Web aplikacija za upravljanje inventarom

Lakić, Jasmina

SVEUČILIŠTE U ZAGREBU
FILOZOFSKI FAKULTET
ODSJEK ZA INFORMACIJSKE I KOMUNIKACIJSKE ZNANOSTI
SMJER ISTRAŽIVAČKA INFORMATIKA
Ak. god. 2018./ 2019.

Jasmina Lakić

Web aplikacija za upravljanje inventarom

Diplomski rad

Mentor: dr. sc. Vedran Juričić, doc.

Zagreb, rujan 2019.

Izjava o akademskoj čestitosti

Izjavljujem i svojim potpisom potvrđujem da je ovaj rad rezultat mog vlastitog rada koji se temelji na istraživanjima te objavljenoj i citiranoj literaturi. Izjavljujem da nijedan dio rada nije napisan na nedozvoljen način, odnosno da je prepisan iz necitiranog rada, te da nijedan dio rada ne krši bilo čija autorska prava. Također izjavljujem da nijedan dio rada nije korišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

_____

(potpis)

Sadržaj

1. Uvod

U diplomskom radu govori se o bazama podataka, njihovoj povijesti, utjecaju i jeziku SQL (engl. *Structured Query Languge*) koji služi za manipulaciju zapisima u bazi. Prikazani su i primjeri SQL naredbi, procedura i ključevi koji se koriste kao identifikatori, te njihov opis. Također, opisana je arhitektura baze podataka koja se sastoji od triju razina i svi načini kako se baza može testirati, te koje su prednosti pojedinih načina testiranja.

Govori se općenito o web aplikacijama, MVC (engl. *Model-View-Controller*) modelu te o razvoju web aplikacija korištenjem metode koja se oslanja na testiranja.

U radu je opisan i PHP (engl. PHP: *Hyperext Preprocessor*) programski jezik, s osvrtom na njegovu povijest, osnovnu sintaksu i tipove podataka. Također, predstavljen je i PEAR (engl. *PHP Extension and Application Repository*) repozitorij u sklopu PHP-a. HTML (engl. *Hypertext Markup Langugage*) jezik za označavanje prikazan je kroz povijest, opisana je njegova sintaksa te su navedeni primjeri iste. U sklopu HTML-a, opisan je protokol HTTP (engl. Hypertext Transfer Protocol).

U sklopu rada razvijena je web aplikacija koja je pisana programskim jezikom PHP, gdje se koristi i HTML za uređivanje web stranica, odnosno korisničkog sučelja. Kreirana je i baza podataka čije je zapise putem web aplikacije moguće uređivati, dodavati nove i brisati stare.

2. BAZA PODATAKA

Baza podataka je skup međusobno povezanih podataka koji su logički organizirani u jednu ili više različitih tablica. Informacije, odnosno zapisi, u bazi podataka imaju točno određeno mjesto u cjelokupnoj strukturi te, kako bi se u nju unijeli, moraju poštivati unaprijed određena pravila. Na primjer, ako je za jedan zapis određeno da mora biti u numeričkome obliku (1, 2, 3), ne može ga se unijeti kao znak (a, b, c). Tablice predstavljaju strukturu baze podataka i među njima se mogu stvarati veze. Takva struktura omogućuje da se sve ne zapisuje u jednu tablicu, koja bi nakon nekog vremena prestala biti upotrebljiva, pa bi se morali koristiti složeniji upiti kako bi se došlo do traženih zapisa, već se oni mogu rasporediti na više tablica.

Date u svojoj knjizi navodi radnje koje korisnici mogu izvoditi sa zapisima (2004):

- Dodavanje novih tablica u bazu.
- Umetanje zapisa u postojeću tablicu.
- Brisanje zapisa iz tablica.
- Uređivanje zapisa u tablicama.
- Brisanje tablica iz baze podataka.

Baze podataka postoje i djeluju bez obzira kako im se pristupa. Za pristupanje nije potrebno posebno računalo. Također, omogućuju da njima pristupa više korisnika istovremeno, što je ujedno i jedna od prednosti baza podataka.

2.1. Povijest baza podataka

Povijest baza podataka započinje šezdesetih godina prošlog stoljeća. U tom razdoblju, točnije šezdesetih i sedamdesetih godina, pojavile su se prve baze podataka. Bile su to navigacijske baze podataka. One su korisnicima omogućavale da putem pokazivača (engl. *pointer*) prelaze s jednog zapisa na drugi. Berg et. al (2012) u radu navode kako je naglasak bio na procesiranju podataka, a ne na strukturi same baze podataka, što znači da su korisnici morali znati strukturu same baze, kako bi mogli doći do željenih zapisa. Prvu navigacijsku bazu podataka zvanu IDS (engl. *Integrated Data Store*) kreirao je Charles William Bachman III (Foote, 2017).

Navigacijska baza podataka nije bila namijenjena svima, njezin nedostatak strukture onemogućavao je da se lako savlada njezino korištenje i dođe do željenih zapisa.

Računala su postajala sve prisutnija, a time i korištenje baza podataka, te se javila potreba za definiranjem smjernica i pravila koja će omogućavati lakše korištenje istih. Taj zadatak je uspješno obavio Bachman, okupivši grupu znanstvenika s kojima je osnovao *Database Task Group* koja je osmislila i standardizirala jezik COBOL (engl. *Common Business Oriented Language*). Jezik je predstavljen 1971. godine od kada je grupa postala poznata kao CODASYL (engl. *Conference/Committee on Data Systems Languages*). Kako bi se došlo do traženih zapisa koristio se jedan od tri načina:

1. Korištenje primarnoga ključa,
2. Korištenje *pointera* za pomicanje kroz zapise,
3. Skeniranje svih zapisa u bazi.

Prema Footeu, pojavom sustava jednostavnijih za korištenje, ovaj pristup brzo je izgubio na popularnosti, zbog svog neintuitivnog dizajna (2017).

Sedamdesetih godina dolazi do pojave prvih relacijskih baza podataka koje, za razliku od navigacijskih baza podataka, nisu zahtijevale poznavanje same strukture baze, već su se oslanjale na veze među zapisima. Edgar Frank Codd osmislio je model koji se oslanjao na tablice sa redovima, stupcima i shematskim prikazom odnosa među njima (Berg et al., 2012).

Dvije najznačajnije baze podataka koje su se oslanjale na njegov model pojavile su se između 1974. i 1977. godine.

- INGRES, koji je nastao 1974. godine na Sveučilištu Berkley u Kaliforniji te je koristio je vlastiti *query*-jezik zvan QUEL (National Research Council, 1999);
- SYSTEM R, koji je nastao je kao eksperimentalna relacijska baza podataka u tvrtki IBM u San Joseu te je također koristio svoj vlastiti *query*-jezik SQL (Chamberlin et al., 1981).

SQL od svojega prvog predstavljanja postaje standard u dohvaćanju zapisa iz relacijskih baza podataka i od tada pa sve do danas sve novonastale baze oslanjaju se na ovaj model.

Komercijalizacijom računala i njihovim korištenjem u poslovnom, kao i u istraživačkom sektoru, baze podataka pokazale su se kao odličan alat za koji je s vremenom, u svakome novome desetljeću, potrebno sve manje znanja za njegovo uspješno korištenje.

2.2. 12 Coddovih pravila

Kako bi se sa sigurnošću utvrdilo da je baza stvarno relacijska, Codd je izdvojio 12 pravila koja svaka relacijska baza mora definirati (Codd, 1985):

1. **Pravilo 0.** – svaka baza podataka za koju se tvrdi da je relacijska, mora moći upravljati zapisima.

   To znači da svaka relacijska baza podataka ima mogućnost manipuliranja zapisima, odnosno uređivanja, brisanja i dodavanja novih ili starih zapisa.

2. **Pravilo 1.** – sve informacije u relacijskoj bazi podataka predstavljaju se isključivo kao vrijednosti u tablici. To uključuje i imena stupaca i tablica.

   Ovo pravilo je potrebno je kako bi sve informacije bile opisane, što na kraju vodi do lakše nadogradnje baze ili različitih sustava koji će ju koristiti. Pravilo je potrebno poštivati kako bi administratori baze podatka mogli lakše i efektivnije održavati samu bazu i biti sigurno u njezinu strukturu i u ono što se u njoj nalazi.

3. **Pravilo 2.** – svim vrijednostima u relacijskoj bazi podataka mora se moći logički pristupiti specificiranjem imena tablice, vrijednostima koje se nalaze u stupcima te vrijednostima primarnog ključa, odnosno reda na kojem se nalazi.
4. **Pravilo 3.** – nulte vrijednosti (engl. *null values*) podržane su u relacijskim bazama podataka, kako bi predstavljale nedostatak informacija

   Potrebno je razlikovati *null* i vrijednost 0 (nula) u bazi podataka. *Null* govori da nema vrijednosti za zapis, dok nula predstavlja vrijednost, iako je to i dalje nula. Primjer za to jest:

Proizvod	**Količina**
Jabuka	4
Kruška	Null
Banana	0

Tablica 1.

Pogleda li se tablicu 1. sa proizvodima, vidljivo je da banana ima vrijednost nula, dok kod kruške imamo samo *null*, što znači da taj zapis nikada nije imao dodijeljenu vrijednost, već je ona prazna.

5. **Pravilo 4.** – struktura cijele baze podataka mora biti opisana u katalogu baze i mora joj se moći pristupiti korištenjem istih upita kao i za pristup samoj bazi.

   Katalog baze podataka je skup metapodataka o objektima koji definiraju bazu podataka kao što su tablice, pogledi, sinonimi, indeksi, korisnici (IBM, 2015)…

6. **Pravilo 5.** – relacijskoj bazi podataka može se pristupiti samo korištenjem jezika sa linearnom sintaksom. Ako se bazi može pristupiti bez korištenja jezika, to se smatra kršenjem ovog pravila i tada se ne smatra relacijskom bazom podatka.

   Jezik mora podržavati sve sljedeće stavke:

   - Definiciju podataka
   - Definiciju pogleda
   - Manipulaciju podataka
   - Autorizaciju
   - Podatkovni integritet
   - Pravila (ograničenja) transakcija

7. **Pravilo 6.** – svi pogledi koji se mogu ažurirati, također se moraju moći ažurirati kroz sustav (w3resources, 2019).
8. **Pravilo 7.** – svaki *query*-jezik koji se koristi u relacijskoj bazi podataka mora podržavati *insert*, *delete* i *update* na zapisima, odnosno dodavanje, brisanje i izmjenu postojećih ili novih zapisa. Također, ove radnje moraju imati mogućnost izvršavanja na većem broju tablica i redova (Tutorialcup, 2015).
9. **Pravilo 8.** – svi zapisi u bazi podataka moraju biti neovisni o promjenama koje se događaju oko njih. Odnosno, moraju ostati nepromijenjeni bez obzira na način na koji su zapisi i tablice pohranjeni.

10. **Pravilo 9.** – odnosi se na način na koji korisnici dolaze do zapisa. Ako se jednu tablicu odluči razdvojiti na više njih, one se korisniku moraju prikazati na isti način kao što su se prikazivale prije razdvajanja.
11. **Pravilo 10.** – baza podataka mora biti neovisna o aplikaciji putem koje se dolazi do zapisa. U prijevodu, sve promjene koje se odvijaju na aplikaciji ne smiju utjecati na rad i strukturu baze podataka.
12. **Pravilo 11.** – korisnici baze podataka ne smiju biti svjesni mogućnosti da je baza spremljena na različitom serveru od onoga koji one koriste. Vrijeme prikazivanja zapisa koje je korisnik zatražio ne smije biti ništa duže zbog udaljenosti (Tutorialcup, 2015).
13. **Pravilo 12.** – jezik baze podataka definira pravila integriteta koja se ni na koji način ne smiju moći zaobići upotrebom jezika niže razine.

Coddova pravila su i dalje temelj svake relacijske baze podataka i putem njih se one definiraju i prepoznaju. Iako su nastala početkom sedamdesetih godina prošlog stoljeća i dalje su to smjernice koje je potrebno pratiti prilikom izrade novih relacijskih baza.

2.3. Arhitektura baza podataka

Kada se govori o arhitekturi baze podataka, može se pozvati na ANSI-SPARC (engl. *American National Standards Institute, Standards Planning And Requirements Committee*) arhitekturu koja, iako nikada nije postala službeni standard, može poslužiti kao dobar primjer. ANSI-SPARC predlaže arhitekturu koja je podijeljena na tri razine: vanjsku (engl. *external*) razinu, konceptualnu (engl. *conceptual*) razinu i unutrašnju (engl. *internal*) razinu (Date, 2004).

2.3.1. Vanjska razina

Vanjska razina je ono što svaki korisnik može vidjeti od baze podataka. Pogled (engl. *view*) je ono što pojedini korisnik zapravo ima pravo vidjeti u bazi te se definira prema pravima koje ima (Date, 2004). Na primjer, korisnik koji radi u odjelu prodaje vidjet će samo one zapise koji su mu bitni za posao i neće biti svjestan da to nije sve što ta baza sadrži, odnosno, da drugi odjeli imaju različite poglede.

2.3.2. Konceptualna razina

Konceptualna razina opisuje sve podatke koji se nalaze u bazi podataka i načine na koji su međusobno povezani. Ona prikazuje podatke kako se oni nalaze u bazi, bez dodatnih prilagođenih pogleda. Razina se odnosi samo na opisivanje podataka i kao takva ne smije ukazivati na to kako su ti podaci fizički pohranjeni (Date, 2004).

2.3.3. Unutrašnja razina

Unutrašnja razina opisuje kako se podaci čuvaju u bazi podataka i na računalu, te na koji način je ona fizički prisutna na računalu(Date, 2004).

2.4. Testiranje baza podataka

Testiranje baza podataka je proces u izradi samih baza. Kako bi se utvrdilo da baza podataka i svi njezini slojevi rade ono za što su predviđeni, potrebno ih je testirati. Baza u kojoj jedna od UPDATE, INSERT ili DELETE naredbi ne bi radila, uzrokovala bi velike probleme i ne bi joj se moglo pristupiti neko vrijeme zbog dijagnoze i uklanjanja problema. Kako bi se to spriječilo, vrše se testiranja. Prilikom testiranja baza podataka koriste se *Black-box* testiranje i *White-box* testiranje.

1. ***Black-box* testiranje**

   Black-box testiranje izvodi se na grafičkom sučelju putem kojega se pristupa bazi podataka. U svom radu, Kaur i Sehra (2015) ukazuju na to da testeri ne moraju znati kod koji se nalazi u pozadini svega i način na koji je baza logički posložena, već samo moraju testirati pojedine funkcionalnosti i bilježiti sve greške koje se događaju. Testira se sučelje i njegova integracija s bazom, što uključuje provjeru nadolazećih podataka, mapiranje podataka i njihovih metapodataka te provjeru izlaznih podataka koji se dobiju kao odgovor na upit. Prema Kaur i Sehra (2015), ova vrsta testiranja omogućuje testiranje baze pri samome početku razvoja, što programerima omogućava da na samome početku znaju što ne radi, kako bi trebalo raditi te da pristupe rješavanju istoga. Testiranje od početka razvoja omogućava sprječavanje nekih problema koji bi se možda manifestirali daljnjim razvojem baze, a vezani su uz neku funkcionalnost koja se razvila na početku, što smanjuje vrijeme koje je utrošeno na pronalaženje problema.

