le-ta ne ve nič o prejšnji zahtevi in obravnava vsako zahtevo kot novo. Potrebno je torej zagotoviti uporabniku, da ostane prijavljen v spletno stran med tem, ko klika po njej.

Poznamo dva mehanizma za ohranjanje sej. Prvi je vodenje seje v URLju (URL Rewriting), drugi pa z uporabo piškotkov. Bolj uveljavljena je metoda s piškotki.

Običajni princip vodenja seje je sledeč. Uporabnik obišče spletno stran. Spletna stran prebere id seje (*sessionID*) preko URLja ali piškotka. Nato se preveri, če je seja veljavna in kateremu uporabniku seja pripada. Če id-ja še ni, se le-ta zgenerira in pošlje po piškotku ali prek URLja uporabniku.

Primer URL Rewritinga je
shop.php?session=dj23hd98234hkh234j

Največja slabost tega je, če uporabnik pošlje URL drugi osebi, bo ta oseba prijavljena kot prvi uporabnik.

Pri postavljanju spletne trgovine smo posvetili veliko pozornosti sami prijavi ter strukturi piškotka. Piškotki so zelo izpostavljeni, saj jih je možno manipulirati in s tem prevzeti tujo identiteto.

V piškotku se skriva več vrednosti:

- niz naključnih znakov z visoko entropijo *[4]*,*[5]*,
- čas v sekundah, ki je minil od 1. januarja 1970 (Unix time),
- ID uporabnika

Poleg zgornjih vrednosti se v piškotku skriva še izvleček zgornjih vrednosti z dodano soljo ter IP uporabnika po algoritmu SHA256. S tem smo si zagotovili integriteto piškotka.

Na koncu pa vse naštete vrednosti z izvlečkom vred zakriptiramo in pošljemo kot piškotek uporabniku.

Preverjanje piškotka pa poteka v obratni smeri. Najprej ga je potrebno dekriptirati; če se kje zatakne, piškotek zavržemo. Sedaj imamo ponovno tri vrednosti ter izvleček le teh. Najprej preverimo, ali je razlika trenutnega časa ter časa, ki se nahaja v piškotku, še vedno v določenem intervalu (določi ga administrator), če ni, ga zavržemo. Sledi preverjanje integritete; iz vseh treh vrednostih ponovno izračunamo izvleček po istem algoritmu ter isto soljo in z dodanim IPjem trenutnega uporabnika. Če se izračunani izvleček ne ujema z izvlečkom, ki se nahaja v piškotku, je prišlo do kakšne napake pri prenašanju paketkov (kar je malo verjetno, saj je dekripcija stekla tekoče) ali pa je uporabnik nekako dekriptiral piškotek ter spremenil vrednost le-tega. Če se izvlečka ne ujemata, preverjanje seveda zavržemo. Zadnje preverjanje je primerjanje uporabniškega IDja, ki se nahaja v bazi z IDjem, ki se nahaja v bazi in sicer tabeli uporabnikov. Če uporabnik z dotičnim IDjem ne obstaja, preverjanje zavržemo.

Niz naključnih znakov nam omogoča, da ima kriptiran piškotek vedno drugačno vrednost. Naključne znake pridobivamo iz vira, ki ima visoko entropijo in sicer iz /dev/urandom.

2.4 Kdo je „ONLINE“

Prikazovanje koliko uporabnikov je trenutno na spletni strani je pomembno. Informacija o tem je za druge uporabnika zanimiva, za administratorje pa pomembna.

Ker je HTTP protokol brez stanj, ne moremo vedeti, koliko je trenutnih aktivnih uporabnikov. Torej če napišemo skripto, ki izpiše trenutne povezane uporabnike, bo rezultat skoraj vedno le ena in sicer bo to tisti uporabnik, ki bo trenutno obiskal oziroma pognal skripto. Namesto tega navadno kažemo uporabnike, ki so bili aktivni recimo v zadnjih desetih minutah.

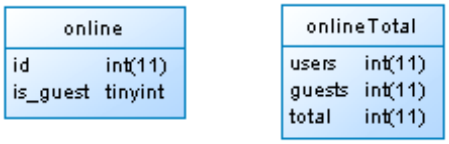
Vsak uporabnik, ki obišče spletno stran, dobi piškotek, vrednost piškotka je zapisana tudi v bazi in sicer v tabeli gostov. V zapisu se poleg vrednosti, ki je enaka kot v piškotku, nahaja čas, kdaj je bil uporabnik oziroma gost nazadnje viden. Če se uporabnik prijavi, se gosta izbriše in se zadnji

viden čas uporabnika v tabeli uporabnikov posodobi. Ob vsaki novi zahtevi, oziroma ob vsaki obiskani strani se uporabniku ali gostu posodobi zadnji viden čas.

Z enostavno poizvedbo izpišemo vse uporabnike ter goste, ki imajo zadnji viden čas mlajšega od 10 minut.

WHERE lastseen>UNIX_TIMESTAMP()-600;

Velika pomanjkljivost zgornje metode je hitrost. Problem je namreč v tem, da sedaj ob vsakem obisku, ob vsakem kliku pogledamo v bazo, koliko je trenutnih uporabnikov. Veliko bolj smiselno je , da naši bazi dodamo dve novi tabeli. V eno tabelo imamo vse trenutne uporabnike, v drugi pa vsoto (Slika 9).



Slika 9. Tabeli uporabljeni za štetje uporabnikov.

Sedaj pa še potrebujemo skripto, ki bo polnila zgornji dve tabeli v določenem intervalu.

Skripta je enostavna.

- Najprej izbriše vse vnose v tabeli „online“.
- Nato mora napolniti prvo tabelo z prijavljenimi uporabniki ter gosti, to stori tako:

```
insert into online select
    id, 0
    from users
    where lastseen>(TRENUTNI ČAS-600)
```

Ter še za goste:

```
insert into online select
    id, 1
```

```
from guests
where lastseen>(TRENUTNI ČAS-600)
```

- Zadnji korak pa je napolnitev vsoto vseh. Enostavno prešteje goste ter prijavljene uporabnike iz tabele „online“ ter še vsoto in zapiše to v zadnjo tabelo

Slabost intervalnega izvajanja PHP kode pa je ta, da jo mora intervalno poganjati sistem. Na UNIX platformah se to doseže s „cronjob“-om.V cron napišemo interval ter ukaz, ki želimo da se izvaja. To pa pomeni, da mora ponudnik spletnega prostora omogočati dostop do crona. To pa zna biti problem varnosti z vidika ponudnika, na katerem gostuje spletno mesto.

Zato smo skripto prevedli še v MySQL. Implementacija dogodkov (angl. Event) je bila predstavljena pri verziji 5.1 in je zelo enostavna.

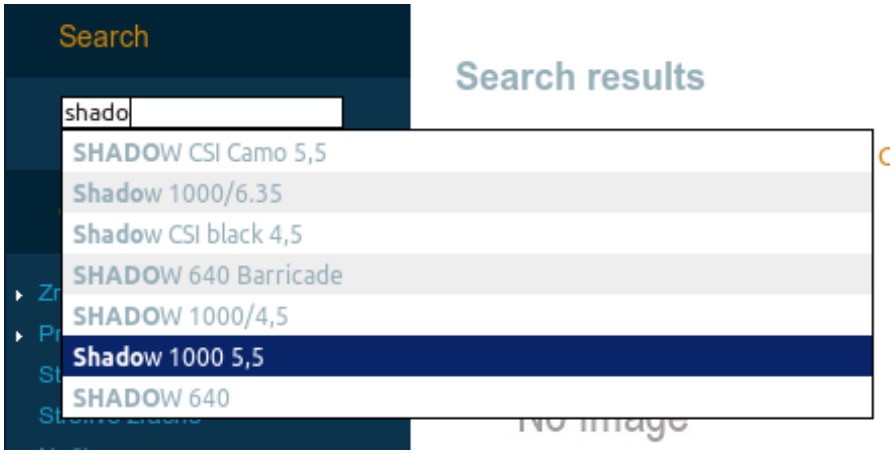
Dogodek se lahko izvrši enkratno, torej ob točno določenem času. Lahko pa je intervalen. Določimo interval, v našem primeru 10 minut. Dodatna dva parametra pa sta, kdaj želimo, da se dogodek začne in kdaj konča.

3 ISKALNIK ARTIKLOV

Implementirana spletna trgovina omogoča preprosto iskanje artiklov in napredno iskanje z dodatnimi kriteriji.

3.1 Preprosto iskanje

Iskalnik je sestavljen iz vnosnega polja, v katero uporabnik vpiše iskan artikel. Iskalnik uporablja zahteve AJAX. Zahteva se pošlje ob tipkanju, pri čemer uporabljamo dogodek onkeyup. Da ne bi preveč obremenili strežnika z nepotrebnimi zahtevki, uporabljamo časovni zamik in zahtevo pošljemo šele takrat, ko uporabnik preneha pisati za določen čas. Poleg tega se zahtevki ne pošljejo, dokler ni uporabnik vtipkal vsaj treh znakov (Slika 10).



Slika 10. Ponujeni artikli ob delnem iskanju.

Na strežniški strani je potrebno pripadajoče artikle poiskati v podatkovni bazi. Primer klica, ki se izvede ob tem, je takšen:

```
SELECT * FROM items WHERE item_name LIKE '%shadow%' and item_deleted=0 and item_published=1
```

Zakaj LIKE in ne MATCH AGAINST? MySQL strežnik že ima integrirano iskanje nizov. Imenuje se MATCH AGAINST in deluje izredno hitro. Iskanje je mogoče samo nad indeksiranimi atributi. Torej bi moral dodati indeks za polje „item_name“ in sicer tipa full-text.

Več o hitrosti iskanja bomo povedali kasneje, ko se ponovno srečamo s tem problemom.

Slabost, ki je v našem primeru velika, je ta, da „fulltext“ ne indeksira nizov krajših od 4 znakov, torej bi moral uporabnik vpisati vsaj 4 znake, da bi iskanje delovala.

Druga, še večja slabost, pa je ta, da ne omogoča iskanje delnih nizov, če ne poznamo začetka.
Če iščemo „*adow“, bi načeloma pričakovali vse artikle, ki se končajo na „adow“, ampak žal temu ni tako. Zaradi same zasnove „fulltext“ indeksa to ni mogoče. Možno je le iskanje polnih besed, npr: „shadow“ ali pa delnih besed brez končnic, npr: „shad*“.

Če uporabnik vpiše v iskalnik „shadow“, se mu ponudijo možno artikli (Slika 10). Če na to pritisne tipko enter, mu bo spletna trgovina izpisala vse artikle, ki vsebujejo iskano v nazivu samega artikla. Če pa uporabnik ne pritisne enter, pa lahko iz med ponujenih artiklov izbere enega in se mu bo le-ta odprl. Če iskani niz vrne samo en artikel, se ta artikel ne izpiše kot rezultat iskanja, ampak uporabnika sistem avtomatsko preusmeri na podrobnejši ogled tega artikla.

3.2 Napredno iskanje in filtriranje

Preglednost pri brskanju po izdelkih je ključnega pomena. Uporabnik se ne zadržuje rad dolgo na spletni trgovini. Uporabniki hočejo najti ustrezen artikel kar se da hitro. Če uporabnik vsaj približno ve, kaj išče, potem je primernejše iskanje s filtri.

Preglednost artiklov običajno rešimo z mnogo kategorijami, s tem artikel nekako opišemo. Recimo zračna puško „Shadow 1000“ kalibra 5,5mm seveda spada v kategorijo „Orožje“. A če vse orožje zmečemo v kategorijo „Orožje“, je brskanje ter iskanje po puškah izredno nepregledno. Rešitev je seveda, da najprej naredimo hierarhijo kategorij. Primerno bi bilo, da naredimo tri kategorije tako:

Orožje => Zračno orožje => 5,5 mm

V zadnjo kategorijo bi vnesli naš artikel. Tako bi uporabnik, ki želi samo zračne puške kalibra 5,5, hitro našel ustreznega.