2. ***White-box* testiranje**

   White-box testiranje malo je kompliciranije i zahtijeva testerovo poznavanje strukture baze podataka, *triggera*, pogleda, ali i poznavanje koda koji vrši radnje nad bazom. Tester mora znati čitati kod, kako bi mogao shvatiti koja se funkcija na što odnosi i gdje dolazi do problema. Prema Kaur i Sehra, testiranje se vrši nad unutarnjom strukturom baze podatak i odnosi se na (2015):

   - Testiranje pogleda i *triggera*
   - Provjeru sheme baze podataka, modela i tablica
   - Testiranje upita koji se izvršavaju nad bazom podataka
   - Provjeru pravila integriteta

White-box testiranje opširnije je i traje duže od *black-box* testiranja, zato što se provjerava funkcionalnost cijele baze, a ne pojedinih dijelova, te zahtijeva poznavanje koda. Prednost je što omogućava pronalaženje pogrešaka u kodu pa se programerima može specificirati što točno ne radi.

3. ***Grey-box* testiranje**

   Grey-box testiranje nalazi se između *White-box* i *Black-box* testiranja. Tester razumije što se mora dogoditi i shvaća kako je baza podataka organizirana, ali nema toliko znanja koliko ima pravi tester *White-box* aplikacija. Tester provjerava funkcionalnosti korisničkog sučelja, zatim pregledava bazu kako bi utvrdio ispravnost testa. Prednost ove vrste testiranja je da se testira sa stajališta krajnjega korisnika, a ne osobe koja ju je razvila ili osmislila (Bartlett, 2018).

Testiranje baze podataka izvodi se u tri koraka. U prvome koraku baza podataka se priprema za testiranje te se unose potrebni zapisi, pogledi i *triggeri*. U drugome se koraku pokreće test i pregledavaju se dobiveni rezultatu. U trećemu koraku, ako su rezultati zadovoljavajući, baza podataka se čisti i može se krenuti pripremati za drugi test, a ako nisu zadovoljavajući, test se pokreće ispočetka (Agile Data, 2006).

2.5. Utjecaj baza podataka

Baza podataka je središnji dio svakog informacijskog sustava. Bez nje, mnogo danas poznatih informacijskih sustava ne bi postojalo.

U poslovnome svijetu, prije pojave baza podataka, tvrtke su sve svoje poslovanje morale voditi na papiru, što je bio zahtjevan i dugotrajan proces. Kako bi se neki podatak izmijenio moralo ga se prvo pronaći među velikim brojem papira i registara, koji su možda bili organizirani, pa bi se vrijeme pronalaska smanjilo, ali i tada se u tome registru moralo pronaći zapis koji je bilo potrebno izmijeniti. Ogromna ušteda vremena, kao jedna od velikih prednosti, pomogla je bazama podataka da se raširе na skoro sve grane poslovanja i da se zadrže do današnjeg dana. Više nije potrebno pretraživati mnogo papira i trošiti vrijeme koje se moglo utrošiti drugdje, već se jednostavno u računalo upiše upit koji u sekundi korisniku vraća traženo. Vođenje cijelog poslovanja na papirima također je zahtijevalo poseban fizički prostor gdje se sve spremalo, čega danas sa bazama podataka nema. Korisnicima je samo potrebno računalo preko kojega će pristupiti bazi i pronaći ili izmijeniti potrebne zapise.

Baze podataka dostupne su korisnicima u bilo kojemu trenutku i može im se pristupiti kada god se poželi. Najbolji primjer za to je baza podataka u knjižnicama. Prije pojave baza podataka, kada se htjelo saznati je li neka knjiga dostupna za posudbu, moralo se otići do knjižnice, ili nazvati, kako bi se to provjerilo i pritom se moralo paziti na radno vrijeme knjižnice. Danas su korisnicima, putem web aplikacije dostupne u bilo kojem trenutku i jednim upitom vraćaju informacije o traženoj knjizi, je li posuđena, postoji li kopija za rad u čitaonici, može li se posuditi u nekoj drugoj knjižnici te čak i fizički opis same knjige, kao što je broj stranica i vrsta uveza. Više nema vremenskog ograničenja kako bi se došlo do željenih informacija.

Razne trgovine i skladišta više ne moraju fizički provjeravati i brojati svaki komad opreme koji imaju. Baza podataka vodi brigu o tome i upitom se na lak, bezbolan i brz način dolazi do tražene informacije. Ako se sve sustavno prati i bilježi u bazu podataka, ona korisnicima može dati razne statističke podatke za duži ili kraći vremenski period, što dovodi do lakšeg planiranja budućeg poslovanja i uvida u to što se može izmijeniti ili poboljšati.

Obrati li se pažnja na društvene mreže, uočljivo je da su one zapravo vrlo velike baze podataka. One na svojem sučelju korisnicima prikazuju zapise o njima ili drugima, koji

su preuzeti iz više tablica. Korisnikovo ime, prezime, *e-mail* adresa, broj telefona, adresa prebivališta, podaci o školovanju i mjesta koja je posjetili su informacije koje su spremljene na raznim mjestima u bazi podataka. Prilikom prijave na društvenu mrežu, web aplikacija u djeliću sekunde iz baze podataka povlači podatke koji se zatim prikazuju korisniku.

Baze podataka su na poslovnoj razini donijele mnoge promjene koje su na pozitivan način izmijenile cijelo poslovanje. Nisu vremenski ni prostorno ograničene te veći broj korisnika može istovremeno pristupiti zapisima, čime su se ubrzali mnogi poslovni procesi. Pristup informacijama je trenutan što pridonosi bržem i efikasnijem tijeku unutarnjih procesa.

3. SQL

SQL (engl. *Structured Query Language*) je jezik koji se koristi u relacijskim bazama podataka kako bi se tražili, brisali, ažurirali i dodavali zapisi u bazu. Razvili su ga Donald D. Chamberlin i Raymond F. Boyce iz IBM-a (Computer History Museum, 2016). SQL je, kao jezika za rad s bazama podataka, iznimno koristan kad su u pitanju međusobno povezani zapisi.

Godine 1986. postao je standard Američkog nacionalnog instituta za standarde (engl. *ANSI - American National Standard Institute*), a sljedeće godine i standard Međunarodne organizacije za standarde (engl. *ISO – International Organization for Standardization*) (Foote, 2017).

3.1. SQL operatori

SQL kao jezik za pristup i rad s bazama podataka ima svoju sintaksu koja se koristi operaterima. U tablici 2. prikazane su aritmetičke operacije, a u tablici 3. operatori za uspoređivanje koji se koriste u SQL-u.

Aritmetičke operacije

+ (predznak)	Čini broj pozitivnim
- (predznak)	Čini broj negativnim
/	dijeljenje
*	množenje
+	zbrajanje
-	oduzimanje

Tablica 2. Aritmetičke operacije

Operatori za uspoređivanje

=	jednako
!=, ^=, <>	nejednakost
>	veće od
<	manje od
>=	veće ili jednako od
<=	manje ili jednako od
IN	nalazi li se vrijednost unutar skupa vrijednosti
NOT IN	ne nalazi li se vrijednost unutar skupa vrijednosti
NOT BETWEEN…AND…	provjerava ne nalazi li se vrijednost unutar opsega vrijednosti
IS NULL	provjerava je li vrijednost NULL
IS NOT NULL	provjerava nije li vrijednost nula

Tablica 3. Operatori za uspoređivanje

3.2. DML (engl. *Data Manipulation Language*)

Data Manipulation Languge (DML) je jezik kojim baza podataka dodaje, briše ili uređuje zapise koji se u njoj nalaze.

INSERT – naredba kojom se dodaju zapisi u bazu podataka.

UPDATE – naredba kojom se uređuje postojeći zapis u bazi podataka.

DELETE – naredba kojom se briše postojeći zapis iz baze podataka.

Primjer 1. INSERT

```
INSERT INTO Korisnik (ime, prezime) VALUES ('Ana', 'Anic')
```

Programski kod 1. Primjer INSERT naredbe

Naredba vidljiva u programskom kodu 1 dodaje u kolone ime i prezime vrijednosti Ana Anic i te kolone se nalaze u tablici „Korisnik“.

Primjer 2. UPDATE

```
UPDATE Korisnik SET ime = 'Ana', prezime = 'Anić' WHERE
korisnikID = 1
```

Programski kod 2. Primjer UPDATE naredbe

Naredba vidljiva u programskom kodu 2 će korisniku s *ID*-em 1 ažurirati ime i prezime. To znači da više neće biti „Ana Anic“, nego „Ana Anić“. *KorisnikID* je potreban kako bi se sa sigurnošću utvrdilo da se radi o tom zapisu koji se želi izmijeniti, zato što se u bazi podataka može nalaziti više korisnika koji se zovu Ana Anić.

Primjer 3. DELETE

```
DELETE FROM Korisnik WHERE korisnikID = 1
```

Programski kod 3. Primjer DELETE naredbe

Naredba iz programskog koda 3 će izbrisati korisnika koji ima ID 1 i koji se nalazi u tablici „Korisnik“, odnosno, izbrisati će Anu Anić.

3.3. DDL (engl. *Data Definition Language*)

DDL ili *Data Manipulation Language* je jezik koji u bazi podataka služi kako bi se kreiralo, uredilo ili obrisalo objekte ili zapise. To su najčešće tablice, indeksi ili korisnici. Naredbe u ovom jeziku su: CREATE, ALTER i DELETE.

CREATE – naredba kojom se u bazu podataka može umetnuti nova tablica ili indeks vidljiva je u programskom kodu 4.

ALTER – naredba pomoću koje se uređuje već postojeći objekt u bazi podataka vidljiva je u programskom kodu 5.

DROP – naredba koja uklanja cijelu tablicu iz baze podataka vidljiva je u programskom kodu 6.

Primjer 1. – CREATE

```
CREATE TABLE (
korisnikID      INTEGER         PRIMARY KEY,
ime             VARCHAR(75)     not null,
prezime         VARCHAR(75)     not null,
datumrodenja    DATE            not null
);
```

Programski kod 4. Primjer CREATE naredbe

Ovom naredbom kreira se nova tablica u bazi podataka. Tablica će imati definirane kolone: „korisnikID, ime, prezime i datumrodenja“, zajedno s postavljenim pravilom koji tipovi podataka su određeni za svaku kolonu. Pogleda li se prvu kolonu „korisnikID“ vidljivo je da je za nju određeno da taj zapis može kao vrijednosti imati samo cijele brojeve.

Primjer 2. ALTER

```
ALTER TABLE Korisnik COMMENT = 'Komentar na tablicu korisnik'
```

Programski kod 5. Primjer ALTER naredbe

Tablica „Korisnik“ se izmijenila tako da joj se dodao komentar. Sa ALTER TABLE omogućeno je dodavati i brisati kolone, indekse, mijenjati tip podataka za postojeće kolone i mijenjati imena kolona i tablica (MySQL, 2019).

Primjer 3. DROP

```
DROP TABLE Korisnik;
```

Programski kod 6. Primjer DROP naredbe

Potrebno je razlikovati ALTER i DELETE naredbe u SQL-u. Isprva se može činiti kako obje naredbe rade istu stvar, ali u praksi to nije tako. Naredba DELETE obrisat će zapise koji se nalazi u određenoj tablici, dok će naredba ALTER ukloniti cijelu tablicu iz baze podataka. Nakon DELETE naredbe, ciljana tablica i dalje ostaje u bazi podataka, samo što više ne sadrži nikakve podatke.

3.4. DQL (engl. *Data Query Languge*)

DQL (engl. *Data Query Language*) je jezik koji služi za dohvaćanje jednog ili više zapisa iz baze podataka. Najznačajnija i najčešće korištena naredba ovog jezika je naredba SELECT, čija sintaksa može imati više razina koje su vidljive u programskom kodu 7. Koliko razina će se staviti, ovisi o tome koliko duboko se želi pretraživati bazu podataka i sa kojim uvjetima.

```
SELECT FROM
WHERE
GROUP BY
HAVING
ORDER BY
```

Programski kod 7. Razine SELECT naredbe

ALL ili '*' koristi se kada se želi da naredba vrati sve redove i stupce iz tražene tablice.

```
SELECT * FROM Korisnik
```

Programski kod 8. Primjer FROM naredbe

Naredba vidljiva u programskom kodu 8 vraća sve zapise iz tablice pod imenom „Korisnik“. FROM se koristi kako bi se ukazalo da se dohvaćaju svi zapisi iz određene tablice.

WHERE se koristi kada je potrebno prikazati sve zapise koji zadovoljavaju određeni uvjet, odnosno da ima tražene vrijednosti koje se specificira.

```
SELECT * FROM Korisnik WHERE korisnikID = 1
```

Programski kod 9. Uvjet WHERE

Upitom vidljivim u programskom kodu 9 prikazuju se samo korisnici koji imaju korisnikID jednak 1. WHERE se može proširiti sa uvjetima AND, OR i NOT.

GROUP BY se koristi kada se rezultate iz tablice želi grupirati po određenoj koloni. Zapisi se prilikom dodavanja u tablicu ne grupiraju, već se samo upisuju nakon zadnjeg dodanog zapisa i uglavnom im se dodjeljuje skriveni jedinstveni identifikator, koji se ne prikazuje u rezultatima. Kako bi se grupiralo korisnike prema vlastitom kriteriju, koristi se GROUP BY, čiji je primjer vidljiv u programskom kodu 10.

```
SELECT * FROM Korisnik GROUP BY ime
```

Programski kod 10. GROUP BY primjer

Ovaj upit vraća sve zapise iz tablice „Korisnik“ i grupira ih prema imenu.

HAVING se koristi kada se iz rezultata koji su dobiveni putem GROUP BY, žele prikazati samo oni rezultati koji zadovoljavaju još jedan uvjet.

```
SELECT * FROM Korisnik GROUP BY ime HAVING ime >= '3'
```

Programski kod 11. GROUP BY i HAVING primjer

Upitom vidljivim u programskom kodu 11 dobiju se grupirana imena korisnika, ali prikazana su samo ona imena koja imaju tri ili više znakova, što se uvjetovalo operaterom >= (veće ili jednako).

ORDER BY naredbom, koja je prikazana u programskom kodu 12, vraćeni su zapisi koji su poredani prema vrijednosti jedne ili više kolona. ORDER BY je jedina naredba pomoću koje je moguće poredati zapise, bez nje se rezultati prikazuju onim redoslijedom kojim su dodani u samoj bazi podataka. ORDER BY je iznimno korisna naredba ako u bazi podataka postoji više zapisa od kojih svaki posjeduje vlastite numeričke ili alfanumeričke vrijednosti. Tako se, na primjer, može zaposlenike poredati po datumu kada su zaposleni, prema visini plaće, godinama staža, dobi, datumu rođenja, odjelu u kojemu su zaposleni, njihovoj tituli, itd.

```
SELECT * FROM Korisnik ORDER BY prezime, ime
```

Programski kod 12, ORDER BY primjer

Ovaj upit vraća sve zapise iz tablica „Korisnik“ te su oni poredani prema prezimenu, a zatim i imenu. Postoji li tako u bazi podataka više korisnika koji se prezivaju Horvat, ali sa različito zovu, oni su na kraju poredani prvo prema prezimenu, a zatim i imenu te prednost ima onaj korisnik čije se početno slovo imena nalazi prije u abecedi.

SELECT je najmoćniji upit koji korisnici imaju na raspolaganju pri pregledavanju zapisa u bazi podataka. Može mu se dati više uvjeta odjednom i odrediti kakav se poredak želi. Uvjeti se mogu kombinirati, ali i ne moraju. Ako se SELECT izvršava na većem broju zapisa preporuča se korištenje više uvjeta, kako bi se što više smanjio obujam zapisa koje može vratiti. Stavi li se, na primjer, da se prikažu samo imena iz tablice „Korisnik“, dobivaju se svi zapisi u njoj. Ali, ako se stavi uvjet da ime ne smije biti duže od pet znakova, sveukupni broj zapisa postaje znatno manji.