Še boljša rešitev pa je, da vsakemu izdelku priredimo razne lastnosti; v tem primeru bi zračni puški „Shadow 1000“ določili, da je kalibra 5,5. Namesto globoke hierarhije kategorij pa bi naredili samo eno globino, Orožje => Zračno orožje. Notri bi se nahajal artikel „Shadow 1000“.

Ko uporabnik klikne kategorijo „Zračno orožje“, se mu poleg vseh artiklov izpišejo še vse lastnosti z vrednostmi za to kategorijo (Slika 11).

Shadow 1000 bi poleg kalibra lahko pripisali še izhodno hitrost, težo, moč, dolžino cevi, tip pogona, material, iz katerega je izdelano kopito in podobno.

Slika 11. Primer lastnosti zračnih pušk.

Vsi ti filtri bi se uporabniku izpisali ob kliku na to kategorijo. Če nato uporabnik želi vse zračne puške kalibra 5,5 mm, lahko to označi in izpisali se bodo le artikli s to lastnostjo (Slika 11).

Če želimo dodati določeno lastnost artiklu, moramo najprej dodati novo lastnost neodvisno od artikla, ki to lastnost ima.

Obstajata dva tipa lastnosti in sicer lastnosti v obliki naštevnih podatkov in številske lastnosti.

Naštevne lastnosti nimajo mnogo vrednosti. Primer takšne lastnosti je, recimo, „pogon“ zračne puške, kjer so vrednosti lahko vzmet, stisnjen zrak, co2... Primer takšne lastnosti so tudi nekatere številske lastnosti, kot recimo kaliber, kjer ne moremo uporabiti metode kjer je potrebno ponuditi uporabniku vse obstoječe kalibre ne pa možnosti vnašanja poljubnih kalibrov.

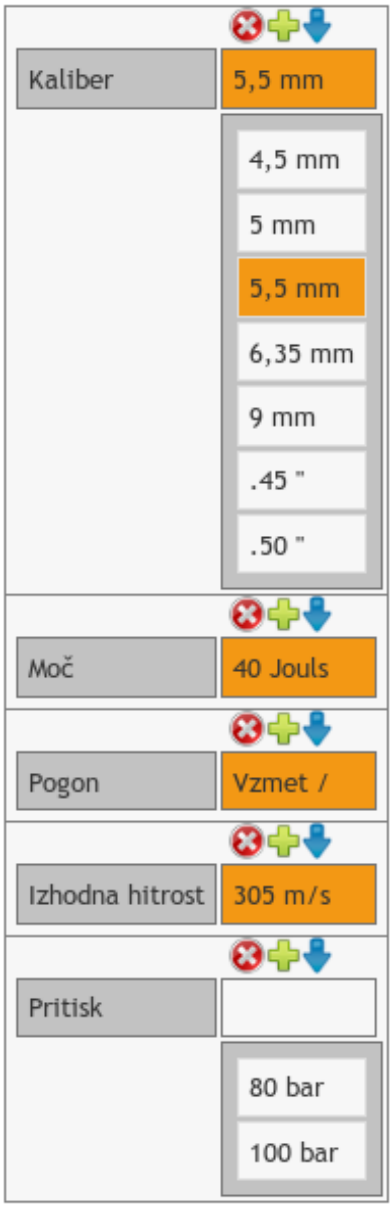
Številske lastnosti, kot recimo, „izhodno hitrost“ ali „moč“ puške administrator opiše tako, da ob dodaji lastnosti izbere tip „interval“ in poda največjo in najmanjšo možno vrednost, pri čemer lahko eno ali drugo izpusti; v tem primeru je najmanjša vrednost 0, največja pa neskončno.

Sistem vrednosti avtomatsko razvrsti po velikosti. Da to deluje pravilno, pa je predpogoj, seveda, da so vse vrednosti vnesene v isti meri. Zato administrator ob vnosu vrednosti določene lastnosti, samo prvič izbere mero, ob nadaljnjem vnašanju vrednosti pa je mera enaka kot ob prvem vnosu. Če administrator spremeni mero določeni vrednosti, se prav tako mera zamenja vsem ostalim vrednostim.

Lastnosti je potrebno dodeliti kategorijam, da aplikacija ve, katere lastnosti mora izpisati, ko uporabnik klikne na določeno kategorijo. Izpišejo pa se samo tiste lastnosti kategorije ter samo tiste vrednosti, ki jih imajo artikli. Če imamo lastnost „kaliber“ in vrednosti 4,5, 5,5 ter 6,35 mm. Trenutno pa ni nobenega artikla, ki je kalibra 6,35, zato tega uporabniku ne ponudimo.

To bi seveda lahko rešili tudi tako, da namesto dodeljevanje lastnosti kategorijam, vsakič, ko uporabnik klikne na določeno kategorijo, pregledali vse artikle ter njihove lastnost in jih izpisali. Tako bi bil enak učinek brez dodatnega koraka. Ampak to ni edini razlog, zakaj je potrebno dodeliti lastnost kategorijam.

Ko administrator doda novi artikel, mora artiklu, poleg drugih stvari, dodati tudi lastnosti ter vrednosti. Ker pa vemo, v kateri kategoriji je artikel, s tem tudi vemo, kakšne lastnosti ima in administrator mora samo vnesti vrednosti za posamezno lastnost. Poleg tega so administratorju izpisane vse vrednosti določenih lastnosti, ki jih imajo drugi artikli in če trenutni artikel že ustreza ponujenim vrednostim, to vrednost izbere. Če pa temu ni tako, pa lahko doda novo vrednost.

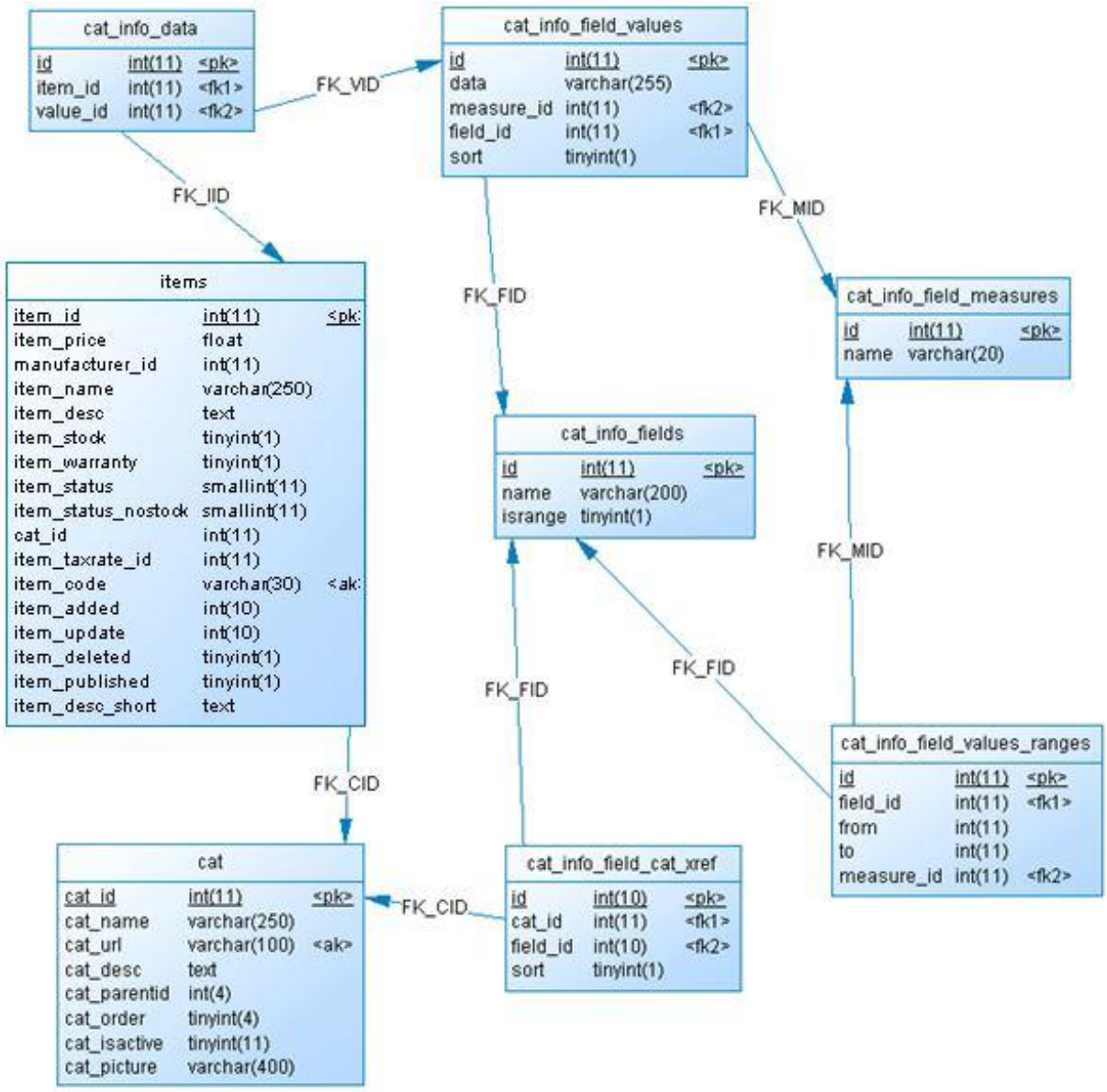


Slika 12. Na sliki so predstavljene vse lastnosti zračnih pušk. Za prvo in zadnjo lastnost pa vidimo tudi vrednosti, s tem, da zadnja lastnost ni izbrana, to se tudi vidi na Slika 12.

Prednost lastnosti je tudi ta, da lahko uporabniku, ko klikne na podrobnosti o artiklu, enostavno izpišemo tehnične podatke artikla tako, da izpišemo vse lastnosti ter vrednosti (Slika 13).

Slika 13. Izpis lastnosti v podrobnem opisu artikla na uporabniški strani.

Lastnosti so tudi osnova za izdelavo primerjalne opcije, kjer lahko uporabnik primerja več artiklov iste kategorije.



Slika 14. Tabele ter povezave med njimi potrebne za implementacijo lastnosti artiklov.

Lastnosti so sestavljene iz 8 tabel (Slika 14).

Tabela „**cat_info_fields**“ vsebuje vse lastnosti. Polje „israngе“ nam pove, ali je možno lastnost prikazati v intervalu.

V tabeli „**cat_info_field_values**“ so zapisane vse vrednosti lastnosti določenega artikla. Vsak zapis vsebuje vrednost lastnosti, ID mere, ID lastnosti ter položaj. Položaj uporabimo pri izpisu

vseh vrednosti določene lastnosti, na primer pri kategoriji.

Če je lastnost intervalna („isrange“ je pozitiven), potem imamo zapis v tabeli „**cat_info_field_values_ranges**“, kjer je zapisan ID lastnosti, od, do ter ID mere. Primer za lastnost »izhodna hitrost« imamo 7 zapisov (Tabela 1).

Tabela 1. Primer podatkov, ki jih vsebuje tabela »cat_info_field_values_ranges«

ID	**ID lastnosti**	**Od**	**Do**	**ID mere**
1	2	*null*	150	1
2	2	151	180	1
3	2	181	200	1
4	2	201	250	1
5	2	251	300	1
6	2	301	350	1
7	2	351	*null*	1

Ker ima zračno orožje neskončno mnogo število različnih izhodnih hitrosti, je to edini način, kako lahko izpišemo vse hitrosti. Ponudimo uporabniku izbiro intervala hitrosti.