3.5. DCL (engl. *Data Control Language*)

DCL je jezik pomoću kojega je moguće kontrolirati kolika prava određeni korisnik ima nad bazom podataka. Korisniku se mogu dati prava da pregledava i uređuje bazu podataka i zapise koji se u njoj nalaze ili mu se samo može dodijeliti pravo da pregledava bazu. Ova prava se u svakome trenutku mogu korisniku oduzeti sa naredbom koja također spada u DCL. Prava mogu dati ili oduzeti samo administratori i/ili vlasnici baze podataka. DCL se koristi kako bi se očuvala sigurnost baze podataka ako njoj pristupa veći broj korisnika.

GRANT – naredbom se korisniku pridaju prava (pregledavanje, uređivanje, dodavanje, brisanje).

REVOKE – naredbom se korisniku oduzimaju prava koja su mu dana .

Primjer 1. GRANT

```
GRANT povlastice ON ime tablice TO korisnik
```

Programski kod 13. GRANT primjer

Naredbom u programskom kodu 13, određenom korisniku dodjeljuju se povlastice koje može izvršavati nad definiranom tablicom u bazi podataka.

Povlastice koje je moguće dodijeliti su:

- SELECT – pravo na pregledavanje zapisa u tablici
- INSERT – pravo na dodavanje zapisa u tablicu
- UPDATE – pravo na uređivanje postojećih zapisa u tablici
- DELETE – pravo na brisanje zapisa iz tablice
- REFERENCES – pravo da se postave ograničenja na tablici
- ALTER – pravo da se koristi ALTER TABLE naredba
- ALL – pravo da se izvode SELECT, INSERT, UPDATE, DELETE i REFERENCES naredbe

Primjer 2. REVOKE

```
REVOKE povlastice ON ime tablice FROM korisnik
```

Programski kod 14. REVOKE primjer

Naredbom REVOKE, koja je vidljiva u programskom kodu 14, korisniku se oduzimaju povlastice koje je imao. Povlastice su jednake kao i pri dodjeljivanju prava (*select, insert, update, delete, references, alter, all*).

3.6. TCL (engl. *Transaction Control Language*)

TCL se koristi kako bi se uspješno kontrolirale transakcije u bazi podataka. Primjer koji se uvijek navodi kod TCL-a su baze podataka koje se bave novčanim transakcijama. Kad jedna osoba šalje novac drugoj osobi, prvo se „osobi A“ mora oduzeti iznos koji se zatim daje „osobi B“. Kako bi transakcija bila uspješna i cijeli taj proces prošao bezbolno, on se ne smije prekidati drugim naredbama i mora se izvršiti kao jedna cjelina. Oba koraka u procesu moraju se izvršiti, što znači da ako je prvi korak uspješan, drugi također mora biti uspješan. Ne smije se dogoditi da se prvi korak izvrši, a drugi ne izvrši.

Svaki sustav koji se bavi transakcijama mora zadovoljavati ACID smjernice. Akronim ACID stoji za *Atomicity, Consistency, Isolation and Durability* i definirali su ga Andreas Reuter i Theo Härder 1983. godine u svom radu Principles of Transaction-Oriented Database Recovery (1983).

Atomarnost (engl. *Atomicity*)

Transakcija se ne smije izvoditi pojedinačno, odnosno svi njezini dijelovi moraju funkcionirati kao jedna cjelina. Kada se transakcija pokrene nad bazom, ona mora biti izvršena u cijelosti. To nadalje znači da se ne može izvršiti samo jedan dio transakcije, naime, ako jedan njezin dio ne uspije, cijela se transakcija mora prekinuti bez posljedica. Atomarnost osigurava da sve transakcije budu u potpunosti uspješne ili u potpunosti neuspješne (Kreibich, 2010).

Konzistentnost (engl. *Consistency*)

Transakcija koja se izvršava nad bazom podataka ne smije ugrožavati integritet same baze podataka. U jednom trenutku prilikom transakcije, zapisi u bazi mogu izgubiti svoj integritet, ali, ako se transakcija uspješno izvrši do kraja, integritet bi se trebao vratiti u prvotno stanje. Ovo se najčešće događa prilikom prebacivanja novčanih sredstava sa jednog računa na drugi (Kreibich, 2010).

Izolacija (engl. *Isolation*)

Kada se pokrene transakcija nad bazom podataka u njezino izvršavanje se ne smiju uplitati drugi upiti. Sve ostale promjene koje su pokrenute prilikom izvršavanja transakcije moraju se odraditi nakon što je transakcija uspješno ili neuspješno završila. Izolacija je potrebna kako bi se sačuvala atomarnost i konzistentnost transakcije.

Izdržljivost (engl. *Durability*)

Izdržljivost se odnosi na izvršavanje transakcije i njezin utjecaj na bazu podataka. Nakon što je transakcija uspješno izvršena, zabilježena je u bazi i ne može se vratiti u prvotno stanje. Nakon ovog trenutka, što god da se pod utjecajem vanjskih faktora dogodilo sa sustavom ili bazom podataka, promjene moraju ostati zabilježene. Ako se nešto dogodi prije nego što se transakcija izvršila, njezine promjene ne smiju biti vidljive u bazi podataka (Kreibich, 2010).

Naredbe koje se koristi u TCL-u su COMMIT i ROLLBACK

COMMIT – naredbom se potvrđuje da su sve promjene koje se želi izvršiti permanentne. Primjer je vidljiv u programskom kodu 15.

ROLLBACK – naredbom se odustaje od svih promjena koje su se trebale dogoditi i vraća bazu u prvotno stanje, odnosno stanje prije početka transakcije. Primjer je vidljiv u programskom kodu 16.

Primjer 1. COMMIT TRANSACTIOIN

```
BEGIN TRANSACTION

SELECT * FROM Korisnik

DELETE FROM Korisnik

INSERT INTO Korisnik (ime, prezime) ('Bruna', 'Brunic')

SELECT * FROM Korisnik

COMMIT TRANSACTION
```

Programski kod 15. Primjer COMMIT

Izvršavanjem COMMIT-a sve promjene ostaju zabilježene u bazi podataka.

Primjer 2. ROLLBACK TRANSACTION

```
BEGIN TRANSACTION

SELECT * FROM Korisnik

DELETE FROM Korisnik

INSERT INTO Korisnik (ime, prezime) ('Bruna', 'Brunic')

SELECT * FROM Korisnik

ROLLBACK TRANSACTION
```

Programski kod 16. Primjer ROLLBACK

ROLLBACK će sve promjene koje su se zatražile povući i te će vratiti bazu u stanje u kojem je bila prije pokretanja transakcije.

3.7. Ključevi u SQL-u

Primarni ključ (engl. *Primary key*)

U bazi podataka primarni ključ označava da su vrijednosti koje se nalaze u koloni na koju je dodijeljen jedinstvene i da se ne smiju ponavljati. Primarni ključ može biti samo jedan u tablici i ne smije sadržavati NULL vrijednost.

Primarni ključ se primarno dodjeljuje koloni za koju se zna da je jedinstvena i koja se prikazuje u drugim tablicama. Na primjer, ako se u tablici korisnika nalaze *korisnik_id, ime, prezime, korisničko ime, lozinka i ema*il, primarni ključ se nalazi na *korisnik_id* i tablica se kreira na način vidljiv u programskom kodu 17:

```
CREATE TABLE Korisnici(
korisnik_id INT AUTO_INCREMENT PRIMARY KEY,
ime VARCHAR(255),
prezime VARCHAR(255),
korisnicko_ime VARCHAR(40),
lozinka VARCHAR(255),
email VARCHAR(255)
);
```

Programski kod 17. Dodjeljivanje primarnog ključa

Jedinstveni ključ (engl. *Unique key*)

Jedinstveni ključ, kao i primarni ključ, označava jedinstvene vrijednosti u tablici. Za razliku od primarnog ključa, jedinstvenih ključeva može biti više u tablici te on može imati jednu NULL vrijednost, ali se isto tako vrijednosti ne smiju ponavljati. U postojećoj tablici se mogu dodati jedinstveni ključevi za kolone koje također imaju jedinstvene vrijednosti. U ovom slučaju to bi bile *korisničko ime* i *email* što je vidljivo u programskom kodu 18.

```
ALTER TABLE Korisnici

ADD UNIQUE ('korisnicko_ime', 'email');
```

Programski kod 18. Dodjeljivanje jedinstvenog ključa

3.8. Prednosti i nedostatci SQL-a

SQL, kao primarni jezik za rad s bazama podataka, ima svoje prednosti i nedostatke. Najveća prednost mu je ta što za njegovo uspješno korištenje nije potrebno znanje programskih jezika. Za izradu tablica postoji grafičko sučelje za koje je možda u početku potrebna prilagodba, ali se brzo savladava, a za pretraživanje baze podataka koriste se upiti. Upite se može pisati na osnovnoj, početničkoj, razini koja sadrži jednu naredbu, ili na naprednijoj razini gdje se može kombinirati više naredbi odjednom i tako suziti rezultate koji bi se vratili. Pisanje upita ovisi o veličini baze podataka i koliko se zapisa u njoj nalazi. Nalazi li se u bazi podataka više stotina zapisa, ne koristi se samo SELECT naredba, već se ona kombinira s GROUP BY ili WHERE naredbama, kako bi se točnije odredilo ono što se zapravo želi izvući iz baze.

Korisnicima baze podataka u svakom trenutku mogu se dodjeljivati ili oduzimati prava, što pridonosi dodatnoj sigurnosti. Može im se dati pravo da samo pregledavaju zapise u bazi, ili im omogućiti da uređuju, dodaju i brišu zapise iz baze. Ovakva slojevitost i raspodjela prava može, ako se nešto dogodi sa bazom zbog radnji koje su se odvijale nad njom u tom trenutku, suziti broj potencijalnih korisnika koji su to uzrokovali. Naravno, postoji i lakši način na koji se može doći do odgovornog korisnika, a to je preko *logova*, no uvijek se moramo voditi pretpostavkom da nisu svi korisnici dovoljno informatički pismeni.

SQL baze podataka su pod standardom ANSI i ISO i kao takve moraju zadovoljavati određene uvjete koje oni definiraju.

SQL je prenosiv i podržava osobna računala, laptope, tablete, servere i mainframove. Moguće ga je pokrenuti sa svakog uređaja koji ima pristup internetu.

Primarni nedostatak SQL–a je izgled njegovog sučelja. Isprva nije jasno gdje se koje opcije nalaze, a da bi se došlo do nekih opcija, potrebno je prolaziti kroz više izbornika.

4. Web aplikacije

Web aplikacije su kompjuterski programi bazirani na klijent–server odnosu, klijent je svako računalo koje je spojeno na mrežu, a server sadrži sve informacije koje klijent od njega traži. Pokreću se u raznim internetskim preglednicima. Svaka web aplikacija ima svoj prednji (engl. *frontend*) i pozadinski (engl. *backend*) dio. *Frontend* je ono što korisnik vidi kada pristupi aplikaciji, odnosno grafičko sučelje, a *backend* je ono što se vrti u pozadini aplikacije i to korisnici ne vide.

U svojim počecima, klijent – server model je dio svojih procesa dijelio sa klijentom. Što je značilo da je svaki korisnik na svojem osobnom računalu morao instalirati klijentski program, koji je prikazivao grafičko sučelje web aplikacije (Britannica, 2009).

Godine 1995. Netscape je predstavio jezik JavaScript koji je programerima omogućavao dodavanje dinamičkih elemenata grafičkom sučelju, koje je do tada bilo statičko (Netscape, 1995). Godinu dana kasnije Macromedia predstavlja Flash, *plug-in* koji se mogao dodati *browseru* kako bi mogao prikazivati vektorske animacije (FundingUniverse, 2006). Godine 2014. službeno dolazi HTLM5, koji zbog svojih grafičkih i multimedijskih mogućnosti više nije od korisnika zahtijevao instaliranje dodatnih programa kako bi internet-preglednik mogao prikazivati animacije i dinamičke elemente (W3C, 2014).

Izrada web aplikacija vrši se putem *web-frameworka* koji su dizajnirani kako bi podržali sam razvoj aplikacija. Većina njih bazirana je na *model–view–controller* (MVC) uzorku koji aplikaciju dijeli na tri logičke komponentne: *model, view i controller*.

4.1. Arhitektura softvera

Prije izrade svakog softvera potrebno je definirati njegovu strukturu. To se, u sklopu razvijanja aplikacija naziva arhitekturom softvera. Ona definiran sve elemente aplikacija, njihove zadatke, svojstva i gdje se nalaze u odnosu na ostale elemente unutar aplikacije te opisuje veze među raznim elementima. Arhitekturu je potrebno definirati kada se dizajnira softver za velike sustave kao što su Google ili SAP, ali se može definirati i za manje sustave, iako nije najpotrebnija (Software Engineering Institute, 2017).

Arhitektura softvera se definira kako bi se odmah na početku utvrdilo od čega se mora krenuti i koji dijelovi softvera mogu uslijediti. Postavljaju se prioriteti kako se prilikom

samog razvoja softvera ne bi od početka koncentriralo na pogrešne dijelove i time ugrozio cijeli tijek razvoja.

Novoseltseva (2017) u svojemu radu ukazuje kako definiranje arhitekture softvera ima nekoliko prednosti:

- Može se predvidjeti ponašanje softvera i hoće li njegove mogućnosti zadovoljavati specifikaciju. Ovime se smanjuje sam trošak razvoja i mogućnost pogreške. Ako se softver krene razvijati prije definiranja arhitekture i provjere zadovoljava li uopće specifikaciju, može se dogoditi da se neka osnovna funkcionalnost ponovno razvije. Time se gubi na vremenu te se povećava trošak.
- Opisivanje različitih elemenata omogućuje njihovo ponovno korištenje u novim sustavima koji će možda imati slične mogućnosti. Ponovnim korištenjem smanjuje se mogućnost pogrešaka u novim sustavima, zato što se već koristio u sustavu koji ga uspješno koristi i koji je prošao sve testove. Također se smanjuje trošak na novome sustavu zbog ponovnog iskorištavanja (Perry i Wolf, 1992).
- Postavljanjem prioriteta i njihovim uspješnim definiranjem umanjuje se mogućnost povećanja troškova i pomaže se da se zadovolje postavljeni rokovi.
- Definiranje arhitekture i njezino predstavljanje klijentu omogućuje lakšu komunikaciju te izricanje želja i ciljeva sustava. Klijent predstavlja svoja očekivanja i arhitektura se može izmijeniti prije početka razvijanja.
- Smanjuje se mogućnost pogreške.
- Sve gore navedene točke omogućuju da se spriječe dodatni troškovi.

Definiranje arhitekture vrši se u nekoliko koraka koji se odvijaju u različitim fazama razvoja (Wikipedia Contributors, 2019):

1. **Analiza arhitekture**

   Korak u kojemu se pregledava u kakvome će okruženju sustav raditi te se definiraju zahtjevi klijenta:

   - što će sustav raditi, odnosno, koje će mogućnosti imati
   - hoće li sustav biti siguran, pouzdan i operabilan te hoće li učinkovito obavljati svoje zadatke
   - hoće li se sustav moći održavati i tko će to raditi
   - kako će se sustav nositi s tehnološkim promjenama

   Ove četiri točke imaju najveći utjecaj na definiranje same arhitekture.

2. **Dizajn arhitekture**

   Nakon što se s klijentom definiraju svi zahtjevi kreće se na dizajn, odnosno izradu arhitekture. Uzimaju se u obzir klijentove želje i očekivanja te se prema njima dizajnira arhitektura.

3. **Evaluacija arhitekture**

   Nakon što se krene u izradu dizajna, prelazi se na korak evaluacije. Provjerava se zadovoljava li trenutni dizajn klijentove želje koje su definirane u analizi arhitekture. Evaluacija se može raditi u kojemu god koraku prilikom dizajniranja, nakon što se dizajnirao dio arhitekture ili nakon završetka, odluka je na arhitektu. Važnost ovog koraka predstavlja i činjenica da postoje posebni softveri za evaluaciju arhitekture, među najznačajnijima su ATAM (engl. *Architecture Tradeoff Analysis Method)* i TARA.