V tabeli „**cat_info_field_cat_xref**“ se nahajajo IDji kategorij, IDji lastnosti ter vrstni red. Iz te tabele preberemo, ko uporabnik klikne določeno kategorijo, katere lastnosti ima in jih izpišemo po vrstnem redu.

Tabela „**cat_info_field_measures**“ je enostavna; v njej so zapisane vse oznake mer.

In še zadnja, verjetno najbolj pomembna tabela, „**cat_info_data**“, ki povezuje artikle z lastnostmi. V njej so zapisani IDji artiklov ter IDji vrednosti lastnosti („cat_info_field_values“).

4 NEKAJ OPRAVIL V ZVEZI S SEZNAMI ARTIKLOV

Za administratorje je izredno pomembno, da je izpis artiklov pregleden, da deluje hitro ter da je urejanje enostavno. Pomembno je tudi, da je omogočeno iskanje po nazivu ali po šifri izdelka.

4.1 Iskanje

Kot že omenjeno ima MySQL že implementirano iskanje nizov, ki se imenuje MATCH AGAINST in se izvaja veliko hitreje od operatorja LIKE. Razlika v hitrosti je ogromna. Razlika je opazna predvsem pri velikem številu artiklov, recimo več kot milijon. Poskusimo iskanje z LIKE nad tabelo s 1.100.859 vnosov.

```
mysql> select SQL_NO_CACHE count(*) from items where item_name like "shado%"\G;
count(*): 7
1 row in set (1.24 sec)
```

Torej poizvedba je vrnila 7 zadetkov in je trajala 1,24 sekunde.

Z uporabo MATCH AGAINST pa:

```
mysql> select SQL_NO_CACHE count(*) from items where match(item_name) against('shado*' in boolean mode)\G;
count(*): 12
1 row in set (0.00 sec)
```

Razlika je ogromna. Poleg tega je zanimivo še število vrstic, ki sta jih vrnili poizvedbi: poizvedba z uporabo LIKE-a je vrnila 7 vrstic, MATCH AGAINST pa 12.

Razlog se skriva v tem, da LIKE gre skozi vse zapise v bazi, in pogleda naziv artikla, ali se ta začne na „shado“. MATCH AGAINST pa išče po drevesu, ki se gradi ob vnosih in spremembah,

v drevesu pa so vse besede, daljše od treh znakov, ki jih vsebuje naziv artikla. Zato je MATCH AGAINST našel tudi tiste artikle, ki se ne začnejo na „shado“ ampak imajo v nazivu besedo, ki se začne tako. Primer je sledeč: „Gamo Shadow RSV“.

Če bi želeli implementirati iskanje LIKE, da bi delal po takšnem principu kot MATCH AGAINST, bi morali iz naziva pobrati vse besede z regularnim izrazom ter šele na to iskati, če se katera beseda začne z „shado“, kar bi čas poizvedbe povečalo za toliko, da bi bilo verjetno iskanje popolnoma brez smisla.

MATCH AGAINST pa ima v praksi dva problema, ne indeksira besede krajše od 4 znakov ter ne moremo iskati zadkov; na primer iskanje „*adow“ ne bo vrnilo zadetkov.

4.2 Urejanje

Za osnovne podatke artikla, kot so naziv, cena ipd (Slika 15). mora biti urejanje omogočeno na trenutni strani, torej na spisku artiklov in ne, da je potrebno odpreti podrobnosti o enem artiklu in tam spreminjati podatke (Slika 16).

To se doseže z JavaScriptom. Lahko gremo še en korak dlje. Namesto, da bi vrednosti za vsa polja, kot so naziv, cena, šifra ipd. zapisali v vnosno polje, to storimo samo, ko uporabnik dvakrat klikne na tekst.

Koda	Ime	Proizvajalec	Kategorija
68	SHADOW 1000/4,5	Gamo	Puške
69	Shadow 1000 5,5	Gamo	Puške
70	SHADOW 640	Gamo	Puške

Slika 15. Primer izpisa artiklov.

Koda	Ime	Proizvajalec	Kategorija
68	SHADOW 1000/4,5	Gamo	Puške
69	Shadow 1000 5,5	Gamo	Puške
70	SHADOW 640	Gamo	Puške

Slika 16. Primer urejanja imena artikla.

S tem smo naredili spisek artiklov izredno pregleden, saj ni nikjer nobenega vnosnega polja in hkrati izredno praktičnega, saj uporabnik za urejanje samo dvakrat klikne na polje in že se to spremeni v vnosno in je pripravljeno za urejanje. Shranjevanje sprememb poteka preko AJAXa in sicer, ko uporabnik pritisne tipko „Enter“.

Vsak artikel ima dve ceni. Neto cena, to je cena brez davka, ter bruto cena, cena z davkom. Obe ceni lahko administrator spremeni, prav tako lahko zamenja stopnjo davka.

Če uporabnik spremeni neto ceno, se bruto cena avtomatsko popravi po enostavni formuli

neto*davek=bruto

Podobno se zgodi, če uporabnik zamenja davčno stopnjo, kjer se poleg davka zamenja tudi bruto cena. Vse vrednosti pa se posodabljajo z zahtevki AJAX.

Kot že omenjeno, spletna trgovina omogoča izbiro valut. Valute pa imajo različne znake za ločevanje decimalk ter različne znake za ločevanje tisočic. Ko uporabnik spremeni ceno artiklu, se zgodijo sledeče stvari

1. Uporabnik dvakrat klikne na bruto ceno, ki je „1.012,68€“.
2. To polje se sedaj spremeni v vpisno in sicer ima vrednost „1.012,68“ (znak izgine). Za vse to je poskrbela JavaScript.
3. Sedaj uporabnik vpiše novo ceno, recimo, da vpiše „1.100,23“ in pritisne tipko „Enter“. Uporabnik bi lahko ceno zapisal tudi brez ločila za tisočice (glej točko 5.).
4. JavaScript pošlje AJAX zahtevek. Pošlje se ID artikla, ime polja ter vrednost, ki je

bila spremenjena.

5. PHP mora najprej preoblikovati ceno v pravilni zapis in sicer v tip „Double“. To se naredi enostavno. Najprej pogleda v bazo, katera valuta je prevzeta, nato zamenja znak za tisočice prevzete valute s praznim nizom, torej zamenja „.“ s „“, tako dobimo „1100,23“. Potem pa še znak za decimalke s piko, torej „,“ s „.“, tako, da dobimo 1100.23, kar je tipa Double. Nato to vrednost vpiše v bazo ter vrne formatirano številko: „1.100,23€“.
6. JavaScript spremeni vnosno polje nazaj v tekstovnega, nato pa zapiše novo vrednost, ki jo dobi od AJAX zahtevka.

Če je prevzeta valuta na primer USD, potem se spremeni formatirani zapis cene tako, da se seveda zamenja znak valute, poleg tega pa se spremeni tudi pozicija znaka, ki je na levi. Tisočice ločimo z vejico ter decimalke s piko.

4.3 Številčenje

Izredno pomembno je, da imamo strani oštevilčene in, da ne vrnemo vseh zadetkov na eni strani, saj bi s tem obremenili strežnik ter prav tako klienta, ker bi predolgo nalagal vse artikle.

Možna sta dva pristopa. Prvi je, da številčenje naredimo na uporabniški strani, torej z JavaScriptom, druga pa, da to naredi strežnik.

Prvi pristop je primeren, če je baza relativno majhna. Strežnik vrne vse zapise iz tabele artiklov, klientov brskalnik pa s Javascriptom uporabniku prikaže samo prvih, recimo, 10, ostale pa skrije. Nato uporabnik klika po navigaciji, tipično „naprej“ ter „nazaj“. To deluje izredno hitro, saj so podatki že na klientovi strani in jih ni zahtevati od strežnika. Problem pa se pojavi, ko imamo več zapisov, že par tisoč je problem. Če je računalnik starejšega tipa in nima dovolj sposobne strojne opreme, se problem pojavi že veliko prej.

Ta pristop bi bil uporaben samo, če bi brskalnik pošiljal zahteve po več zapisov. Recimo, da imamo v bazi milijon zapisov, tako bi jih brskalnik zahteval le prvih 1000 in iz tega naredil

navigacijo. Ko uporabnik pride do zadnje strani, pa zahteva ponovno naslednjih 1000 zapisov in tako naprej.

Drugi pristop pa je ta, da številčenje strani prepustimo strežniku. V tem primeru pa brskalnik vedno prikaže samo toliko zapisov, kot jih uporabnik lahko/hoče videti, tipično od 10 do 100 zapisov na enkrat. V tem primeru pa ne rabimo skrbeti, da bi bil računalnik, ki ga uporablja klient prepočasen, saj prikazuje malo število podatkov.

Pri tem pristopu pa se pojavi drugačen problem in sicer, kako optimizirati MySQL poizvedb.

Najslabši način je, da poizvedba vrne vse zapise in nato PHP vrne samo potrebne zapise.

```
SELECT SQL_NO_CACHE item_id FROM items where item_deleted=0 order by item_id asc limit 100000,10;
array_slice($data,($page-1)*$limit,$limit,true);
```

Zgornja zahteva se je končala v 5,5 sekunde. Izredno slabo.

Drugi pristop, ki je mnogo hitrejši, pa je ta, da sestavimo tako poizvedbo, ki vrne samo potrebne zapise. To lahko dosežemo z SQL ukazom LIMIT.

LIMIT sprejme en ali dva argumenta in sicer N in M in sta tipa INT. Če LIMIT napišemo samo z enim argumentom, se dejansko zgodi to, da se prvemu dodeli 0 in drugemu naša vrednost.

Torej LIMIT 5 je isto kot LIMIT 0,5, kar se bo zgodilo, je, da bo MySQL vrnil prvil 5 zadetkov. LIMIT 20,5, bo pa vrnil 5 zadetkov od 20. naprej.

Če želimo izpisati 10 artiklov in sicer od 10000 naprej naredimo tako:

```
SELECT SQL_NO_CACHE item_id FROM items where item_deleted=0 order by item_id asc limit 100000,10;
```

Zgornja poizvedba se je izvajala 1,51 sekunde. Še vedno ni najboljše.

Pri uporabi prvega načina, kjer je baza vrnila vse zapise, PHP pa je vrnil, to, kar je uporabnik zahteval, je prednost ta, da lahko uporabniku tudi izpišemo, koliko je vseh artiklov. In ker vemo koliko imamo artiklov, lahko tudi izpišemo, koliko strani je.

Ko pa uporabimo LIMIT, pa tega podatka nimamo in za to tudi ne vemo, koliko strani imamo. Vse, kar lahko storimo je, povezavo za naslednjo stran ter prejšnjo.

Problem pa lahko rešimo z uporabo „SQL_CALC_FOUND_ROWS“, ki zapiše število vrstic, za pridobitev rezultata pa je potrebno poklicati dodatno poizvedbo.

```
SELECT SQL_NO_CACHE SQL_CALC_FOUND_ROWS item_id FROM items where item_deleted=0 order by item_id asc limit 100000,10;
SELECT FOUND_ROWS()
```

Vse to pa seveda ni poceni, saj poveča čas poizvedbe.

Številčenje strani z LIMIT M,N nikakor ni dobra ideja. Slabosti so naslednje.

- Pri velikem odmiku bo povečal aktivni seznam, ker mora rezultate prenesti v primarni spomin sistema, ki pa niso nikoli vrnjeni uporabniku,
- poizvedba bo še bolj počasna, ko bo baza večja od razpoložljivega pomnilnika,
- majhna količina poizvedb z visokim odmikom lahko popolnoma obremenimo I/O diska (ozko grlo),
- če želimo izpisati „30-40 od 1.000.000 zadetkov“ bo moral MySQL prešteti 1.000.000, kar vzame ogromno časa.

Potrebno se je izogniti „Count(*)“ poizvedbi.

- Namesto skupno število zadetkov se prikažita samo gumba „naprej“ ter „nazaj“.

- Ne sešteva se skupno število zadetkov ob vsaki zahtevi; uporabnika ne zanima, če je 532764 ali pa 532664 zadetkov, temveč se raje prikaže „30-40 od mnogo“.
- število vrstic se shrani in zmanjša oz. zveča pri brisanju/dodajanju zapisov.

Če se želimo znebiti odmika, se je potrebno znebiti direktnih skokov na N-to stran.
LIMIT N je v redu, LIMIT M,N je izredno slabo. Namesto Mja potrebujemo namig, kje začeti stran. Poiskati moramo potrebne zapise z uporabo restriktnega WHERE-a z uporabo namiga in ORDER BY ter LIMIT N (brez odmika!).

Tabela 2. Primer iskanja namiga.

10 11 12 13 **14**	Stran 1 Naslednja stran
15 16 17 18 **19**	Stran 2 Prejšnja stran Naslednja stran
20 21 22 23 **24**	Stran 3 Prejšnja stran Naslednja stran

In sedaj zgornji HTML (Tabela 2) prevedemo v SQL poizvedbe.

Naslednja stran
Naslednja stran

WHERE **id>14** /*zadnji viden*/

ORDER BY id **DESC** LIMIT 5 /* brez odmika!!!*/

In še v drugo smer

 Prejšnja stran

WHERE **id<20** /*zadnji viden*/

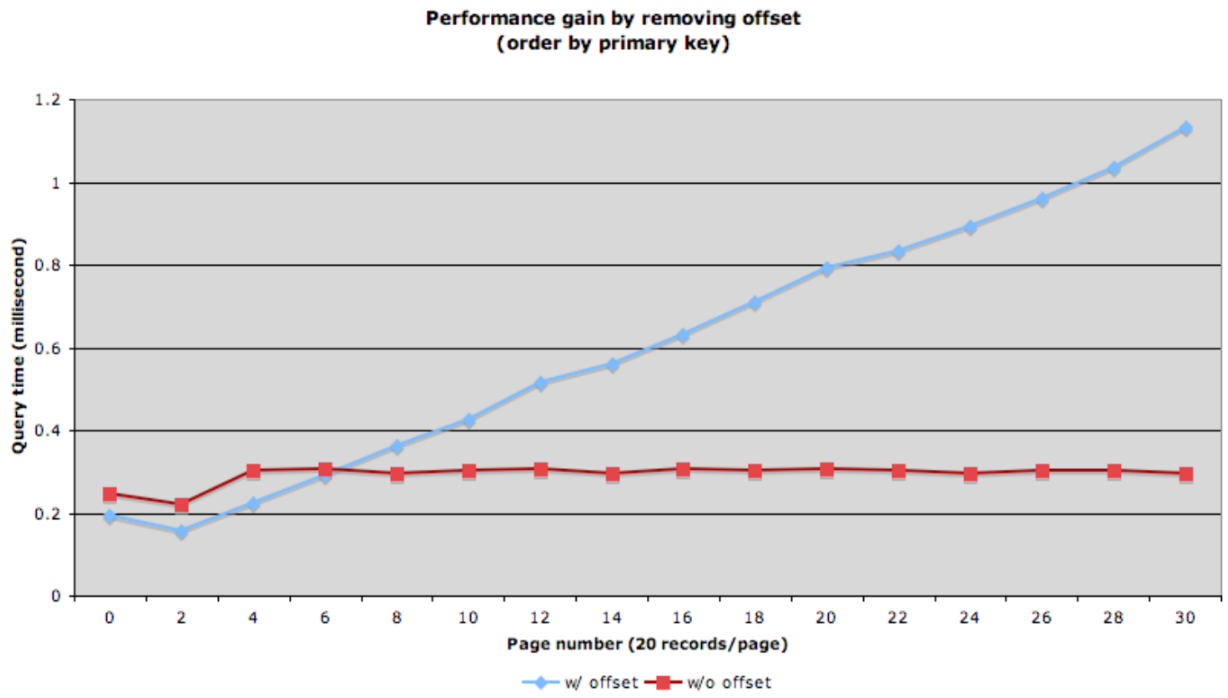
ORDER BY id **ASC** LIMIT 5 /* brez odmika!!!*/

V tem primeru moramo rezultate še obrniti, sicer bodo v obratnem vrstnem redu.

Poizvedba

SELECT SQL_NO_CACHE item_id FROM items where item_deleted=0 and item_id>1382446 order by item_id asc limit 10;

se je izvajal 0,36 sekunde, kar je odlično. Artikel z IDjem 1382446 se nahaja na 100000. mestu.



Slika 17. Prikaz hitrosti nalaganja strani (x os) v relaciji s časom (y os). Modra je z odmikom (LIMIT M,N), rdeča pa brez odmika.

Ostale slabosti uporabe LIMIT-a z odmikom pa so, da se medtem, ko uporabnik bere, lahko pojavijo novi vnosi podatkov.

- Če podatke prikazujemo padajoče, torej novejše najprej in se zgodi izbris vnosa ali nov vnos, se bodo zapisi premikali naprej/nazaj. Če je uporabnik, na primer na peti strani in branjem, je pride do novega vnosa (na stran 1 – novejši najprej), se bodo strani zamaknile za en vnos naprej. Ko uporabnik klikne stran 6, je prvo sporočilo že videl na prejšnji strani.

Številčenje naslednjih in prejšnjih lahko implementiramo tako, da:

- preberemo nekaj več vrstic, kot jih potrebujemo, tako, da lahko naredimo recimo 5 strani naprej in 5 nazaj,
- uporabimo majhni odmik, tako, da ne preberemo dodatne podatke v naprej ampak z obstoječim namigom dodamo odmik, tako, da omogočimo naslednjih par strani. Primer:

  trenutna stran: WHERE id>10 ORDER BY id DESC LIMIT 5
  naslednja: WHERE id>10 ORDER BY id DESC LIMIT 5,5
  naslednja: WHERE id>10 ORDER BY id DESC LIMIT 10,5
  ...

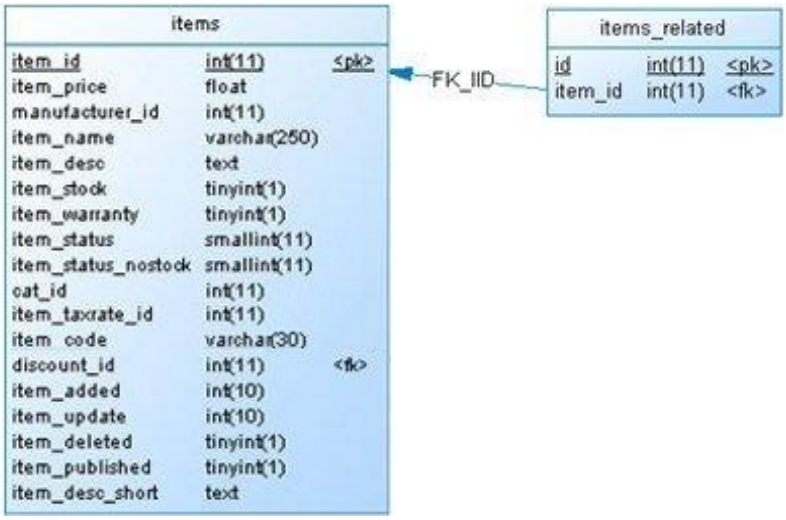
4.4 Podrobnosti o artiklih

Do podrobnosti o artiklih lahko administrator pride tako, da najprej odpre seznam vseh artiklov, nato pa klikne na gumb „uredi“, ki se nahaja pri vsakem artiklu. Podrobnosti se razdelijo v 4 zavihke.

- Splošno
- Slike
- Lastnosti
- Podobni izdelki

V prvem zavihku smo večino polj že opisali pri poglavju „Spisek artiklov“. Dodatna polja so recimo, trajanje garancije, dobavljivost, kratki opis ter dolgi opis. Zavihek „lastnosti“ smo tudi že predstavili.

Podobni izdelki pa so artikli, ki se navezujejo na trenutni artikel. Implementacija je preprosta (Slika 18). Administrator izbere za določen artikel, podobne oziroma artikle, ki se mu zdijo, da si jih bi uporabnik rad ogledal.



Slika 18. Tabeli ter povezava za implementacijo podobnih artiklov.

Slike artiklov na spletni trgovini so ključnega pomena. Ker pa je veliko artiklov ter še več slik, je pomembno, kako administrator naloži slike za posamičen artikel. Bistvo nalaganja je, da je enostavno in učinkovito. Običajno nalaganje slik izgleda tako, da:

- uporabnik izbere sliko iz svojega računalnika in klikne „v redu“, ter potrdi obrazec
- brskalnik pošlje sliko strežniku,
- strežnik jo shrani in brskalnik osveži stran.

Slika pa mora biti v naprej prilagojena po velikosti ter ožigosana.

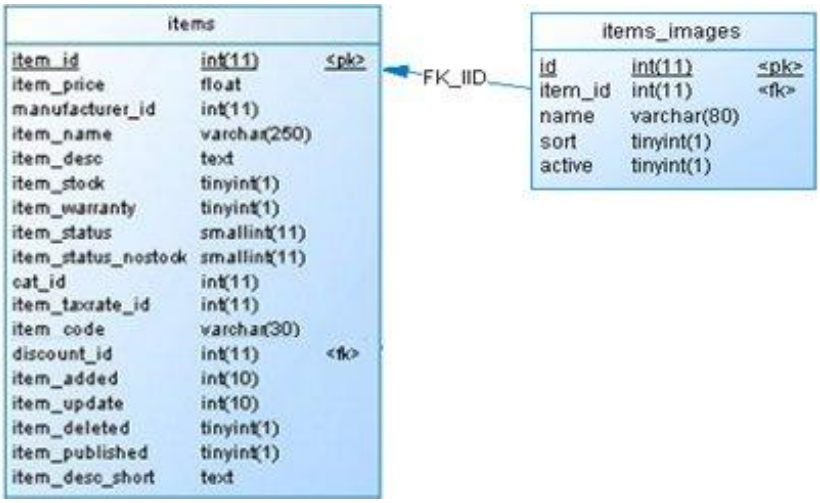
Naša implementacija je rahlo drugačna ter izpopolnjena in omogoča:

- nalaganje slik iz diska ter spleta (potrebujemo povezavo do slike),
- obrezovanje slik,
- osveževanje,
- brisanje slike,
- onemogočanje slike,
- sortiranje slik,
- več dimenzij slik ter
- žigosanje.

Prvi korak lahko izvedemo na dva načina.

Prvi deluje preko okna za izbiro slike iz uporabnikovega računalnika. Ko uporabnik izbere sliko in potrdi izbiro s klikom na gumb „v redu“, se slika avtomatsko pošlje strežniku, za pošiljanje pa uporabi AJAX. Ko strežnik shrani sliko, se slika avtomatsko pojavi na trenutni strani. Tako lahko administrator hitro naloži več slik, brez da bi se mu celotna stran osvežila.

Drugi način je nalaganje slik iz spleta. Po navadi, ko dobimo nov izdelek, poiščemo sliko na spletu. Če sliko najdemo, nam je ni potrebno shranjevati na disk, ampak lahko povezavo do slike prilepimo v okno, za nalaganje slik iz spleta in ta se bo naložila na strežnik in prikazala. To omogoča še hitrejše nalaganje, saj preskočimo dva koraka, shranjevanje slike ter ponovno izbiranje slike za nalaganje.



Slika 19. Struktura tabele slik ter povezava z tabelo artiklov.