4. **Razvoj arhitekture**

   Klijent u kojemu god trenutku može zatražiti da se dodaju nove mogućnosti ili promijene stare, to zahtijeva da se arhitektura izmijeni. Nove mogućnosti se moraju dodati i pritom ne smiju utjecati na druge mogućnosti (osim ako je zahtijevano drugačije) te se moraju uklopiti u postojeću arhitekturu.

4.2. Uzorci u arhitekturi

Različiti softveri mogu imati sličnu arhitekturu, što znači da nailaze na iste probleme prilikom razvoja. Kako bi se ti problemi uklonili koriste se uzorci koji ih prepoznaju i predlažu rješenja.

4.2.1. Blackboard

Blackboard-uzorak koristi se u razvijanju softvera kako bi se koordiniralo više različitih sustava koji moraju raditi zajedno, pritom se na prvo mjesto stavljaju izvori znanja. Uzorak se sastoji od varijabli kojima se može pristupiti putem više procesa. Kontroler pregledava ploču i određuje koji su izvori znanja prioritet (Microsoft, 2012).

Sastoji se od tri dijela: ploče koja sadrži objekte, izvora znanja koji sadrži module i kontrolera koji odabire, konfigurira i pokreće iste.

Spada u grupu uzoraka koji identificiraju metode koje programi koriste za komunikaciju. Fokusiraju se na odnose među objektima i kako oni rade zajedno.

4.2.2. ECS (engl. *Entity component system*)

ECS se koristi u razvijanju videoigara. Uzorak se sastoji od objekata, u videoigrama su to, npr. vozila, neprijatelji, drveće, koji su ustvari entiteti te se svaki sastoji od više komponenti koje mu dodaju funkcionalnosti.

Entitet je svaki objekt u videoigri. Naime, ako u igri postoji dvadeset istih zgrada, to je dvadeset entiteta, a ne samo jedan. Dodavanjem komponenata mu se opisuje funkcionalnost (Adam, 2007).

Komponente su podaci o funkcionalnosti i ponašanjima koji se dodaju entitetima.

Sustav bi u video igri mogao pratiti kada npr. lik udari čudovište. Sustav bi prolazio kroz sve entitete dok ne bi pronašao one koji imaju odgovarajuću komponentu koja bi, u ovom slučaju, bila neka fizička komponenta. Sustav bi tada prepoznao da je čudovište udareno npr. mačem i zabilježilo bi to kao događaj. Ovdje se još može nalaziti komponenta koja bi bila zadužena za praćenje životnih bodova čudovišta. Prilikom pokretanja događaja udaranja mačem, pokrenula bi se i komponenta za praćenje životnih bodova te bi izračunala koliko ih se mora oduzeti čudovištu.

Svi sustavi rade istovremeno i pokreću radnje nad entitetima koji sadrže komponente istog aspekta kao sustav (Adam, 2007).

4.2.3. SOA (engl. *Service–oriented architecture*)

SOA je uzorak koji omogućava da se usluge pružaju komponentama putem komunikacijskih protokola. Usluga nije ovisna o tehnologijama ili produktima i kao takvoj joj se može pristupiti izvana i uređivati je.

Usluga ima četiri svojstva (The Open Group, 2016):

- Na logički način prikazuje aktivnosti s određenim rezultatima
- Samoodrživa je
- Ona je crna kutija (engl. *black box*) za korisnike što znači da oni mogu vidjeti svoje unose i rezultate koje im usluga vraća, no ne mogu vidjeti na koji način ona funkcionira iznutra
- Može sadržavati ostale manje servise

U listopadu 2009. godine izdan je manifest koji definira šest temeljnih vrijednosti SOA-e (SOA Manifesto, 2009):

1. Poslovnim vrijednostima pridaje se veća vrijednost nego tehničkoj strategiji.

2. Strateškim ciljevima pridaje se veća vrijednost nego specifičnim prednostima koje donose određeni projekti.

3. Unutrašnjoj interoperabilnosti pridaje se veća vrijednost nego prilagođenoj integraciji.

4. Servisima koji se mogu dijeliti pridaje se veća vrijednost nego implementacijama koje imaju specifičnu svrhu.

5. Fleksibilnosti se pridaje veća vrijednost nego optimizaciji.

6. Usavršavanju se pridaje veća vrijednost nego perfekciji koja se želi postići od početka.

Iako ne postoji standard koji bi opisao SOA-u, postoje principi kojima se vode mnoge industrije i koji su općeprihvaćeni (Guru99, 2019):

- Standardizirani ugovori servisa:

  Servisi poštuju standarde komunikacije koji su definirani u jednom ili više dokumenata.

- Autonomija referirajućih servisa:

  Veza među servisima je svedena na minimum, oni su samo svjesni ostalih servisa, a ne i njihovih uloga.

- Transparentnost lokacije servisa:

  Servisi se mogu pozivati sa koje god lokacije na mreži, bez obzira gdje se nalaze.

- Dugovječnost servisa:

  Servise se dizajnirani sa ciljam da imaju što duži životni vijek. Pozove li se neki servis danas, taj isti servis mora biti dostupan sutra.

- Apstraktnost servisa:

  Servisi djeluju kao crne kutije, korisnicima nije poznato na koji način rade, već samo vide rezultate.

- Autonomija servisa:

  Servisi su neovisni i kontroliraju funkcionalnosti koje enkapsuliraju.

- Odsutnost stanja servisa:

  Servisi nemaju stanja, oni samo mogu vratiti rezultat ili izbaciti pogrešku. Djeluju na taj način kako bi se smanjila količina resursa koje koriste.

- Granularnost servisa:

  Ovim principom osigurava se da će servisi imati određenu veličinu, a područja djelovanja i funkcionalnosti koje pružaju korisniku moraju biti relevantna.

- Normalizacija servisa:

  Servisi se normaliziraju kako bi se smanjila redundancija. Provodi se kada je potrebno optimizirati procese.

- Složenost servisa:
  Jedan servis ne smije sadržavati cijelu funkcionalnost, nego se ona mora rasporediti na više servisa.
- Otkrivanje servisa:

  Servisi sadrže metapodatke putem koji ih se može pronaći i interpretirati.

- Ponovno korištenje servisa:

  Logičko djelovanje dijeli se na više servisa, kako bi se kod mogao koristiti za neke slične sustave.

- Enkapsulacija servisa:

  Ako se neki servisi ne nalaze unutar SOA - e, oni se mogu enkapsulirati i dodati istome.

Svaki dio SOA-e može imati neku od sljedećih uloga (Medium, 2019):

- Service provider - kreira web uslugu i pruža svoje informacije registru servisa.
- Service broker, service registry ili service repository - pruža mogućnost uvida u sve informacije vezane uz web servise.
- Service request/consumer - pronalazi pokvarene zapise u registru i veže ih uz *providera* usluge kako bi se pokrenuli neki od njegovih web servisa

SOA se može implementirati s web servisima kako bi njezini dijelovi postali dostupni putem standardnih internet-protokola. Servisi mogu predstavljati nove aplikacije ili već postojeće pripremiti za rad na mreži.

4.2.4. EDA (engl. *Event-driven architecture*)

Event se može sastojati od dva dijela: zaglavlja i tijela. U zaglavlju se nalaze informacije kao što su ime, tip eventa i vrijeme kada je pokrenut, a u tijelu se mogu pronaći informacije o promjeni stanja.

Prema Michelson (2006), EDA je bazirana na slojevima kojih ima sveukupno četiri: generator *eventa* (engl. *event generator*), *event* kanal (engl. *event channel*), *engine* za procesuiranje evenata (engl. *event processing engine*) te nizvodne aktivnosti pokrenute *eventima* (engl. *downstream event-driven activity*).

- *Event* generator

  Prvi je sloj EDA-e i u njemu se generiraju svi *eventi*. Oni mogu nastati iz raznih izvora kao što su aplikacije, senzori ili *e-mail* (Michelson, 2006).

- *Event* kanal

  Zadužen je za podjelu *evenata* koji su nastali u prvom sloju. Više kanala može biti otvoreno odjednom i *eventi* se procesuiraju redom kako su stigli (Michelson, 2006).

- *Engine* za procesuiranje evenata

  Sloj je zadužen za evaluaciju pristiglih *evenata* iz prethodnog sloja. Pregledavaju se postavljena pravila u i skladu s njima se pokreću određene akcije. Na primjer, ako je na skladištu postavljeno pravilo da količine artikala ne smiju pasti ispod 10%, a dođe do toga, sustav će korisniku izbaciti pogrešku.

- Nizvodne aktivnosti pokrenute eventima

  U zadnjem se sloju prikazuju aktivnosti koje su se dogodile kao posljedica djelovanja prethodnih triju slojeva. To može biti u obliku *e-maila* poslanog korisniku ili pogreške koju aplikacija izbaci (Michelson, 2006).

Michelson u svojemu radu (2006) razlikuje tri različita načina procesuiranja *evenata*: jednostavni (engl. *simple event processing*), tekući (engl. *event stream processing*) i složeni (engl. *complex event processing*).

1. Jednostavni – odnosi se na *evente* koji imaju specifična i mjerljiva stanja. Za primjer se može pogledati na koji način mobitel javlja da je baterija pri kraju. Postotak baterije se spusti ispod prihvatljive razine što pokreće *event* koji će na ekranu prikazati upozorenje.

2. Tekući – analizira sve *evente* među uređajima i na temelju toga identificira uzorke i veze (Watts, 2018).
3. Složeni – prati i analizira sve *evente*, bez obzira sa kojega su izvori došli. Na temelju uzoraka prepoznaje bitne *evente*, uglavnom prijetnje sustavu (Rohrmann, 2016).

4.3. REST (engl. *Representational State Transfer*)

REST je vrsta arhitekture softvera koja različitim računalima na mreži omogućava da komuniciraju bez zapreka. Web servisi koji se vode pravilima REST arhitekture poznatiji su kao RWS (engl. *RESTful Web sevices*). Kako bi web servis bio prepoznat kao RWS, mora zadovoljavati sljedećih šest smjernica (Codecademy, 2017):

1. **Klijent – server arhitektura**
   U REST-u klijent i server stoje kao zasebne jedinice i ne moraju znati za postojanje drugoga, bitno je samo da znaju u kojem formatu je potrebno poslati poruku. Ovakva podjela omogućava da se promjene na strani klijenta ili servera odvijaju bez utjecaja na drugu stranu. Tako se, na primjer, kod na strani servera može izmijeniti bez da utječe na klijenta i obrnuto (Codecademy, 2017).
2. **Odsutnost stanja (engl. *statelessness*)**
   Smjernica omogućava klijentu i serveru komunikaciju bez potrebe da ijedna strana zna u kakvome je stanju druga strana.
3. **Mogućnost spremanja u priručnu memoriju (engl. *cache*)**
   Podaci koje server šalje imaju informaciju o tome mogu li se oni spremiti u priručnu memoriju ili ne. Spremanje u priručnu memoriju je moguće zbog provjere verzije podataka koji dolaze, ako verzija već postoji zahtjev će se odbiti (Avraham, 2017).
4. **Slojeviti sustav**
   Između klijenta koji je poslao zahtjev i servera može se nalaziti više servera koji imaju različite funkcije, na primjer za sigurnost ili za spremanje u priručnu memoriju. Njihovo postojanje ne bi smjelo utjecati na zahtjev ili odgovor (Avraham, 2017).

5. **Jedinstveno sučelje**

   Osnovna je osobina RESTful sustava koja pojednostavljuje i odvaja arhitekturu, pritom omogućavajući različitim dijelovima da se razvijaju neovisno o ostalima.

   - Zahtjev prema serveru mora sadržavati identifikator resursa.
   - Server mora vratiti odgovor koji sadrži dovoljno informacija kako bi se resurs na klijentskoj strani mogao izmijeniti.
   - Svaki zahtjev i odgovor sadrže informacije koje klijentu i serveru omogućuju razumijevanje istih.
   - Server može klijentu, uz odgovor slati i poveznice putem kojih može doći do više mogućnosti. Na primjer, ako klijent traži informacije o proizvodu, može mu se ponuditi poveznica koja će ga uputiti na slične proizvode.

6. **Kod na zahtjev (engl. *code on demand*)**

   Klijent ima mogućnost od servera tražiti kod koji će uglavnom dobiti u obliku skripte (Avraham, 2017).

4.4. MVC (engl. *Model-view-controller*)

MVC model vrši podjelu između prezentacijskog i logičkog dijela web aplikacije. U logički dio spada *model*, a u prezentacijski *view*, dok se *controller* nalazi između ta dva dijela kao posrednik.

Ova podjela omogućava da se svaki pojedini dio testira pomoću lažnih objekata koji oponašaju stvarne objekte aplikacije i da se svaki pojedini dio razvija paralelno s ostalima. Nije potrebno prvo razviti *model* zatim *view* i onda *controller,* već se oni svi mogu razvijati zasebno, ali istovremeno.

MVC omogućava TDD (engl. *Test Driven Development*), što je proces u razvoju softvera i aplikacija, koji se oslanja na testiranje svake pojedine mogućnosti aplikacije, dok ona ne zadovoljava uvjete postavljene u testovima (Microsoft, 2009). Ciklus TDD–a ima nekoliko koraka (Agile Data, 2006):

1. Dodavanje testa

   Prilikom dodavanja svake nove mogućnosti aplikaciji, ona se prvo mora testirati. Kako bi se napisao dobar test, mora se razumjeti što točno taj dio aplikacije mora raditi i koji su zadovoljavajući rezultati kako bi se test uspješno završio. Test se piše prema definiranoj specifikaciji i zahtjevima, kako bi se osiguralo da se aplikacija razvija u dobrom smjeru.
2. Pokretanje svih testova

   Test se pokreće i očekuje se da neće uspješno završiti, zato što nema koda koji mu je potreban. Ovaj korak je potreban kako bi se sa sigurnošću moglo utvrditi da novi test neće uspjeti i pokazivati krive uspješne rezultate.
3. Pisanje koda

   U ovom koraku piše se test koji će osigurati da novi test prođe uspješno. Kod isprva ne mora biti savršeno napisan, mora samo osigurati prolaz, a uljepšavanje koda vrši se na petom koraku.
4. Pokretanje testova

   Prođu li svi testovi uspješno i bez grešaka (koje čine druge mogućnosti aplikacije da se ponašaju čudno ili prestanu raditi), može se reći da kod zadovoljava sve dotad postavljene uvjete.

5. Refaktoriranje koda

   Zadnji korak u ciklusu je uljepšavanje koda. Brišu se duplikati, a imena varijabli, klasa, modula, objekata i metoda moraju predstavljati ono za što se koriste. Kod se može preslagivati kako bi bio logički pregledniji.

Svakim novim testom ovih se pet koraka ponavlja, pritom se osigurava da sve funkcionalnosti rade kao jedna cjelina i u skladu sa zahtjevima.

Model

Model u MVC sadrži sve podatke i logiku rada aplikacije. *View* se oslanja na njega kako bi dobio tražene informacije, a *controller* mu govori kako reagirati na ono što korisnik želi. Na primjer, ako korisnik želi dobiti cijelu listu proizvoda u nekoj trgovini, *controller* to pretvara u razumljivu naredbu modelu, koji zatim *viewu* daje ono što je potrebno, odnosno prikaz svih proizvoda.

View

View je dio aplikacije koji je zadužen za prikazivanje informacija korisnicima, što se uglavnom odnosi na grafičku komponentu sučelja. Kako bi *view* imao što za prikazati korisniku, mora usko surađivati sa *model* komponentom. *Model* komponenta sadrži podatke koje *view* preuzima i prikazuje ih korisniku (Guru99, 2019).