Tabela slik je preprosta (Slika 19). Sestavljena je iz ID artikla, imena, zaporedne številke ter polja, ki nam pove, ali je slika aktivna ali ne. Aktivna slika je vidna vsem, neaktivna pa samo administratorjem.

Ko administrator naloži novo sliko artikla na strežnik, je ta neaktivna. Neaktivne slike so prikazane svetlejše oziroma bolj bledo. To smo dosegli z CSSjem. Če je slika neaktivna se ji določi razred (angl. Class), ki vsebuje stopnjo transparence nastavljeno na 0.2, torej 20 % (Slika 20). Administrator enostavno spremeni status s klikom na gumb „X“ (križec). Z AJAXom se pošlje zahteva, ki vsebuje ID slike ter status, PHP zapiše nov status v bazo, križec se spremeni v kljukico ter transparenca izgine. Tako je slika vidna tudi na uporabniški strani.

Slika 20. Neaktivna slika.

Če slika ni takšna, kot jo uporabnik želi, jo lahko obreže. S klikom na gumb za obrezovanje slik se odpre slika v originalni velikosti. Z miško enostavno označi predel, ki ga želi obdržati. S

klikom na „Shrani“, se pošljejo podatki o obrezanem polju strežniku, le-ta sliko obreže in shrani na disk. Ta zahteva je poslana preko AJAXa, tako, da se slika avtomatsko zamenja z novo, brez osveževanja celotne strani.

Tukaj pa smo naleteli na problem. Brskalniki si vse slike zapomnijo v predpomnilnik. Posledica tega pa je ta, da ko smo sliko obrezali ter shranili, se ta ni spremenila v brskalniku, ker je brskalnik bil mnenja, da gre za isto sliko, saj ima isto ime.

Rešitev je seveda trivialna. Ko Javascript dobi odgovor od zahtevka, ki ga je poslal brskalnik preko AJAXa (ko uporabnik klikne „shrani“), spremenimo atribut „src“ v elementu „img“ tako, da poleg poti do slike dodamo „?“ ter naključen niz. Uporabili smo kar trenuten čas v sekundah (angl. Timestamp). S tem prisilimo brskalnik, da na novo naloži sliko.

Administrator lahko tudi izbriše sliko, če to želi. S klikom na gumb za brisanje se slika izbriše iz baze ter iz diska.

Ob naložitvi se vsaka slika kopira v večih dimenzijah in nato shrani na disk. Dimenzije slik določi administrator v nastavitvah. Natančen potek shranjevanje je naslednji:

1. Najprej generiramo naključno ime slike po postopku izvlečka (angl. Hash). In sicer za vrednost vzamemo trenutni čas v mikrosekundah ter ime trenutne slike.
2. Sliko nato shranimo v mapo „original“, kjer imamo vse originalne slike.
3. Sliko je potrebno shraniti še v bazo. Zaradi vrstnega reda, najprej potrebujemo zadnjo sliko za ta artikel, nato pa vstavimo novo sliko z zaporedjem povišanim za ena.
4. Nato pa še v zanki sprehodimo po dimenzijah, ki jih je določil administrator ter sliko spremenimo v trenutno dimenzijo in shranimo v mapo z istim imenom.

Ko sliki spreminjamo dimenzije, je pomembno, da ohranimo razmerje med višino in širino, sicer lahko slika postane preširoka ali preozka.

Če je administrator določil eno izmed dimenzij 700x400 pikslov, slika pa ima dimenzije

1000x500 pikslov se zgodi naslednje.
Najprej izračunamo razmerje širin in višin, torej

700/1000=0.7, ter

400/500=0.8.

To nam pove, za koliko procentov moramo zmanjšati sliko. Torej če želimo širino spraviti na 700 pikslov moramo trenutno širino zmanjšati za 30 %. Višino pa 20 %.

Vidimo, da če želimo ohraniti razmerje, moramo obe dimenziji, širino in višino zmanjšati za 30 %.
Torej naredimo:

1000*0.7=700 pikslov, ter

500*0.7=350 pikslov.

Tako smo dobili novo dimenzijo slike, ki ustreza želeni dimenziji ter je še vedno v razmerju.
Če širina niti višina slike ni večja od želene širine in višine, ne storimo ničesar, saj v vsakem primeru izgubimo na kvaliteti, tudi, če ostane v razmerju.

Če se administrator odloči, da bo naložil sliko z transparentnimi elementi (transparenco omogočata formata PNG ter GIF), se tudi ta ohrani v vseh dimenzijah.

Slika se vedno ohrani v istem formatu, kot jo je naložil administrator.

Administratorju je omogočena tudi možnost, da žigosa slike. V primeru, da gre za avtorske slike, je priporočeno, da se na njih doda žig.

Za žig lahko administrator izbere tekst ali sliko. Lahko tudi izbere pozicijo žiga in sicer v vseh kotih ali pa sredinsko. Za žig pa lahko izbere tudi sliko; v tem primeru lahko pa lahko sliko pozicionira sredinsko (Slika 22), ob čemer se spremeni tudi velikost žiga.

Slika 21. Primer nežigosane aktivne slike artikla.

Slika 22. Primer žigosane aktivne slike artikla s sliko poravnano sredinsko.

Gumb osveži pa ponovno shrani sliko v določene dimenzije ter jih požigosa. Če trenutna slika ni žigosana (in je žigosanje izklopljeno) in administrator želi od sedaj naprej žigosane slike, mora najprej vklopiti žigosanje. Ker pa ob shranjevanju trenutne žigosanje ni bilo vklopljeno, slika nima žiga. Če pa želimo tudi to sliko žigosati, pritisnemo gumb „osveži“.

Ko pritisnemo gumb, se slika ponovno shrani v vse dimenzije, ki jih je določil administrator. Če pa administrator spremeni dimenzije ali pa izbriše/doda dimenzijo, se pa pojavi problem. Po kliku gumba „osveži“, se bo pretvorila v nove dimenzije. Stare bodo ostale na disku. Zato se namesto, da bi se pred osveževanjem sprehodili v zanki skozi vse dimenzije, ki jih je določil administrator (baza), sprehodimo po disku in izbrišemo sliko iz vsake mape razen iz mape „original“. Nato shranimo sliko v dimenzije, ki jih je določil administrator.

Administrator lahko tudi sortira slike po poljubnem vrstnem redu. Prva slika v vrsti je prevzeta slika artikla. Prevzeta slika pa se na primer pojavi v uporabniškem delu, pri zadetkih iskanja. Administrator zamenja vrstni red z enostavnim premikom slike na določeni položaj. Z miško klikne na sliko in gumb drži, jo premakne na želeno mesto in spusti (angl. Drag and drop). Ob spustu se sproži dogodek, ki pošlje preko AJAXa vrstni red slik za določeni artikel.

5 KOŠARICA IN IZVEDBA PLAČILA

Košarica na spletni trgovini ima enak namen kot prava košarica. V njej ima uporabnik vse izdelke, ki jih ima namen kupiti.

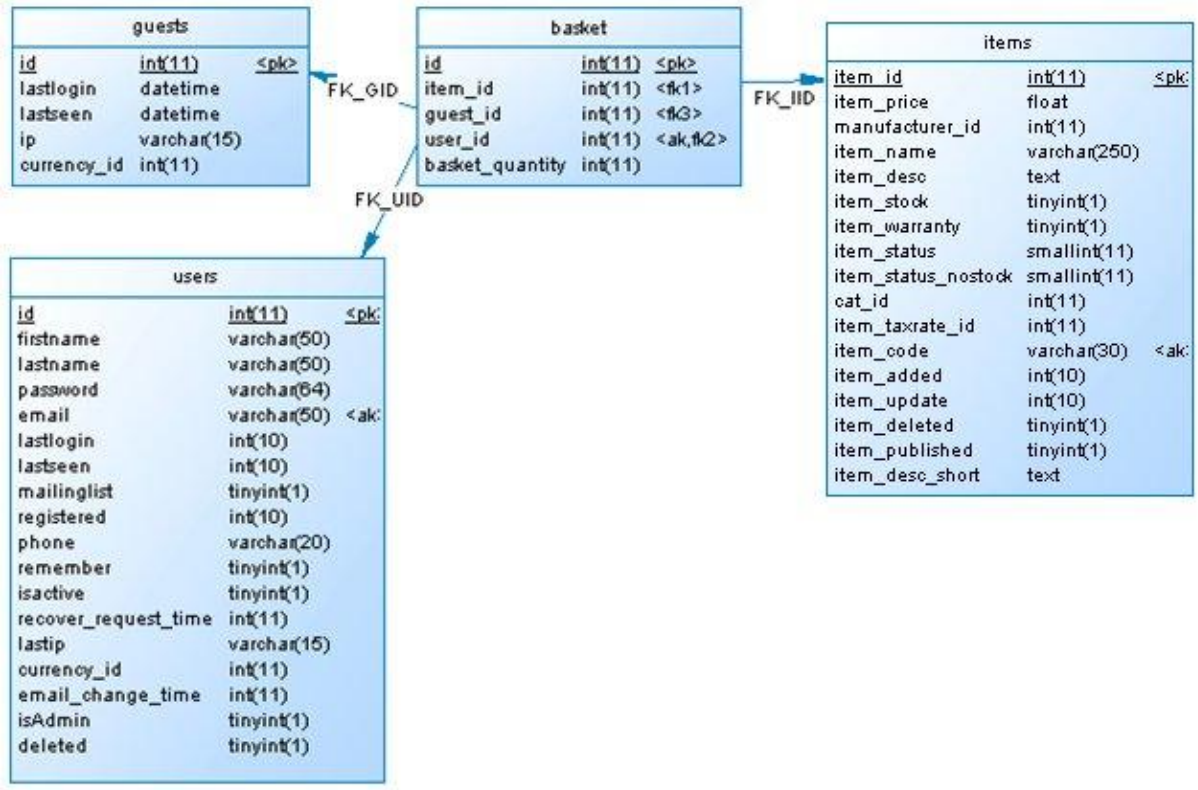
Dostopnost košarice je ključnega pomena. Uporabnik mora imeti vedno pregled nad njo. Zato košarica vedno „plava“ na vrhu brskalnika, ne glede na to, kako nizko gre uporabnik z drsnikom (Slika 23). Košarica pokaže svojo vsebino takrat, ko gre uporabnik z miško na njo in tam počaka določen čas (Slika 24).

V njej so izpisani artikli, količina ter cena za en kos. Količino lahko spreminjamo tako, da v okence vpišemo novo količino in pritisnemo enter ali pa gumb „osveži“. Posebnost osveževanja količine je ta, da uporabniku ni potrebno ob vsaki spremembi osvežiti količino, ampak lahko za vse artikle spremeni količino in na koncu klikne „osveži“ ali enter in količina se osveži za vse artikle. Če uporabnik klikne gumb „Zbriši“, se ta artikel izbriše iz njegove košarice. Vse spremembe se seveda dogajajo s klici AJAX in tako dajo uporabniku prijeten občutek, saj ni potrebe po osveževanju celotne spletne strani. Končni znesek se prav tako zamenja preko AJAXa.

Slika 23. Skrita košarica na vrhu brskalnika.

Slika 24. Odprta košarica ter njena vsebina.

Sestava tabel je enostavna (Slika 25). V košarici hranimo uporabniški ID. Prvo polje je ID artikla. Če je uporabnik prijavljen, potem je njegov ID zapisan v polju „user_id“, polje „guest_id“ pa je prazno oziroma brez vrednosti (angl. Null). V primeru, da uporabnik ni prijavljen, pa se vrednosti obrnejo; „user_id“ je prazen ter „guest_id“ vsebuje ID uporabnika. Zadnje polje pa je količina določenega izdelka, ki ga ima uporabnik v košarici. Več informacij pridobimo s preprosto poizvedbo, kjer združimo vse prikazane tabele (Slika 25).