Controller

Controller je dio aplikacije koji prenosi zahtjeve korisnika ostalim modelima. Pokazuje im koje je radnje korisnik poduzeo i što mu se treba prikazati ili napraviti (Guru99, 2019).

4.5. PHP

PHP je programski jezik opće namjene koji je isprva zamišljen kao jezik za izradu web aplikacija. Programski jezici opće namjene dizajnirani su na načina da mogu podržavati širok spektar domena u kojima se mogu koristiti i što se pomoću njih može izraditi. Domenski specifični jezici su uglavnom specificirani za izradu jedne domene, kao, što je, na primjer, HTML, koji se koristi za izradu web stranica i Unreal Engine za izradu video igara. Granica između jezika opće namjene i domenski specifičnih jezika nije uvijek najjasnija, neke mogućnosti domenski specifičnih jezika mogu se koristiti i za neka druga područja (Kramer i Sturm, 2005).

Akronim PHP stoji za *PHP: Hypertext Preprocessor*, to je programski jezik otvorenog koda te je posebno namijenjen za izradu web aplikacija. Glavni cilj PHP–a je izrada dinamički generiranih web stranica. Inspiracija pri osmišljanju sintakse su mu bili C, Java i Perl te je zbog toga vrlo jednostavan za brzo korištenje i razvijanje u njemu.

U svojim počecima, PHP je bio poznatiji kao PHP/FI. Razvio ga je Rasmus Lerdorf 1994. godine (PHP, 2019). Razvio ga je kako bi mogao pratiti broj pregleda svojeg životopisa, a napisan je u programskog jeziku C. Razvijeni skup skripti nazivao je *Personal Homepage Page Tools*, odnosno PHP Tools. Nakon nekog vremena, Lerdorf je pregledao i ponovno napisao PHP Tools. Nova inačica imala je mogućnost povezivanja sa bazama podataka i *framework* koji su korisnici mogli koristiti kako bi sami mogli razviti web aplikacije prema vlastitim željama.

U rujnu 1995. godine, Lerdorf je objavio izvorni kod PHP Tools–a, što je korisnicima omogućilo da koriste PHP Tools kako sami žele. Također im se dopuštalo da prijavljuju sve greške pronađene u radu te, ako su u mogućnosti, da ih isprave. Veći broj programera mogao je raditi sa PHP Tools, prijavljivati svoje pogreške i rješenja ili drugima pomagati sa njihovim problemima te na taj način doprinositi razvoju samoga jezika (PHP, 2019).

Lerdorf je u svojoj obavijesti naveo nekoliko funkcija koje PHP Tools posjeduje (1995):

- Bilježene pristupa u vlastitim privatnim *logovima.*
- Pregled *loga* u realnome vremenu.
- Razumljivo sučelje za pregled *logova.*
- Prikaz posljednjih pristupa korisnikovim stranicama.
- Brojače koji su bilježili pristupe na dnevnoj razini i koje se mogli prikazati na sveukupnoj razini.
- Ukidanje pristupa stranicama blokiranjem određenih domena.
- Zahtijevanje zaporke ako se pristupa putem određenih domena.
- Praćenje pristupa putem *e-mail* adresa.
- Mogućnost da se ne prati pristup sa određenih domena.
- Lako kreiranje i prikazivanje obrazaca.

U rujnu 1995. godine, PHP je nadograđen i tada se nije zadržalo ime PHP, već je samo ostalo FI (engl. *Forms Interpreter*). Ova nova verzija zadržala je neke mogućnosti PHP-a, imala je varijable slične Perlu, automatsko interpretiranje varijabli obrazaca i HTML. Sintaksa je bila slična Perlovoj samo što je bila dosta jednostavnija, limitiranija i u jednu ruku nekonzistentna (PHP, 2019).

Ime FI nije se dugo zadržalo te je u listopadu 1995. godine jezik prošao još jedno ponovno pisanje i postao *Personal Home Page Construction Kit*. Jezik je s namjerom dosta sličan C-u, kako bi se korisnici koji su već upoznati s njim, Perlom, ili sličnim jezicima, imali jednostavniju prilagodbu novome jeziku.

U travnju 1996. godine dolazi do još jedne revizije koja spaja imena PHP i FI te nastaje PHP/FI. Ova revizija je imala podršku za DBM, mSQL i Postgres95 baze podataka, kolačiće, itd…Iste godine u lipnju jezik je nadograđen na verziju 2.0.

PHP/FI je uživao veliku popularnost u vremenu kada je razvoj web aplikacija bio još na početcima. U programskom kodu 19 vidljiv je primjer PHP/FI koda. Godina 1997. i 1998., imao je nekoliko tisuća korisnika u svijetu i, prema Netcraftovoj anketi provedenoj u svibnju 1998. godine, blizu 60,000 domena je imalo zaglavalja koja su sadržavala PHP (PHP, 2019).

```
<!--include /text/header.html-->
<!--getenv HTTP_USER_AGENT-->
<!--ifsubstr $exec_result Mozilla-->
  Hey, you are using Netscape!<p>
<!--endif-->
<!--sql database select * from table where user='$username'--
>
<!--ifless $numentries 1-->
  Sorry, that record does not exist<p>
<!--endif exit-->
  Welcome <!--$user-->!<p>
  You have <!--$index:0--> credits left in your account.<p>
<!--include /text/footer.html-->
```

Programski kod 19. Primjer PHP/FI koda

PHP 3.0

Verzija 3.0 bila je najsličnija PHP-u kakvog poznajemo danas. Andi Gutmans i Zeev Suraski iz Tel Aviva 1997. godine razvijali su aplikaciju koristeći PHP/FI. Zbog nedostataka koje je PHP/FI imao, krenuli su sa ponovnim pisanjem parsera te su stupili u kontakt sa Lerdorfom i to s prijedlogom ponovnog pisanja i implementacije. Lerdorfu se ideja svidjela te su krenuli u razvijanje novog, neovisnog programskog jezika koji su predstavili pod imenom PHP (PHP, 2019).

Ova verzija predstavljena je javnosti u lipnju 1998. godine i testirana je punih devet mjeseci, sve do ožujka 1999. godine. U to vrijeme PHP 3.0 bio je instaliran na više od 70,000 domena na globalnoj razini i otprilike 10% web servera. Više nije bio limitiran samo na POSIX operacijske sustave nego se koristio i na Windows 95, Windows 98, Windows NT i Macintosh operacijskim sustavima (PHP, 2019).

PHP 3.0 podržavao je objektno–orijentirano programiranje, konzistentnu sintaksu, sučelje za veći broj baza podataka, protokola i API–a (PHP, 2019).

PHP 4.0

Gutmans i Suraski željeli su sam rad aplikacija učiniti boljim i unaprijediti modularnost PHP-a, ali PHP 3.0 nije bio sposoban nositi se sa kompleksnijim aplikacijama. Kako bi to bilo moguće izvesti razvili su „*Zend Engine*“ na kojemu je baziran PHP 4.0. Osim što se mogao nositi sa kompleksnijim aplikacijama, također je podržavao veći broj web servera, HTTP sesije i dozu sigurnosti prilikom rada sa korisničkim inputima (PHP, 2019).

„*Zend Engine*“ predstavljen je 1999. godine, a službeno je izašao 2000. godine, pisan je u C-u i PHP ga koristi kao vlastiti kompajler. PHP svoje skripte učitava u „*Zend Engine*“ koji ih kompajlira u Zend opkodove, ti kodovi se zatim pokreću i generiraju HTML koji se šalje klijentu (PHP, 2007).

PHP 5.0

U svibnju 2004. godine stiže PHP 5.0. zajedno sa „*Zend Engine II*“. Ova verzija proširila je svoju podršku za objektno–orijentirano programiranje i novu ekstenziju za PHP podatkovne objekte (engl. *PHP Data Objects*) koja je osiguravala konzistentno sučelje za pristup bazama podataka (PHP, 2019).

U veljači iste godine prestala je podrška za PHP 4 te se programere poticalo da prijeđu na PHP 5, putem *GoPHP5* inicijative. Prestanak podrške značio je da su programeri morali sami održavati svoje PHP 4 projekte i pritom ne koristiti nove mogućnosti koje su došle s novom verzijom, ili su mogli prijeći na PHP 5 i uživati u novim prednostima (GOPHP5, 2011).

Za razliku od četvrte verzije, peta verzija imala je bolju podršku za XML, koji je tada bio među najčešće korištenim web servisima. PHP 5 je XML mogao procesuirati brzo i lagano što je omogućavalo lakši i brži razvoj web aplikacija. PHP 4 je za svaku bazu podataka koristio drugačiji set funkcija, što je znatno otežalo njihovo održavanje. Pojavom PDO-a u novijoj verziji, više nije bilo potrebe za više različitih setova funkcija za različite baze podataka (GOPHP5, 2011).

PHP 6.0

Šesta verzija trebala je omogućiti podršku za UNICODE standard i UTF–16 kodiranje. Do toga nikada nije došlo zbog raznih problema na koje su nailazili prilikom razvoja (PHP, 2019).

PHP 7.0

PHP 7 pojavio se 2015. godine, a zajedno s njime „*Zend Engine 3*“. Brzina se, s obzirom na prethodnu verziju, povećala skoro dvostruko. Korištenje memorije se smanjilo i dobila se mogućnost specificiranja varijabli koje je do tada PHP sam stavljao, a između ostalog i podrška za *integer*, *float*, *string* i *boolean* tipove podataka (Lerdorf, 2019).

4.5.1. Sintaksa

PHP sintaksa najsličnija je C sintaksi, uz nekoliko pravila koja se moraju poštivati kako bi se osiguralo lakša interoperabilnost među PHP kodovima (PHP-FIG, 2013).

- Moraju se koristiti samo oznake <?php ili <?=.
- Mora se koristiti samo UTF–8 kodiranje znakova.
- Trebaju se deklarirati sve klase, funkcije, konstante ili promijeniti postavke u *ini*. datoteci. Jedno od toga, a ne oboje.
- Svaka klasa mora biti u zasebnoj datoteci.
- Sve klase se moraju deklarirati kao *StudlyCaps* (metoda pisanja gdje se neka slova pišu velikim tiskanim slovima, a neka malim, po određenom uzorku).
- Konstante moraju biti pisane velikim tiskanim slovima koje dijeli *underscore* (_) simbol.
- Metode se moraju deklarirati kao camelCase (metoda pisanja u kojoj, ako imamo spojene dvije riječi, druga riječ započinje sa velikim tiskanim slovom, a prva se riječ piše malim tiskanim slovima).

PHP podržava deset primitivnih tipova podataka koji su podijeljeni u tri kategorije: skalarni (engl. *scalar*) tipovi podataka, složeni (engl. *compound*) tipovi podataka te specijalni (engl. *special*) tipovi podataka.

Skalarni tipovi

Skalarni tipovi mogu sadržavati samo jednu informaciju. U ovu skupinu spadaju *boolean*, *integer*, *float* i *string*.

1. ***Boolean***

   Booleani su dobili ime prema Georgeu Booleu (2. studenog 1815. – 8. prosinca 1864.), britanskom matematičaru i filozofu po kojemu je ime dobila i Booleova algebra (Britannica, 2019). *Boolean* tip podataka može imati samo jednu od dvije vrijednosti: *true ili false*, vrijednosti se mogu pisati malim ili velikim slovima. U PHP-u pretvaranjem u *boolean* sve sljedeće vrijednosti bit će *false (PHP, 2006)*:

   - Sam boolean *false*
   - Integeri 0 i -0.0 (nula)
   - Float 0.0 i -0.0 (nula)
   - Prazni stringovi i string „0“
   - Array s nula elemenata
   - Specijalni tip podatka NULL
   - SimpleXML objekti nastali iz praznih oznaka

2. ***Integer***

   Integeri su cijeli brojevi koji su podržani u PHP-u. Predstavljeni su kao decimalni (baza 10), heksadecimalni (baza 16), oktalni (baza 8) i binarni (baza 2) brojevi (PHP, 2007). Za različite notacije koriste se različiti predznaci i vidljivi su u programskom kodu 20:

```
$a = 1234              // decimalni broj
$a = -123             // negativni broj
$a = 0132            // oktalni broj
$a = 0x1A           // heksadecimalni broj
$a = 0b11111111   //binarni broj
```

Programski kod 20. Primjer integera

3. ***Float***

Float brojevi, još poznati kao (*floats*, *doubles* ili realni brojevi) u PHP se mogu definirati korištenjem jedne od sljedećih sintaksi vidljivih u programskom kodu 21 (PHP, 2006):

```
$a = 1.234

$b = 1.2e3

$c = 7E-10
```

Programski kod 21. Float

4. *String*

Stringovi u PHP-u su slijed znakova gdje je jedan znak jednak jednom bajtu. Od svoje sedme verzije, PHP više nema ograničenja na duljinu stringa na 64-bitnim verzijama (engl. *builds*), dok na 32-bitnim verzijama on može biti veličine do 2 GB (najviše 2147483647 bitova) (PHP, 2016).

PHP stringove može se označiti na četiri načina: navodnicima i duplim navodnicima, čiji su primjeri vidljivi u programskom kodu 22, te *heredoc* sintaksom i *nowdoc* sintaksom, čiji su primjeri vidljivi u programskom kodu 23 i 24.

```
echo 'string sa navodnicima'

echo ''string sa duplim navodnicima''
```

Programski kod 22. Označavanje stringova

Heredoc označavanje započinje se „<<<“ operaterom nakon kojega slijedi identifikator i zatim *string*. Identifikator za zatvaranje mora stajati u prvome stupcu i mora pratiti pravila imenovanja PHP-a; smije se pisati samo alfanumeričkim znakovima i mora započinjati sa slovom ili „_“ (engl. *underscore*) (PHP, 2016).

Nowdoc koristi ista pravila kao i *heredoc,* samo što se prvi identifikator mora stavljati u navodnike. Ostala pravila koja vrijede za *heredoc* vrijede i za *nowdoc* (PHP, 2016).

```
$str = <<<IDE

String koji

pokriva

više linija.

IDE;
```

Programski kod 23. Heredoc sintaksa

```
$str = <<<'IDE'

String koji

pokriva

više linija.

IDE;
```

Programski kod 24. Nowdoc sintaksa

Složeni (engl. *compund*) tipovi

1. *Array*

Array u PHP-u može sadržavati više od jedne vrijednosti i može ga se upotrebljavati umjesto varijabli. Ako se, na primjer, želi sve vrste životinja u Republici Hrvatskoj staviti u varijable, to bi bio prevelik posao pa ih se može staviti u *array*. Svakoj životinji dodijeljen je indeks koji uvijek kreće od nule ili se može ručno dodijeliti, što je vidljivo u programskom kodu 25.

```
//automatski dodijeljen indeks (0, 1, 2)

$zivotinje = array(''pas'', ''vuk'', ''ris'');

//ručno dodijeljen indeks

$zivotinje[0] = ''pas''

$zivotinje[1] = ''vuk''

$zivotinje[2] = ''ris''
```

Programski kod 25. Automatski i ručno dodijeljen indeks

2. Objekti

PHP objekti kreiraju se iz klasa od kojih će preuzeti metode i svojstva. Objekti se međusobno razlikuju po imenima, ali i po vrijednostima koje imaju. Svaki objekt koji je nastao iz iste klase neće imati sve iste vrijednosti (PHP Enthusisast, 2015). Kako bi se iz klase stvorio novi objekt potrebno ga je kreirati ključnom riječi „new“.

U složene tipove spadaju još i *callable* i *iterable*. *Callable* su sve funkcije, klase, objekti i metode koje se mogu pozivati, a *iterable* se koristi kada se želi naglasiti da funkcija zahtijeva set vrijednosti, bez obzira u kakvom se obliku nalazili (PHP, 2017).