Slika 25. Slika tabel ter povezav za implementacijo košarice.

5.1 Cene izdelkov

Vsak izdelek ima ceno. Ta cena pa je sestavljena iz več delov.

Cena je najprej razdeljena v dve tabeli po valuti. Prva je v valuti, ki je prevzeta za spletno trgovino in jo nastavi administrator. Druga je v valuti, ki jo ima izbrano uporabnik. Če uporabnik ne spreminja valute, je ta tabela enaka privzeti. V notranjosti se nahajata še dve tabeli in sicer prva vsebuje podatke, ki so fiksni, oziroma ceno, kot jo je določil administrator. Druga tabela pa vsebuje podatke o ceni in niso fiksni, ampak se na podlagi uporabnika spreminjajo. Spreminja se, če je dotičen artikel v akciji, ali pa če ima uporabnik kupon.

Primer cene:

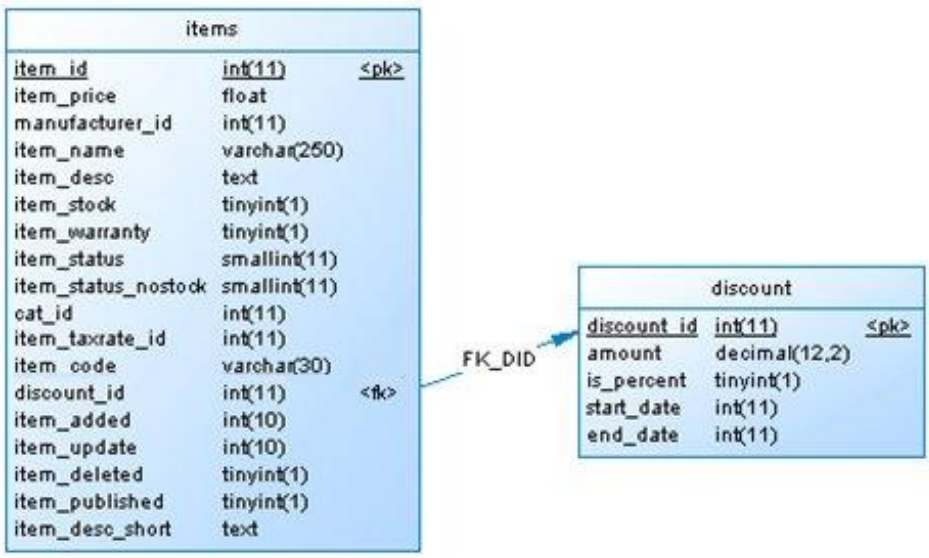
```
[price] => Array
        (
            [prevzeta] => Array
                (
                    [normalna] => Array
                        (
                            [notax] => 199.17
                            [notax_format] => 199,17€
                            [price] => 239.004
                            [price_format] => 239,00€
                            [tax] => 39.834
                            [tax_format] => 39,83€
                        )
                    [vasa] => Array
                        (
                            [price] => 239.004
                            [price_format] => 239,00€
                            [notax] => 199.17
                            [notax_format] => 199,17€
                            [tax] => 39.834
                            [tax_format] => 39,83€
                        )
                )
            [valuta] => Array
                (
                    [normala] => Array
                        (
                            [notax] => 268.999002
                            [notax_format] => $269.00
                            [price] => 322.7988024
                            [price_format] => $322.80
                            [tax] => 53.7998004
                            [tax_format] => $53.80
                        )
                    [tvoja] => Array
                        (
                            [notax] => 268.999002
                            [notax_format] => $269.00
                            [price] => 322.7988024
                            [price_format] => $322.80
                            [tax] => 53.7998004
                            [tax_format] => $53.80
                        )
                )
            [davek] => Array
                (
```

```
                [rate] => 0.2
                [rate_format] => 20%
            )
    )
```

5.2 Popusti

Če bi temu artiklu določili akcijsko ceno in sicer popust za 10 %, bi cena izgledala tako:

```
[vasa] => Array
                    (
                        [price] => 215.1036
                        [price_format] => 215,10€
                        [notax] => 179.253
                        [notax_format] => 179,25€
                        [tax] => 35.8506
                        [tax_format] => 35,85€
                        [discount] => Array
                            (
                                [amount] => 23.9004
                                [percentage] => 10.00
                                [start_date] => 0
                                [end_date] => 0
                                [amount_format] => 23,90€
                                [percentage_format] => 10,00%
                            )
                    )
```



Slika 26. Sestava tabel ter povezava za prikaz popustov.

Bazo za popuste bi lahko zasnovali drugače. Lahko bi tabela popustov vsebovala ID artikla ter tabela artikla bi bila brez ID popusta. Tako bi imel za vsak znižan artikel en zapis v tabeli

popustov. Tako, kot pa smo jo implementirali mi, pa se lahko en popust navezuje na več artiklov, kar je veliko bolj smiselno (Slika 26).

„Amount“ je količina popusta in je lahko znesek popusta ali pa procent popusta; ali gre za znesek ali delež, pove spremenljivka „is_percent“.

Če želimo artikel znižati za 10 %, se zgodi naslednje:

```
//$discount['amount']=10;
$data['amount']=$priceWithTax-($priceWithTax*(1-$discount['amount']/100));
$data['percentage']=$discount['amount'];
```

Če pa hočemo artikel znižati za 50 EUR, se zgodi sledeče:

```
//$discount['amount']=50;
$data['amount']=$discount['amount'];
$data['percentage']=$discount['amount']/$priceWithTax*100;
```

„start_date“ ter „end_date“ pa nam pove, kdaj velja popust. Če je popust že potekel, se vsem artiklom s tem popustom nastavi „discount_id“ na vrednost „NULL“.

5.3 Kuponi

Kuponi so popusti, ki veljajo samo za kupce, ki imajo posebno kodo za kupon in veljajo samo za uporabnike, ki so registrirani na spletni trgovini. Kuponi so zapisani v tabeli „coupons“. Ko uporabnik vpiše kodo, se le-ta pošlje preko zahtevka AJAX. Če koda ni pravilna, se prikaže ustrezno sporočilo, drugače pa se uporabniku izpiše vrednost ter vrsta kupona.

Pri vnosu kupona velja isto kot pri popustu artikla; vrednost kupona je lahko vsota popusta ali procentualna vrednost.

Implementirali smo več vrst kupona (Tabela 3) (Slika 27).

Tabela 3. Prikaz možnih variant za kupon.

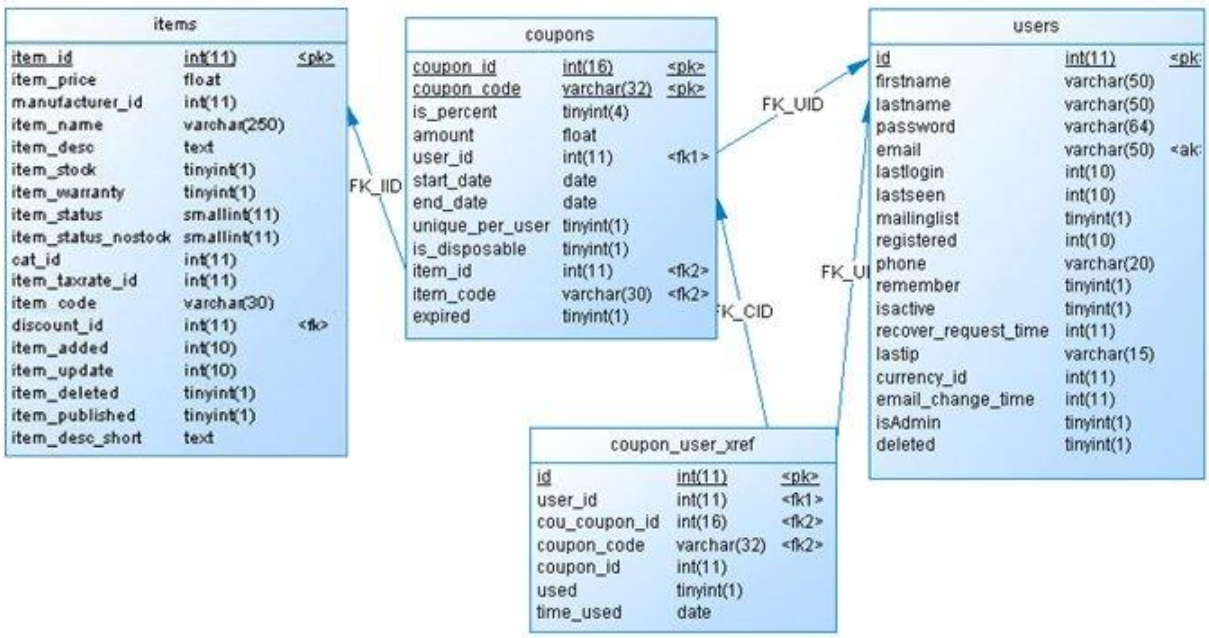
Unique_per_user	1	1	0
is_disposable	1	0	0

Če sta atributa „unique_per_user“ ter „is_disposable“ nastavljena na „true“, potem gre za bon. Bon je vrsta kupona, ki ga lahko uporabi samo en uporabnik in to samo enkratno. Prvi atribut onemogoči, da bi kupon aktiviral več kot en uporabnik, drug atribut pa poskrbi, da je po uporabi kupon izbrisan.

Če nastavimo prvi atribut na „true“, drugega pa na „false“, potem lahko kupon uporabijo vsi, ampak to samo enkrat.

Če pa nastavimo oba atributa na „false“, pa je kupon veljaven do preklica.

Če atributu „item_id“ nastavimo vrednost, oziroma vpišemo ID artikla, bo kupon možno vnovčiti samo na ta artikel, če pa pustimo vrednost na „NULL“, pa se kupon obračunava na končni znesek.



Slika 27. Prikaz tabel in povezav za implementacijo kuponov.

Če administrator želi kupon onemogočiti, lahko nastavi atribut „expired“ na „true“, če pa je nastavljen datum zaključka, pa se „expired“ nastavi na „true“ avtomatično, ko poteče.

5.4 Valute

Uporabniki imajo možnost izbiranja valute, v katerih spletna trgovina prikazuje zneske. Cene so preračunane povsod. Pri pregledu artikla, v košarici, v nadzorni plošči pri pregledovanju starih naročil ter tudi v emailu, ki je poslan ob oddaji naročila. Administratorji pa v administracijskem delu strani vidijo vse vsote v privzeti valuti.

Valute se enkrat dnevno osvežijo iz datoteke XML , ki se nahaja na Evropski Centralni Banki, na strani http://www.ecb.int/stats/eurofxref/eurofxref-daily.xml.

Program, ki osveži valute:

```
from db import db
from xml.dom import minidom
from xml.dom.minidom import Node
import urllib

dbc=db("usr", "pw", "db")
xmldoc = minidom.parse(urllib.urlopen('http://www.ecb.int/stats/eurofxref/eurofxref-daily.xml'))

for n in xmldoc.getElementsByTagName("Cube"):
    code=n.getAttribute("currency")
    rate=n.getAttribute("rate")
    current=dbc.selectLine("select * from currency where code='%s'" % code)
    if(current):
        dbc.execute("update currency set rate=%s where id=%s" %(str(rate), str(current['id'])))
    else:
        dbc.execute("insert into currency(name,code,rate) values('%s','%s',%s)" %(code, code, str(rate)))
```

Kako je sestavljena tabela valut, kaže Slika 28, primer dveh zapisov valut pa v Tabela 4.

Slika 28. Slika prikazuje tabelo v kateri so hranjene valute.

Tabela 4. Primer vnosa dveh valut in sicer EUR ter USD.