Specijalni tipovi

U specijalnim tipovima podataka nalaze se samo *resources* i NULL. *Resources* se koristi kada se želi referencirati vanjski izvor i koristi specijalnu funkciju get_resource_type() (PHP, 2009). NULL tip podatka predstavlja varijablu koja nema dodijeljene vrijednosti (PHP, 2017).

4.5.2. PEAR (engl. PHP Extension and Application Repository)

PEAR je ekstenzija za PHP koju je prvi započeo razvijati Stig Saether Bakken 1999. godine sa misijom da se često korišteni kod PHP-a koristi u raznim projektima bez potrebe da se stalno iznova piše (PEAR, 2019). PEAR programerima omogućava da na brz i efikasan način pronađu i koriste potreban kod.

PEAR nudi sljedeće mogućnosti:

- Strukturiranu zbirku kodova koji su slobodni za korištenje te su na raspolaganju svim PHP programerima.
- Sustav koji je zadužen za distribuiranje i održavanje paketa.
- Cjelokupni kod koji ima standardiziran stil pisanja.
- PECL (engl. *PHP Extension Community Library*) koji služi za distribuiranje PHP ekstenzija, koje je moguće instalirati pomoću PEAR *instalera*
- Web stranicu i mailing liste

Sve PEAR ekstenzije se nazivaju paketima. U bazi ih se trenutno nalazi 604 raspoređenih kroz 46 kategorija i preuzeti su 261,546,579 puta (PEAR, 2006).

PEAR je projekt pokrenut od strane zajednice i kao takav dopušta svim PHP programerima da svoje razvijene pakete ustupe na korištenje svima. No, da bi se paket ustupio svima, programer istoga mora proći nekoliko koraka prije nego što to bude moguće. On svoj paket šalje moderatorima koji glasanjem odlučuju hoće li se paket prihvatiti ili odbiti. Moderatori imaju sedam kalendarskih dana kako bi dali svoj glas. Glasovi se daje u brojčanom obliku od -1 do 1 gdje -1 znači da je moderator protiv prihvaćanja paketa, 0 da je kod pregledan, ali moderator neodlučan , a 1 da je kod za prihvaćanje. Kako bi bio prihvaćen od strane moderatora, zbroj svih glasova mora biti jednak ili veći od pet (PEAR, 2006).

4.6. HTML (engl. HyperText Markup Languge)

HTML je standardni jezik za označavanje, koji je zadužen za strukturu elemenata na stranici. HTML nije programski jezik i kao takav ne opisuje što koji element radi, već određuje gdje se oni na stranici nalaze te kako izgledaju. Uzme li se, na primjer tablica koja se nalazi na nekoj web stranici, korištenjem HTML-a može se odrediti gdje će se točno tablica nalaziti na stranici, hoće li biti u lijevom ili desnom kutu ili centrirana.

HTML spada u grupu jezika za označavanje zvanim deskriptivni (engl. *descriptive*) jezici. Deskriptivni jezici za označavanje služe za opisivanje dijelova dokumenata bez uputa kako bi se oni trebali i mogli procesuirati (Bray, 2003). U ovo grupu osim HTML-a spadaju i LaTex i XML.

Osim deskriptivnih jezika za označavanje, postoje još i prezentacijski (engl. *presentational*) jezici za označavanje te proceduralni (engl. *procedural*) jezici za označavanje.

Prezentacijski jezici za označavanje koriste kodiranje kako bi se naznačilo što u tekstu mora kako izgledati (Bray, 2003). Na primjer, ako se u tekstu svaki strani pojam podvuče i naglasi, koda iza njega će ukazati „ovo je podvučeno“ i „ovo je naglašeno“.

Proceduralni jezici za označavanje računalima pokazuju kako se tekst mora procesuirati. U ovu skupinu spadaju: troff, PostScript i TeX jezici (Coombs et al., 1987).

Prvi dokument koji opisuje HTML nastao je 1991. godine i napisao ga je Tim Berners–Lee pod imenom „*HTML Tags*“. U ovom dokument opisano je 18 elemenata HTML-a, a njih 11 je još postojalo u HTML-ovoj četvrtoj verziji. Druga verzija HTML-a nastala je 1995. godine s namjerom da se prihvati kao standard na koji će se sve buduće verzije oslanjati. Sljedeće godine, održavanje HTML-a prelazi na W3C (engl. *World Wide Web Consortium*), koji 1997. godine objavljuje treću verziju, 1998. godine četvrtu verziju te 2014. godine petu i posljednju verziju (W3O, 1998).

Svaki HTML dokument sastoji se od niza oznaka (engl. *tags*) i njihovih atributa, čime se definiraju svojstva. Oznake uvijek dolaze u parovima osim praznih elemenata poput <img> koji mogu stajati samostalno. Prva oznaka u dokumentu naziva se otvarajuća (engl. *opening*) oznaka, a posljednja zatvarajuća (engl. *closing*) oznaka. Otvarajuću oznaku pišemo u obliku <title>, a zatvarajuću u istome obliku, samo što dodajemo / (kosu crtu) nakon znaka < (manje od), što izgleda ovako: </title>. Ovim primjerom zatvoren je *title* element, u kojem se definira naslov stranice.

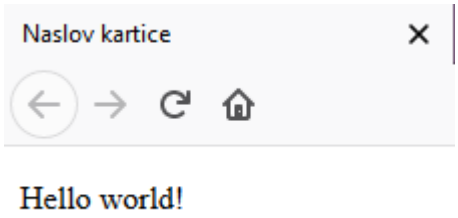
Primjer *Hello World!* HTML dokumenta

```
<!DOCTYPE html>
<html>
  <head>
    <title>Naslov kartice</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

Programski kod 26. HTML oznake

U programskom kodu 26 vidljivo je da svaka oznaka: <html>, <head>, <title>, <body> i ima svoju odgovarajuću zatvarajuću oznaku: </html>, </head>, </title>, </body> i .

Sve ono što se nalazi unutar oznake <head> nije vidljivo na web stranici, osim oznake <title> koja definira ono što će pisati na kartici (engl. *tab*) Internet-preglednika koji se koristi. Oznaka <body> označava ono što će na stranici biti vidljivo. U ovome slučaju to je tekst *Hello world!*, koji se nalazi unutar oznake koja označava novi paragraf. Na *slici 1.* vidljiv je izgled web stranice i naslov kartice kako je definirano HTML oznakama.



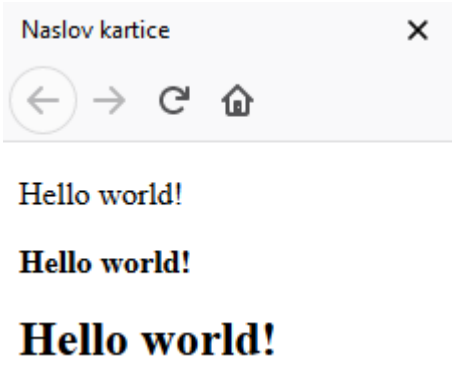
Slika 1. Hello World!

Dodavanjem još elemenata u *Hello world!*, može se uređivati prikaz web stranice.

```
<!DOCTYPE html>

<html>

<head>

<title>Naslov kartice</title>

</head>

<body>

<p>Hello world!</p>

<p><strong>Hello world!</strong></p>

<p><strong><font size="5">Hello world!</font></strong></p>

</body>

</html>
```

Programski kod 27. Dodatne HTML oznake

U programskom kodu 27, oznakom istaknulo se da znakovi moraju biti naglašeni (engl. *bold*), a u zadnjem paragrafu uz oznaku <bold> također je prisutna i oznaka <font> koja dodatkom *size* definira veličinu slova za prikaz. Ovime se dobije prikaz koji se nalazi na *slici 2*.



Slika 2. Hello World! verzija 2

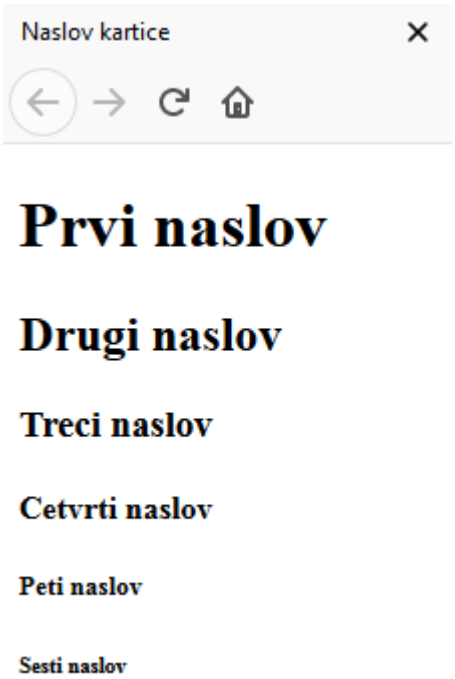
Oznaka <!DOCTYPE html> koristi se kako bi se deklariralo da se radi o HTML verziji 5. Ako se ta oznaka ne nalazi na vrhu HTML dokumenta, preglednici web stranice prikazuju bez zadovoljavanja W3C (engl. *World Wide Web Consortium*) i IETF (engl. *Internet Engineering Task Force*) standarda.

U <body> element HTML dokumenta se također može staviti i naslov web stranice. Naslove označavamo sa oznakom <hx> gdje *x* označava broj naslova. U HTML-u ih može biti sveukupno šest: <h1>, <h2>, <h3>, <h4>, <h5>, <h6> te također imaju svoje zatvarajuće oznake: </h1>, </h2>, </h3>, </h4>, </h5>, </h6>, što je vidljivo u programskom kodu 28.

```
<html>
  <head>
    <title>Naslov kartice</title>
  </head>
  <body>
    <h1>Prvi naslov</h1>
    <h2>Drugi naslov</h2>
    <h3>Treci naslov</h3>
    <h4>Cetvrti naslov</h4>
    <h5>Peti naslov</h5>
    <h6>Sesti naslov</h6>
  </body>
</html>
```

Programski kod 28. Dodavanje naslova

Ovim oznakama dobije se prikaz koji se nalazi na *slici 3*.



Slika 3. Naslovi u HTML-u

Vidljivo je da su svi naslovi podebljani, ali i da se, što se više ide u dubinu, smanjuje veličina znakova. Prvi naslov je najveći, dok je šesti jedva vidljiv na stranici.

4.6.1. Oblikovanje teksta

Za oblikovanje teksta za prikaz u HTML dokumentu, koristi se više oznaka koje se mogu zajedno kombinirati. U nastavku će biti nabrojane najčešće korištene oznake.

- …
  Oznaka služi za kreiranje novih paragrafa. Oznaka je postojala u dokumentu „*HTML tags*“, standardiziran je u HTML verziji 2 i koristi se i dalje.
- …
  Želi li se neki tekst prikazati u nakrivljenom (*italic*) stilu koristi se oznaka . Oznaka je standardizirana u HTML verziji 2 te se koristi i danas.
- …
  Oznaka koristi se za označavanje teksta koji se smatra bitnim. Tekst se prikazuje u podebljanoj verziji. Standardizirana je u HTML verziji 2 i koristi se i danas.
- …
  Ova oznaka tekst također prikazuje u nakrivljenom (*italic*) stilu, ali je značenje drugačije. Koristi se kako bi se istaknule fraze na stranom jeziku ili misli. Oznaka je standardizirana u HTML verziji 2 i koristi se i danas.
- <u>…</u>
  Oznaka se koristi za podcrtavanje riječi ili rečenice u tekstu.
- <small>…</small>
  Koristi se kako bi se smanjila veličina fonta.
- <s>…</s>
  Oznaka se koristi kako bi se precrtao dio teksta koji više nije relevantan u tekstu.
-
…</br>
  Oznaka za prelazak u novi red.
- <mark>…</mark>
  Koristi se kako bi se tekst označio žutim markerom.

- […]

  Oznaka podiže tekst.
- _…

  Oznaka spušta tekst.
- <hr>…</hr>

  Oznaka iscrtava vodoravnu crtu
- <code>…</code>

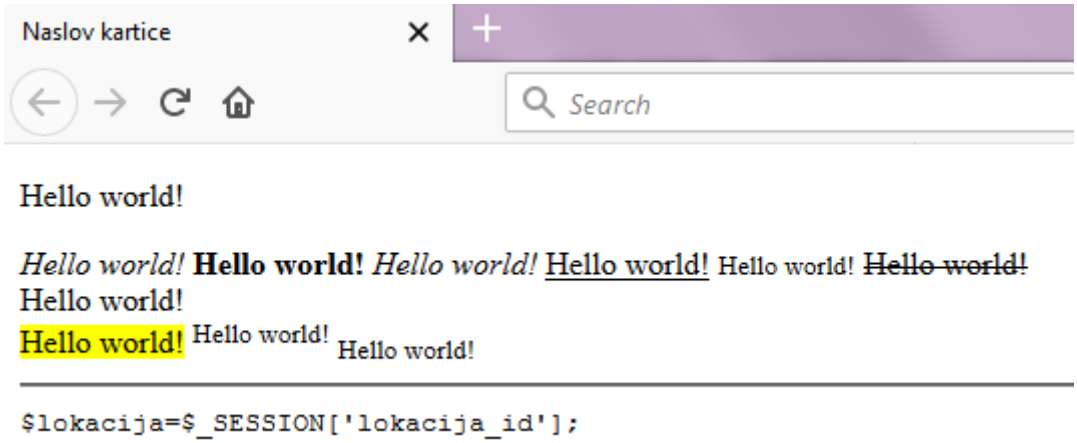
  Oznaka koja kod stavlja u malu kućicu.

Primjer korištenja svih gore navedenih oznaka vidljiv je u programskom kodu 29.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Naslov kartice</title>
  </head>
  <body>
    <p>Hello world!</p>
    <em>Hello world!</em>
    <strong>Hello world!</strong>
    <i>Hello world!</i>
    <u>Hello world!</u>
    <small>Hello world!</small>
    <s>Hello world!</s>
    <br>Hello world!</br>
    <mark>Hello world!</mark>
    <sup>Hello world!</sup>
    <sub>Hello world!</sub>
    <hr></hr>
    <code>$lokacija=$_SESSION['lokacija_id'];</code>
  </body>
</html>
```

Programski kod 29. Oznake za oblikovanje teksta

Ovim oznakama dobije se izgled web stranice koja se nalazi na *slici 4.*



Slika 4. Oblikovanje teksta

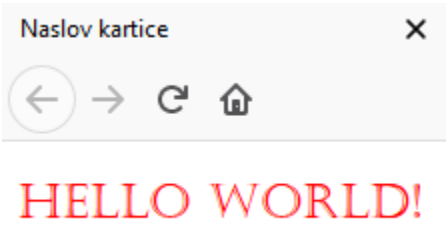
Osim oblikovanja teksta, moguće je promijeniti veličinu, stilove i boje slova sljedećim oznakama: , i koje su vidljive u programskom kodu 30.