Atributi	**EUR**	**USD**	**Razlaga**
ID	1	2	
Name	Euro	US Dollar	Ime, ki je prikazano uporabnikom, ko izbirajo valuto
Code	EUR	USD	Koda valute po standardu ISO 4217
Rate	1	1.36	Razmerje v primerjavi z evrom
Sign	€	$	Znak
Dec	,	.	Znak, ki ločuje decimalni del
thousand	.	,	Znak, ki predstavlja tisočice
signIsRight	1	0	Ali je znak na desni strani
decnum	2	2	Število decimalk
default	1	0	Prevzeta valuta
updated	*null*	1297291526	Kdaj je bila vrednost nazadnje posodobljena

5.5 Plačilo

Če želimo imeti veliko prometa s spletno trgovino, se moramo čim bolje približati uporabniku ter mu na vse možne načine olajšati nakup. Le tako bo kupec zadovoljen in bo pripravljen izpeljati nakup do konca.

Ključnega pomena je način plačila. V svetu spletnih nakupov poznamo tipične načine plačila, kot sta:

- predhodno nakazilo na račun trgovine in
- plačilo po povzetju

V Sloveniji je zagotovo najbolj aktualno plačilo po povzetju, saj ima tako stranka največji občutek varnosti. Pri nakazilu na račun pa se ponavadi kupec lahko izogne visoki višini poštnine, saj je poštnina nižja v primeru, da ni odkupnine. Lahko pa seveda nakaže tudi poštnino in tako samo prevzame paket.

Manj pogosta pri nas pa sta plačevanje s sistemom PayPal ter direktno plačevanje s kreditnimi karticami, kot je 3D-Secure, ki ga pri nas vodi podjetje Bankart.

5.5.1 3D-Secure

Da je vprašanje varnosti nakupov bistven razlog, ki zavira višjo rast internetnega poslovanja, sta spoznala tudi vodilna kartična sistema Visa in MasterCard. Z namenom povečati varnost poslovanja ter hkrati še vedno ohraniti z vidika kupca enostaven postopek izvedbe postopka nakupa, sta uvedla sistem varnega poslovanja v spletu na osnovi rešitve imenovane 3D Secure. Rešitev je bila razvita z aktivnim sodelovanjem Vise pri razvoju, kasneje pa je to rešitev privzel še MasterCard. Visino komercialno poimenovanje rešitve je Verified by Visa (VbV), MasterCardovo pa Secure Code.

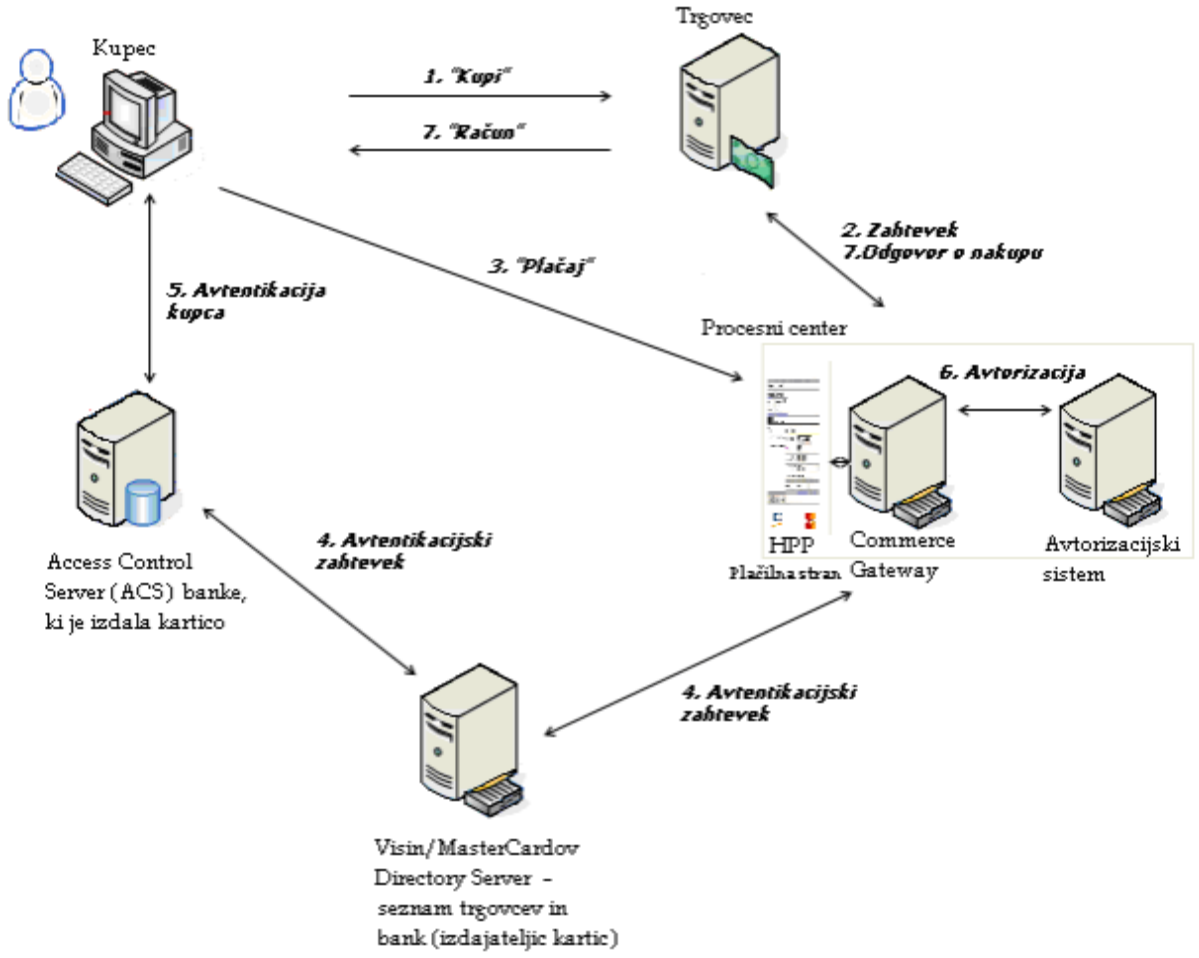
3D Secure je protokol, ki temelji na XML tehnologiji. Uporablja se ga kot dodatni varnostni sloj za kartične transakcije preko spleta. Razvilo ga je podjetje Visa, da bi izboljšali varnost nakupov preko spleta. Poimenovali so ga Verified by Visa. Kasneje je MasterCard razvil MasterCard

SecureCode, ki temelji na istem protokolu.

Avtentikacija je sestavljena iz treh članov, iz tega tudi izhaja ime 3-D (3 Domains).

- Kupec ter lastnik kartice,
- spletna trgovina ter
- banka.

Potek transakcije kaže sSlika 29.



Slika 29. Prikaz poteka tokov pri transakciji.

1. Kupec, imetnik plačilne kartice, si izbere artikle za nakup na spletnem prodajnem mestu. S klikom na gumb "Kupi" potrdi izbiro.

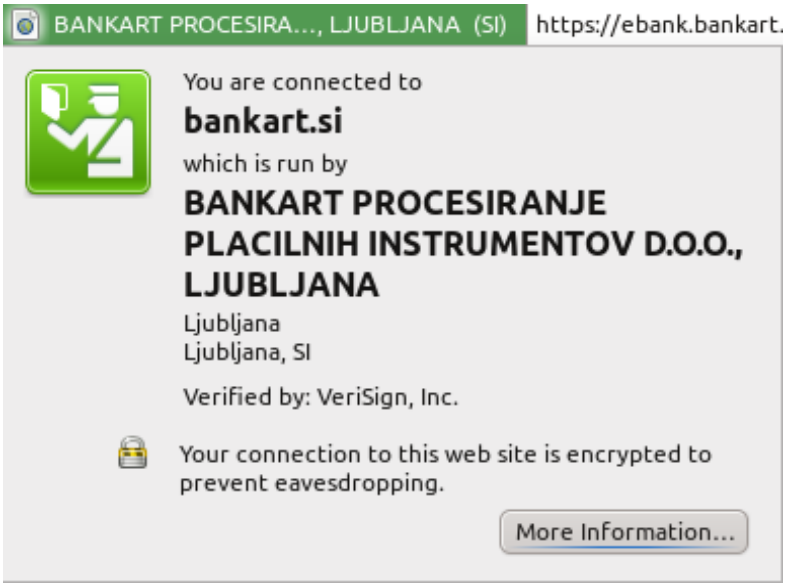
2. Spletna trgovina pošlje zahtevek procesnemu centru o nakupu.
3. Kupec je nato preusmerjen na plačilno stran (kupec se znajde na bankartovi spletni strani), kjer izpolni preprosti vnosni obrazec (številka kartice, potrditveno številko ter datum zapadlosti). Povezava je vzpostavljena preko SSL/TLS!
4. Bankart (Commerce Gateway) pošlje avtentikacijski zahtevek z zbranimi podatki na Viso oziroma MasterCard, kjer preverijo, ali sta trgovec in banka, ki je izdala kartico uporabljeno pri tem nakupu, registrirana kot 3D Secure.
5. Če je banka, ki je izdala uporabljeno kartico, registrirana kot 3D Secure, preusmerijo zahtevek na to banko. Sicer (če banka ni na seznamu vpisanih) se transakcija nadaljuje brez avtentikacije imetnika (brez točke 6.).

   S tem korakom je trgovec, ki je registriran kot 3D Secure trgovec, z vidika zlorab zaščiten po določilih MasterCarda in Vise. Torej tudi v primeru, če kupca ni možno preveriti.
6. Izdajatelj kartice interaktivno preveri kupca in o „rezultatu“ obvesti Viso oziroma MasterCard.
7. Visa oziroma MasterCard pošlje odgovor o avtentikaciji nazaj na procesni center, kjer se sedaj izvede avtorizacija (proces pridobivanja odobritve nakupa). Ob avtorizaciji se preveri datum veljavnosti, morebitne blokacije kartice, stanje na računu.
8. Procesni center pošlje odgovor o avtorizaciji trgovcu, ki ga le ta nato prikaže kupcu na zaključni strani (spletni račun, zaključno stran - nakup odobren/neodobren, podatki o nakupu, podatki o plačilu – datum, znesek…).

Vsa komunikacija med udeleženci poteka v kriptirani obliki in sicer z uporabo SSL/TLS. Ko spletna trgovina komunicira z Bankartom (Commerce Gateway), Bankart zahteva od spletne trgovine certifikat podpisan s takim overjaviteljem, ki mu Bankart zaupa.

Potrebno je vedeti tudi, da spletna trgovina ne more shraniti številko kartice ter potrditvene številke, saj kupec vnaša te podatke direktno na Bankartov strežnik. Ta korak pa je najbolj sumljiv za kupce, kajti preusmerjeni so na neko tretjo stran. Zato pa je vedno potrebno preveriti identiteto spletnega mesta, na katerem se trenutno nahajamo (Slika 30). Bakartovo stran bi lahko implementirali v iFrame, tako kupec ne bi imel občutka, da je zapustil trgovčevo stran. A slabost take implementacije pa je, da kupec ne more preveriti certifikata, kar je verjetno iz vidika tistih,

ki se vsaj malo spoznajo na varnost, slabše.



Slika 30. Pogled na Bankartov certifikat.

5.5.2 PayPal

PayPal omogoča plačevanje ter prenos denarja preko spleta. Prenos denarja nadomešča bankovce ter čeke. PayPal račun lahko povežemo direktno na banko ali pa na kreditno kartico.

Postopek plačevanja ter potek dogodkov je zelo podoben 3D-Secure sistemu.