U HTML dokumentu oznake bi bile poredane na sljedeći način:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Naslov kartice</title>
  </head>
  <body>
    <p><font size="5"><font color="red"><font
face="Castellar">Hello world!</font></font></font></p>
  </body>
</html>
```

Programski kod 30. Oblikovanje slova

Dobije se prikaz koji se nalazi na *slici 5.*



Slika 5. Oblikovanje slova

Iako se za oznaku boje slova koristilo samo ime boje, može se koristiti i heksadecimalni zapis boje koji bi u ovom slučaju bio #ff0000.

4.6.2. HTML Forme

Forme su elementi HTML-a koji korisnicima omogućuju da unose podatke koji se zatim prosljeđuju serveru koji ih procesuira. One mogu biti u obliku praznih polja u koje korisnici unose podatke, gumba ili polja koja korisnik označava (engl. *checkbox*). Definiraju se putem oznake <form> koja određuje da se podaci, koji se u nju upisuju, negdje zapišu i mogu se sastojati od više elemenata (w3schools, 2019).

Tekst – prazna kućica u koju korisnici upisuju tekst.

E-mail – prazna kućica u koju se upisuje tekst u obliku *e-mail* adrese.

Broj – prazna kućica u koju se upisuje brojevi.

Lozinka – prazna kućica u kojoj je upisani tekst sakriven ili zamijenjen sa znakom „*“.

Radio Gumb – korisniku nudi par mogućnosti od koji se može označiti samo jedna. Uglavnom se prikazuju kao prazna okrugla polja, kada korisnik izabere jednu od opcija, tada to polje dobiva točkicu. Kao primjer se može uzeti odabir spola prilikom ispunjavanja forme, korisniku se samo nudi „muško“ i „žensko“ te se može odabrati jedna stavka, a ne obje.

Datoteka – omogućuje odabir lokalno pohranjene datoteke na korisnikovom računalu i njezino pohranjivanje na web server. Pohranjivanje se odrađuje tek kada se odabere datoteka i završi forma.

Gumb za resetiranje - vraća pohranjene vrijednosti kako su bile prije korisnikovih promjena.

Gumb za potvrdu – potvrđuju se podaci koji su upisani ili izmijenjeni u formi. Nakon što se gumb pritisne, promjene se bilježe na serveru.

Polje za tekst – omogućuje upisivanje teksta, ali, za razliku od elementa „tekst“, može se unijeti više redova teksta, a ne samo jedan.

Padajući *(*engl*. Drop-down)* izbornik – nudi nekoliko predefiniranih vrijednosti koje korisnik odabire. Vrijednosti se izlistaju nakon što korisnik pritisne polje za padajući izbornik.

HTML 5 je uveo nekoliko oznaka koje od korisnika zahtijevaju određeni format unosa. Ako je polje označeno kao *e-mail*, korisnik mora unijeti *e-mail* adresu u ispravnom formatu. Oznaka za telefonski broj „tel“ zahtijeva unos u formatu telefonskog broja, a „number“ unos numeričkih vrijednosti (w3schools, 2019).

4.6.3. HTTP

HTTP (engl. *HyperText Transfer Protocol*) je protokol koji se na internetu koristi za prijenos slika, zvukovnih zapisa i HTML dokumenata od web servera do klijentskog internet preglednika. HTTP radi na klijent–server modelu gdje je *internet*-preglednik klijent, a server je web server kojemu šalje zahtjev za podacima o web stranici (Techopedia, 2011). HTTP zahtjev izvodi se u nekoliko koraka:

1. Pokreće se veza prema HTTP serveru.
2. Zahtjev se šalje prema serveru.
3. Server procesuira zahtjev.
4. Server šalje odgovor na zahtjev.
5. Veza se zatvara.

Svakom novom pretragom na *internet*-pregledniku otvara se nova veza prema web serveru i šalje se zahtjev koji server vraća u obliku rezultata pretrage. Odlaskom na neki od ponuđenih rezultata šalje se novi zahtjev, gdje *internet*-preglednik kao odgovor dobiva informacije kako odabrana web stranica mora izgledati. Svakim daljnjim prelaskom na nove stranice putem poveznica (engl. *links*), otvara se novi zahtjev.

HTTP je aplikacijski protokol koji se pokreće zajedno s TCP/IP (engl. *Transmission Control Protocol/Internet Protocol*). On omogućuje komunikaciju raznim međusobno

povezanim mrežama te sastoji se od četiri sloja: internetskog, mrežnog, transportnog te aplikacijskog sloja (Oracle, 2010).

Mrežni sloj

Mrežni sloj je najniži sloj u TCP/IP protokolu. To je skupina metoda i protokola koji djeluju samo na onoj mreži na koju je korisnik fizički spojen (Oracle, 2010).

Internetski sloj

Internetski sloj zadužen je za prihvaćanje i slanje paketa. Sam odlučuje o putu kojim će paketi stići do cilja bazirajući se na IP adresi domaćina (engl. *host*). Ako je paket prevelik, razbija se na manje dijelove, šalje i zatim ponovno slaže. U ovome sloju nalaze se IP (engl. *Internet Protocol*), ARP (engl. *Address Resolution Protocol*) i ICMP (engl. *Internet Control Message Protocol*) (Oracle, 2010).

Transportni sloj

Transportni sloj osigurava da svi paketi stignu bez greške i ispravnim redoslijedom, a ako dođe do pogreške, zahtijeva ponovno slanje. Protokoli koji se nalaze u ovom sloju su TCP (engl. *Transmission Control Protocol*) i UDP (engl. *User Datagram Protocol*) (Oracle, 2010).

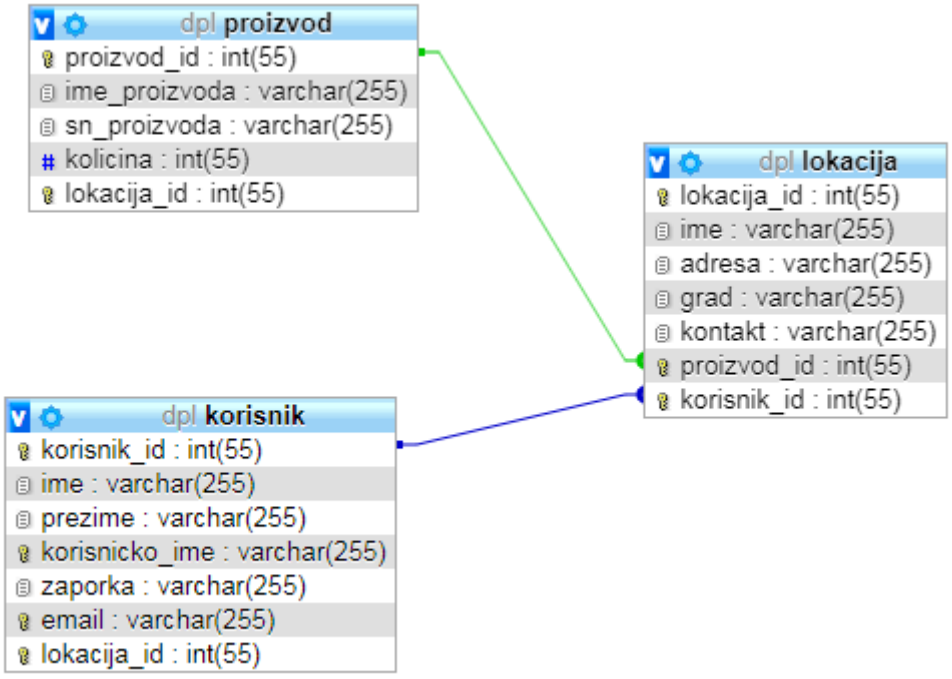
Aplikacijski sloj

Aplikacijski sloj najviši je sloj TCP/IP modela. Opisuje rad aplikacija i procesa u mreži i koristi niže modele kako bi dobio tražene podatke. SMTP (engl. *Simple Mail Transfer Protocol*), FTP (engl. *File Transfer Protocol*), SSH (engl. *Secure Shell*) i HTTP (engl. *Hypertext Transfer Protocol*) protokoli djeluju u ovom sloju (Oracle, 2010).

5. Web aplikacija

Za kreiranje web aplikacije za upravljanje inventarom korišten je mySQL i PHP jezik.

U mySQL-u su kreirane tri tablice (*slika 6.*): korisnici, lokacija i proizvod. Korisnicima aplikacije je omogućeno dodavanje, izmjena i brisanje proizvoda iz inventara te pregled svih proizvoda po lokacijama.



Slika 6. Tablice u bazi podataka

Kako bi se pristupilo bazi podataka koja je kreirana, potrebno je uspostaviti konekciju s njom.

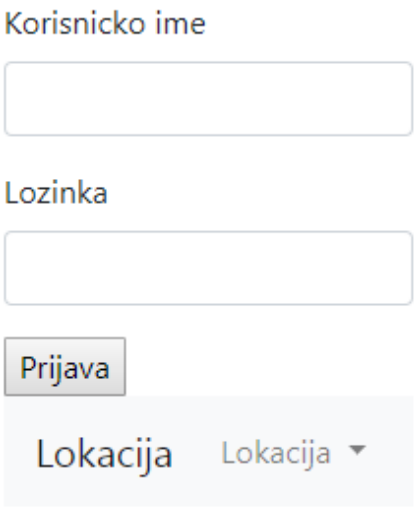
5.1. Prijava

Nakon što je spajanje s bazom podataka prošlo uspješno slijedi proces prijave u sustav. Korisnici imaju svoje korisničko ime, lozinku i oznaku lokacije. Sva tri polja (*slika 7.)* je potrebno ispravno popuniti, kako bi se došlo do naslovne strane.

5.2. Autentikacija

Provjeravanje ispravnosti upisanih podataka vrši se nakon što korisnik pritisne gumb za prijavu. U pozadini se zatim učitava drugi PHP dokument koji, ako je sve ispravno upisano, korisnika preusmjerava na njegovu naslovnicu.

Pokrenuta je sesija session_start(); kako bi se sačuvala lokacija_id. Ovaj identifikator je potreban kako bi se korisnicima mogli ispisati samo oni zapisi koji se odnose na njihove lokacije. Zato se oznaka lokacije zahtijeva prilikom prijave, kako bi se mogla spremiti u sesiju i koristiti u daljnjem radu sa zapisima. Ako ove oznake ne bi bilo, svim korisnicima bi se ispisivao cijeli inventar, bez obzira na lokaciju na kojoj se ustvari nalazi i za koju ima prava.



Slika 7. Forma za prijavu

5.3. Naslovna stranica

Korisnik nakon uspješne prijave dolazi na svoju personaliziranu naslovnu stranicu (*slika 8.*), gdje mu se u formatu tablice ispisuje sav trenutni inventar lokacije.

Slika 8. Naslovna stranica

Ovo je moguće zbog polja lokacijaID koji prenosimo od trenutka kada se korisnik uspješno prijavio.

Kako bi to prenošenje bilo moguće sesija se mora započinjati u svakoj novoj PHP datoteci.

Korisniku se na naslovnoj strani nude tri gumba s različitim mogućnostima rada sa zapisima u tablici, odnosno za rad sa inventarom, kao što je vidljivo na *slici 8*.

Korisnik može:

1. Dodati novi proizvod u bazu podataka
2. Uređivati proizvod – mijenjati količine koje se nalaze na zalihama
3. Izbrisati proizvod iz baze podataka

5.4. Dodavanje proizvoda

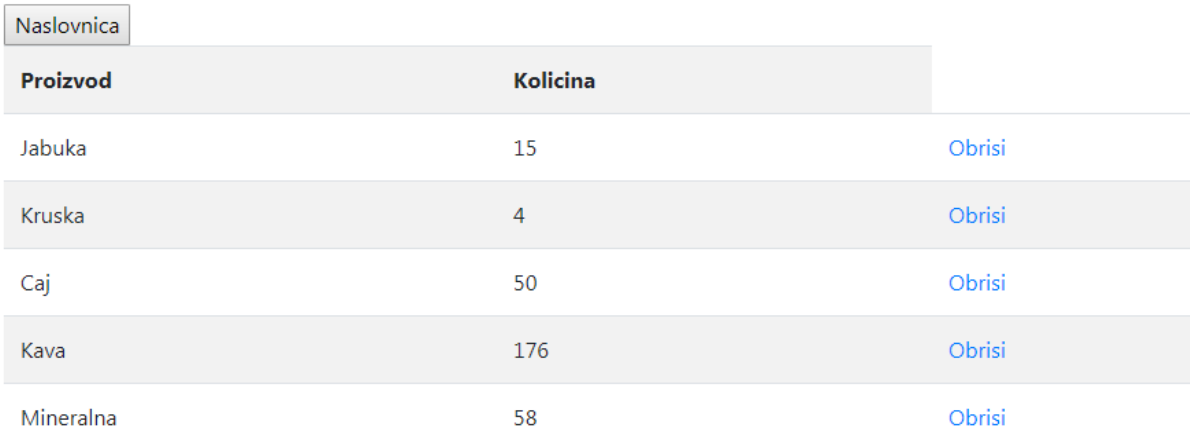
Ako korisnik želi dodati novi proizvod u bazu podataka, pritiskom na tipku „Dodavanje proizvoda“ dolazi na novi ekran gdje mu se nude polja za unos informacija o proizvodu, a to su „Ime proizvoda“ i „Količina“. Za polje „Količina“ je određeno da može biti nula, ako korisnik želi popuniti inventar s nadolazećim proizvodima, a kasnije možda za to ne bi imao vremena. Zapisi o proizvodima čija je količina nula prikazuju se na naslovnoj strani i moguće ih je urediti u svakom trenutku.

Kao što je vidljivo na *slici 9*., korisniku se nudi forma s dvama praznim poljima, gdje se upisuju ime proizvoda i količina. Ispunjavanjem potrebnih polja za dodavanje proizvoda i pritiskom na gumb „Dodaj“, u pozadini se učitava PHP datoteka koja vrijednosti zapisuje u bazu podataka s odgovarajućom oznakom lokacije.

Slika 9. Dodavanje novog proizvoda

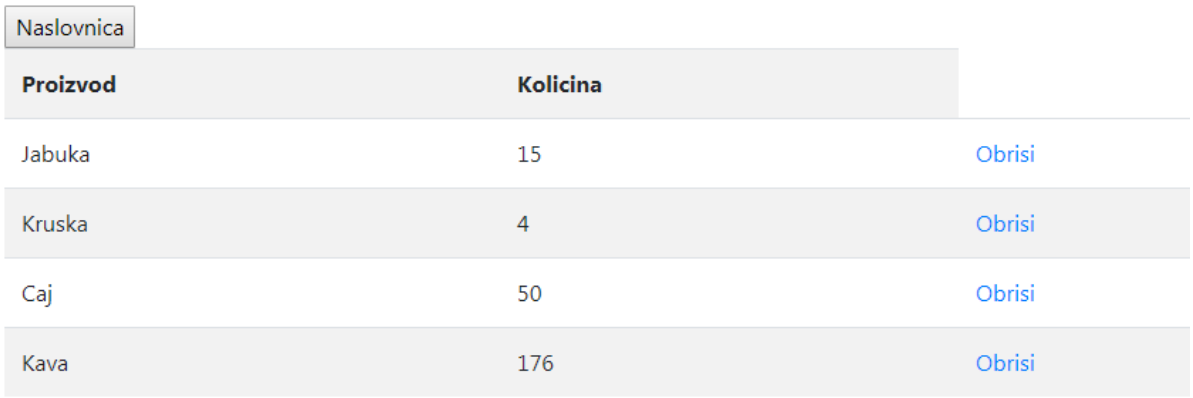
5.5. Brisanje proizvoda

Pritiskom na gumb „Brisanje proizvoda“ korisnik prelazi na drugi ekran gdje mu se pored svakog proizvoda nudi opcija brisanja istog, a iznad tablice se nalazi gumb za povratak na naslovnu stranu (*slika 10.*).



Slika 10. Brisanje proizvoda

Potvrdom brisanja proizvoda u pozadini se učitava druga PHP datoteka koja, nakon što je uspješno uklonila zapis iz baze podataka, korisnika vraća na isti ekran na kojemu se nalazio prije potvrde (*slika 11.*).

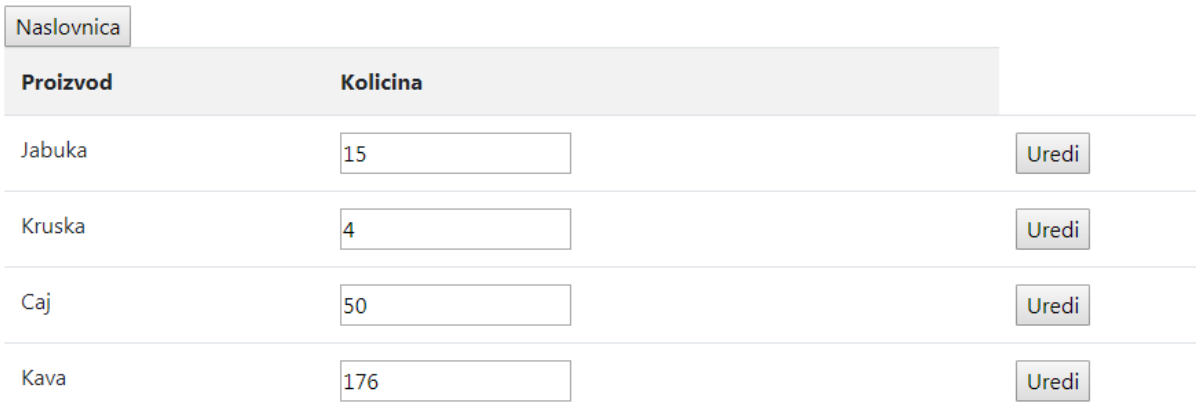


Slika 11. Rezultat brisanja proizvoda

5.6. Uređivanje proizvoda

Korisnicima je omogućeno mijenjati količine proizvoda koje se nalaze na skladištu.

Nakon što korisnik na svojoj naslovnoj strani odabere „Uređivanje proizvoda“ prelazi na novi ekran na kojemu se vrijednosti, koje se nalaze u stupcu „Količina“ sada nalaze u poljima koja je moguće uređivati (*slika 12.*).



Slika 12. Uređivanje proizvoda

Nakon što se izmjene potvrde, u pozadini se učitava druga PHP datoteka koja izmjene bilježi u bazi putem UPDATE upita te korisnika vraća na njegovu početnu stranicu.

5.7. Prednosti i nedostaci

Aplikacija korisnicima omogućava vođenje sveukupnog inventara na jednom mjestu na koje se može pristupiti u bilo kojem trenutku i s bilo koje lokacije. Ona nije ograničena prostorom i vremenom. Korisnik joj može pristupiti s bilo kojeg udaljenog mjesta, a za to mu je potrebna samo veza s internetom.

Veza s internetom je nedostatak ove aplikacija. Ako se dogodi da korisnik nije u mogućnosti spojiti se na internet, ne može doći do aplikacije, pregledavati ni uređivati ju, te se mora vratiti na stari način vođenja inventara. Sve ono što se izmijenilo u trenutku kada veza nije bila moguća, mora se unijeti naknadno kada se ona uspostavi.

Također, korisnici nisu u mogućnosti pratiti stanja prije izmjene, odnosno, u trenutku kada se inventar promijeni, ne bilježe se njegova stara stanja.

5.8. Buduća proširenja

Dodat će se podrška za praćenje stanja inventara od početka korištenja aplikacije, s datumom, vremenom i količinom. Povijesti će se moći pristupiti pritiskom na gumb gdje će zatim na novome ekranu biti prikazan sav inventar lokacije. Inventar će biti moguće ručno pretraživati ili odabrati vremenski okvir koji se želi pogledati.

Korisnicima će također biti pružena mogućnost eksporta povijesti lokacije u PDF ili Excel formatu.

6. Zaključak