- Uporabnik izbere plačilo s PayPalom
- na strežniku se nastavijo parametri, kot so, URL, če je plačilo izvedeno, URL, če ne preklicano, logotip ter podobne oblikovne stvari, ter seveda končni znesek naročila,
- strežnik preusmeri uporabnika na PayPalovo spletno stran,
- uporabnik se prijavi v PayPal, izbere tip plačila ter pritisne gumb „continue“. V tem koraku se še ne zajema denaraja.
- PayPal uporabnika preusmeri nazaj na spletno trgovino in sicer na pregled naročila – zadnji korak.

Ko uporabnik pregleda podatke klikne „Zaključi naročilo“ in je ponovno preusmerjen na PayPal.

Tokrat se mu izpišejo vsi naročeni artikli, količina ter cena. Na koncu pa skupen znesek, ki mu bo bil trgan. Ko klikne „Pay“, se zgodi prenos denarja na trgovčev PayPalov račun. Uporabnik pa je preusmerjen na spletno trgovino, kjer se mu izpiše zahvala o nakupu.

6 POSEBNOSTI PRI DELU S HTTP

6.1 Prepis URLjev

Rewrite engine je dodatek na strani strežnika, ki modificira ter prepisuje naslove (URLje). Rečemo jim tudi lepi ali kratki URLji ter URLji prijazni do iskalnikov (SEF – search engine friendly) [12].

Prednosti prepisanih URLjev so:

- so čistejši, prijaznejši; njihov pomen je razumljiv tako uporabnikom kot iskalnikom,
- skrijejo interno delovanje kode, v našem primeru PHPja,
- tudi po migraciji na druge tehnologije (drug programski jezik ali samo drugačna struktura funkcij), se URLji ohranijo.

Slabosti pa so:

- potrebno je dodatno delo, kajti za vsak URL je potrebno pravilo,
- ko uporabnik navigira po straneh s pomočjo AJAXa, se URL ne spreminja, kar onemogoča shranjevanje strani ter pošiljanja le-teh prijateljem.

Prepis deluje tako, da se med datotečnim sistemom, ki generirajo spletno stran ter URLji, ki so vidni zunanjemu svetu ustvari dodatni sloj.

Primer brez prepisa:
shop.php?group=item&item_id=149

ter s prepisom:
shop/zracno_orozje/zpuske/challenger-2000-149

Vidimo, da ima artikel ID 149, prav tako to vidimo v prepisanem primeru, saj je zadnja številka enaka ID artikla in to je 149.

Pravila za prepis je potrebno vpisati v .htaccess ali pa v konfiguracijsko datoteko strežnika Apache, ki velja za to spletno stran. Pravila so dejansko regularni izrazi, ki iz obiskovanega URLja poberejo določene dele in jih pošljejo v pravilni obliki naprej.

Primer pravila za prepis, ki ustreza zgornjemu prepisanemu URLju:
RewriteRule ^/shop/.*-([0-9]+)$ /shop.php?group=item&item_id=$1

Naš HTML mora kazati na „shop/zracno_orozje/zpuske/challenger-2000-149“. Ko uporabnik klikne na ta URL, ga Apache prestreže, pogleda pravila in ugotovi, da regularni izraz drži. Torej s tem regularnim izrazom pobere številko, ki se nahaja desno od zadnjega pomišljaja „-“ in jo pošlje /shop.php?group=item&item_id=149.

Isto velja tukaj:
shop.php?group=cat&cat_id=1

ter še prepisan URL
shop/zracno_orozje/zpuske

Pravilo:
RewriteRule ^/shop.*/(.*)$ /shop.php?group=cat&cat_url=$1

Razlika je le v tem, da ni nikjer nobenega IDja. Skrivnost je v tem, da se v tabeli kategorij nahaja poleg primarnega ključa še indeks, ki je unikatni in je tipa „varchar“. Torej namesto, da bi iskali kategorijo po njegovem primarnem ključu, ga iščemo po indeksu „cat_url“.

6.2 Globoke povezave in AJAX (angl. Deep linking)

Globoka povezava [9] na spletu je povezava, ki kaže na določeno vsebino ali sliko na določeni spletni strani, namesto, da bi kazala na domačo stran.

Na primer URL „http://www.google.com/about.html“ vsebuje vse informacije, da lahko pokaže

določeno stran, v tem primeru „about.html“, namesto da bi kazal na domačo stran „http://www.google.com“.

Tehnologija za HTTP (Hypertext Transfer Protocol) dejansko ne razlikuje med „globokimi“ povezavami ter drugimi. Vse povezave so enakovredne. Razlog za to je v osnovni ideji spleta, da lahko avtor poveže vsebino z vsebino, ki se nahaja kjer koli, tudi na drugih straneh. Tako je možnost „globokih“ povezav že vgrajena v HTTP ter URL. Stran seveda lahko poskusi preprečiti globoke povezave, vendar je za to potrebno dodatno delo. Ena možnost za preprečitev globokih povezav je preveritev prejšnjega URLja (Referring URL) z uporabo Rewrite engine-a.

Globalne povezave so obvezne za shranjevanje povezav (angl. Bookmarking). Prav tako omogočajo uporabo gumba „nazaj“, ki uporabnika vrne na prejšnjo stran.

6.3 AJAX

AJAX strani nalaga skripta, ki se izvaja na uporabnikovi strani v brskalniku. Zato nimajo posebnega URLja, ki bi ga videl uporabnik. Globoke povezave do vsebine pridobljene z AJAXom torej ni. Namen Ajaxa je namreč, da prepreči nalaganje celotne spletne strani ob spremembi vsebine.

Problem se pojavi pri kliku na gumb „nazaj“. Brskalnik ne bo naložil prejšnje strani ampak stran, ki jo je uporabnik obiskal pred trenutno (zadnja sprememba URLja). Razlog je v tem, da je gumb „nazaj“ narejen tako, da sledi spremembam URLja, v tem primeru pa je URL ostal enak. V veliko primerih to ni problem. Če gre za manjše stvari, kot so na primer preverjanje obrazcev. Problem se pojavi, ko je osnovna stran pridobljena z AJAXom. V takem primeru pa je potrebno implementirati globoke povezave v AJAX.

Za implementacijo je potrebno osnovno znanje HTMLja. Uporabimo povezave, ki se začnejo z „#“. Originalno se „#“ (angl. Hash) uporablja za povezave, ki kažejo na isto stran in sicer na določene odstavke oz. Sekcije. Na primer

Skoči na Opis

Zgornja povezava kaže na element z imenom „opis“. Primer

To je opis

Ko bo uporabnik kliknil na povezavo, ga bo brskalnik „zapeljal“ do odstavka, ki ima atribut „name“ enak „opis“.

Točno ta princip uporabimo za globoke povezave za AJAX strani, ker brskalnik ne osveži spletne strani, ko je povezava kliknjena.

Ko uporabnik obišče spletno stran, naredimo naslednje. Najprej pogledamo, če URL vsebuje znak „#“. Če jo vsebuje, potem naložimo potrebno spletno stran. Nato poslušamo, kdaj bo AJAX povezava kliknjena; ko se to zgodi, Ko jo uporabnik klikne, spremenimo URL tako, da mu pripnemo določeno vrednost ločeno z „#“.

Za primer vzemimo, da imamo na spletnem strežniku datoteko opis.html. Ko uporabnik odpre spletno stran (http://spletnastran.si), se naloži prevzeta stran. Ko klikne na povezavo „opis“, se z AJAXom pokliče „opis.html“ ter se vsebina te datoteke zamenja z prevzeto. Poleg tega pa se URL spremeni v http://spletnastran.si/#opis. Če uporabnik osveži spletno stran, skripta opazi znak „#“ v URLju, zato avtomatsko pokliče http://spletnastran.si/opis.html ter zamenja prevzeto vsebino.

Ko uporabnik obišče http://spletnastran.si/#opis, spletni strežnik dejansko vrne brskalniku vsebino spletne strani, ki se nahaja na http://spletnastran.si, nato se šele zgodi AJAX klic, ki vrne vsebino iz „opis.html“. Torej se zgodita dva zahtevka.

7 SKLEP

Spletna trgovina je zelo kompleksen sistem in vzame mnogo časa za programiranje. Največ časa nam poberejo ravno nezanimivi, trivialni deli. Ravno iz tega razloga obstajajo že odprtokodni sistemi, kot je na primer Joomla!. A problem se pojavi, ko želimo kakšno dodatno, specifično funkcionalnost, ki zahteva poseg v samo jedro sistema. Joomla! tudi podpira raznorazne dodatke, ki pa so največji vzrok za luknje in zlorabo. Posebno pozornost, ki je v diplomski nalogi nismo posebej omenjali, je potrebno nameniti napadu po metodi vrivanja v bazo (angl. MySQL injection). Pred izvedbo zahtevka je vedno potrebno preveriti ali je prišlo do posega.

Trenutna spletna trgovina je pripravljena, da preide v živo verzijo. To pa še ne pomeni, da bo ostala takšna kot je. V diplomski nalogi nismo omenjali same hitrosti izvajanja PHP programov. Dejstvo je, da je počasno, še bolj počasne pa so lahko bazne poizvedbe. Veliko nam pove že to, da Facebook uporablja več kot 30.000 strežnikov, ko naredimo izračun, ugotovimo, da en strežnik v eni uri postreže 100 uporabnikom. Torej brez predpomnjenja ne gre. Velik razlog za počasnost same kode je, da se ob vsakem zagonu interpretira v binarno kodo in nato požene. Razlog, da je temu tako, je verjetno marketinške narave, namreč Zend (podjetje, ki razvija PHP) ponuja lastni, seveda plačljivi, predpomnilniški sistem, ki interpretirano kodo drži v pomnilniku. Ena izmed alternativ je uporaba APC modula, ki dela ravno to, drži prevedeno kodo v pomnilniku in s tem omogoča hitrejše izvajanje. Pravi predpomnilnik pa je Memcached, ki drži skoraj vse vrste podatkov v pomnilniku (objekte, nize, številke). Če ima spletna stran veliko obiska, lahko zmanjšamo obremenjenost baze. Namesto, da gremo vsakič po podatke v bazo, jih raje zapišemo v pomnilnik za določen čas. Tako se podatki izpišejo iz baze samo prvemu uporabniku, naslednjim uporabnikom pa se, za določen čas, prikazujejo direktno iz pomnilnika. Od samih funkcionalnosti, pa je potrebno še implementirati primerjanje artiklov, sledenje prometa po uporabnikih, dinamično določanje popusta za določen artikel ter določenega uporabnika pod različnimi pogoji…

8 VIRI IN LITERATURA

[1] Verified by Visa System Overview External Version 1.0.2. Dostopno na: https://partnernetwork.visa.com/vpn/global/retrieve_document.do?documentRetrievalId=119

[2] What is openid. Dostopno na: http://openid.net/get-an-openid/what-is-openid/

[3] OpenID. Dostopno na: http://en.wikipedia.org/wiki/OpenID

[4] Entropy. Dostopno na: http://en.wikipedia.org/wiki/Entropy_(computing)

[5] Random. Dostopno na: http://en.wikipedia.org/wiki//dev/random

[6] Permissions – Facebook developers. Dostopno na: http://developers.facebook.com/docs/authentication/permissions/

[7] Efficient pagination using MySQL. Dostopno na: http://www.slideshare.net/suratbhati/efficient-pagination-using-mysql-6187107#

[8] Graph API. Dostopno na: http://developers.facebook.com/docs/reference/api/

[9] Deep linking. Dostopno na: http://en.wikipedia.org/wiki/Deep_linking

[10] OAuth. Dostopno na: http://oauth.net/

[11] Facebook Supports OpenID For Automatic Login: http://developers.facebook.com/blog/post/246/

[12] URL Rewriting Guide – Apache http Server. Dostopno na:

http://httpd.apache.org/docs/2.0/misc/rewriteguide.html