Baze podataka postale su neizostavan dio svakog informacijskog sustava. One na jednom mjestu sadrže velik broj podataka i moguće im je pristupiti s bilo kojeg računala. Osim što omogućuju pregledavanje zapisa, omogućuju i njihovo uređivanje i brisanje. Nije im potreban velik fizički prostor i otvorene su za pristup u bilo koje vrijeme. Također, ne postoji jedan program kojim se može pristupiti bazi, već se za sve baze mogu raditi personalizirane aplikacije koje će na najjasniji način korisnicima prikazati tražene informacije.

Cilj rada bio je izraditi aplikaciju koja će korisnicima omogućiti da vode vlastiti inventar u elektroničkom obliku, što bi im omogućilo jednostavniji pregleda inventara pa bi se smanjila potreba za zapisivanjem proizvoda i količina na papir. Korisnicima je omogućeno da svoj inventar pregledavaju, uređuju postojeće proizvode i količine, dodaju nove proizvode ili brišu postojeće.

Za izradu aplikacije izrađena je baza podataka u koju se upisuju proizvodi i njihova količina. Razvijena je web aplikacija u programskom jeziku PHP, koja omogućuje izvršavanje radnji nad bazom podataka putem web sučelja, što znači da joj se može pristupiti s bilo kojeg mjesta i u bilo koje vrijeme. Naposljetku, korišten je HTML jezik za označavanje, kako bi se dodali gumbi, *login*-forma i tablice te kako bi se uljepšao prikaz korisničkog sučelja.

7. Literatura

1. ACP.NET MVC Overview. URL: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/overview/asp-net-mvc-overview (16.6.2019)
2. Alter table. URL: https://dev.mysql.com/doc/refman/5.5/en/alter-table.html (10.6.2019.)
3. Avraham, S.B., What is REST-A Simple Explanation for Beginners, Part 2: REST Constraints. URL: https://medium.com/extend/what-is-rest-a-simple-explanation-for-beginners-part-2-rest-constraints-129a4b69a582? (7.7.2019)
4. Bartlett, J., What is Grey-Box Testing? (2018). https://blog.testlodge.com/what-is-grey-box-testing/ (27.6.2019.)
5. Berg, Kristi & Seymour, Dr. Tom & Goel, Richa. (2012). History Of Databases. International Journal of Management & Information Systems (IJMIS). 17. 29. 10.19030/ijmis.v17i1.7587. URL https://www.researchgate.net/publication/298332910_History_Of_Databases (10.6.2019.)
6. Blackboard Design Pattern URL: https://social.technet.microsoft.com/wiki/contents/articles/13215.blackboard-design-pattern.aspx (3.7.2019)
7. Booleans. URL: https://www.php.net/manual/en/language.types.boolean.php (22.6.2019.)
8. Bray, T. On Semantics and Markup. URL: http://www.tbray.org/ongoing/When/200x/2003/04/09/SemanticMarkup#p-1 (24.6.2019.)
9. Chamberlin,D.D. et.al(1981) A History and Evaluation of System R. Computing Practices, 24(10) URL: https://people.eecs.berkeley.edu/~brewer/cs262/SystemR.pdf (10.6.2019.)
10. Classes, objects, methods and properties. URL: https://phpenthusiast.com/object-oriented-php-tutorials/create-classes-and-objects (23.6.2019)
11. Client-server architecture. URL: https://www.britannica.com/technology/client-server-architecture (30.8.2019.)

12. Codd, E.F. (1985) Is your DBMS really relational?. Computerworld, 19/1985 (41). URL: https://archive.org/details/computerworld1940unse_0/page/n233 (6.9.2019)
13. Codd's Rule in DBMS. URL: https://www.tutorialcup.com/dbms/codds-rule.htm(10.6.2019)
14. Codd's Rules for RDBMS. URL: https://www.w3resource.com/sql/sql-basic/codd-12-rule-relation.php(10.6.2019)
15. Database Testing: How to Regression Test a Relational Database. Dostupno na: http://www.agiledata.org/essays/databaseTesting.html#HowToTest (29.8.2019.)
16. Date, C.J. (2004) An Introduction to Database Systems. 8. izd. SAD: Pearson Education International
17. Donald Chamberlin. Dostupno na: https://www.computerhistory.org/fellowawards/hall/donald-chamberlin/ (29.8.2019)
18. Entitiy Systems are the future of MMOG development-Part 2 (2007) URL:http://t-machine.org/index.php/2007/11/11/entity-systems-are-the-future-of-mmog-development-part-2/ (4.7.2019)
19. Float. URL: https://www.php.net/manual/en/language.types.float.php (22.6.2019)
20. Foote, K.D., A Brief History of Database Management (2017). URL: https://www.dataversity.net/brief-history-database-management/(10.6.2019)
21. FundingUniverse. URL: http://www.fundinguniverse.com/company-histories/macromedia-inc-history/ (30.8.2019)
22. George Boole. URL: https://www.britannica.com/biography/George-Boole (30.8.2019)
23. GOPHP5. URL: https://web.archive.org/web/20110720002753/http://www.gophp5.org/faq (20.6.2019.)
24. Haerder, T.; Reuter, A.(1983), Principles of Transaction-Oriented Database Recovery. Computing Surveys, 15 (4). Dostupno na: https://sites.fas.harvard.edu/~cs265/papers/haerder-1983.pdf (30.8.2019)
25. History of PHP. URL: https://www.php.net/manual/en/history.php.php (18.6.2019.)

26. HTML Input Types. URL: https://www.w3schools.com/html/html_form_input_types.asp (2.7.2019)
27. Hypertext Transfer Protocol (HTTP). URL: https://www.techopedia.com/definition/2336/hypertext-transfer-protocol-http (26.6.2019)
28. IBM Knowledge Center. URL: https://www.ibm.com/support/knowledgecenter/SSGU8G_12.1.0/com.ibm.sqlr.doc/ids_sqr_010.htm (29.8.2019.)
29. Integers. URL: https://www.php.net/manual/en/language.types.integer.php (22.6.2019.)
30. Iterable. URL: https://www.php.net/manual/en/language.types.iterable.php (23.6.2019)
31. Kaur, T.; Sehra, S.K. (2015) Designing and Development of Database Testing Tool. Internation Journal of Computer Applications, 120. https://pdfs.semanticscholar.org/c924/8e3e7cb7451b17dbaa647a7aa9428094356b.pdf (27.6.2019.)
32. Kramer. O.; Sturm. A. (2005)., The Relationships between Domain Specific and GeneralPurpose Languages, URL: https://pdfs.semanticscholar.org/83a2/a44fb9a5329442b135920a222bd5da5c75bb.pdf (30.8.2019.)
33. Kreibich, J.A.,(2010) Using SQLite: Transaction Control Language. URL: https://www.oreilly.com/library/view/using-sqlite/9781449394592/ch04s07.html (12.6.2019.)
34. Lerdorf, R.(1995) Personal Home Page Tools (PHP Tools). URL: https://groups.google.com/forum/#!msg/comp.infosystems.www.authoring.cgi/PyJ25gZ6z7A/M9FkTUVDfcwJ (18.6.2019.)
35. Lerdorf, R., SPEEDING UP THE WEB WITH PHP 7(2015), URL: http://talks.php.net/fluent15#/ (21.6.2019.)
36. Coombs, J.H., Renear, A. H., DeRose, S. J., Markup Systems and the Future of Scholarly Text Processing, URL: http://xml.coverpages.org/coombs.html (31.8.2019.)
37. Michelson, B.M. Event-Driven Architecture Overwiev. URL: https://www.omg.org/soa/Uploaded%20Docs/EDA/bda2-2-06cc.pdf (6.7.2019)

38. National Research Council. 1999. Funding a Revolution: Government Support for Computing Research. Washington, DC: The National Academies Press. https://doi.org/10.17226/6323. https://www.nap.edu/read/6323/chapter/8#165 (10.6.2019.)
39. Netscape. URL. https://web.archive.org/web/20070916144913/http://wp.netscape.com/newsref/pr/newsrelease67.html (30.8.2019)
40. Novoseltseva, E., (2017) Benefits of Software Architecture. URL: https://dzone.com/articles/15-benefits-of-software-architecture (30.8.2019)
41. NULL. URL: https://www.php.net/manual/en/language.types.null.php (23.6.2019)
42. Package Statistics. URL: https://pear.php.net/package-stats.php (23.6.2019.)
43. Perry, D.E.; Wolf, A.L.(1992) Foundations for the Study of Software Architecture. SOFTWARE ENGINEERING NOTES, 17 (4), str. 40 URL: http://users.ece.utexas.edu/~perry/work/papers/swa-sen.pdf (3.7.2019)
44. PSR-1: Basic Coding Standard. URL: https://www.php-fig.org/psr/psr-1/#21-php-tags (22.6.2019.)
45. Resources. URL: https://www.php.net/manual/en/language.types.resource.php (23.6.2019)
46. Rohrmann, T., Introducing Complex Event Processing (CEP) with Apache Flink. URL: https://flink.apache.org/news/2016/04/06/cep-monitoring.html (6.7.2019)
47. Service Oriented Architecture: What is SOA. URL: https://web.archive.org/web/20160819141303/http://opengroup.org/soa/source-book/soa/soa.htm (4.7.2019)
48. SOA Manifesto. URL: http://www.soa-manifesto.org/ (4.7.2019)
49. SOA (Service Oriented Architecture) Principles. URL: https://www.guru99.com/soa-principles.html (5.7.2019)
50. Software Architecture. URL: https://www.sei.cmu.edu/research-capabilities/all-work/display.cfm?customel_datapageid_4050=21328 (3.7.2019)
51. Wikipedia Contributors (2019). Software architecture. URL: https://en.wikipedia.org/wiki/Software_architecture (3.7.2019)
52. W3C., A History of HTML.(1998) URL: https://www.w3.org/People/Raggett/book4/ch02.html (31.8.2019)

53. W3C. URL: https://www.w3.org/2014/10/html5-rec.html.en (30.8.2019)
54. String. URL: https://www.php.net/manual/en/language.types.string.php (22.6.2019)
55. TCP/IP Protocol Architecture Model. URL: https://docs.oracle.com/cd/E19683-01/806-4075/ipov-10/index.html (26.6.2019)
56. Using PEPr. URL: https://pear.php.net/manual/en/newmaint.proposal.step3.php (23.6.2019.)
57. Watts, S., What is Stream Processing? Event Stream Processing Explained. URL: https://www.bmc.com/blogs/event-stream-processing/ (6.7.2019)
58. What is MVC?. URL: https://www.guru99.com/mvc-tutorial.html (17.6.2019.)
59. What is PEAR?. URL: https://pear.php.net/manual/en/about.pear.php (23.6.2019.)
60. What is REST?. URL: https://www.codecademy.com/articles/what-is-rest (7.7.2019)
61. What is Service-Oriented Architecture?- URL: https://medium.com/@SoftwareDevelopmentCommunity/what-is-service-oriented-architecture-fa894d11a7ec (5.7.2019)
62. Zend API: Hacking the Core of PHP. URL: https://www.php.net/manual/en/internals2.ze1.zendapi.php (30.8.2019.)

Web aplikacija za upravljanje inventarom

Sažetak

Rad se temelji na tehnologijama koje su korištene za razvijanje web aplikacije za upravljanje inventarom. Objašnjene su baze podataka koje su osnova cijele aplikacije i SQL, koji u pozadini izvodi tražene radnje s bazom. Opisani su osnovni upiti i načini na koje se mogu slagati.

PHP-u, kao programskom jeziku kojim je pisana aplikacija, opisana je povijest te je također opisana njegova osnovna sintaksa. U sklopu PHP-a nalazi se i HTML koji je korišten kako bi se uredio sami prikaz aplikacije u pregledniku. Navedene su oznake s primjerima koje se u HTML-u koriste za oblikovanje teksta i prikaza.

Aplikacija je razvijena sa ciljem da se smanji ovisnost korisnika o vođenju cijeloga inventara na papiru. Može joj se pristupiti putem webu i od korisnika ne zahtijeva instaliranje dodatnih programa na računalo ili pametne telefone. Vođenje je preneseno u elektronički oblik, što korisniku omogućuje pristup u bilo koje vrijeme i s bilo kojeg mjesta.

Ključne riječi: SQL, PHP, MVC, baze podataka, web aplikacija

Inventory management web application

Summary

This thesis covers the technologies used to develop an inventory management web application. It describes the database at the core of the application and the SQL query language used to communicate with the database. Basic queries and their use are explained.

The thesis also covers the history and basic syntax of the PHP programming language. The web page is made using HTML. Examples of HTML tags used for text and other visual content are given.

The thesis includes an inventory management web application. The goal of the application is eliminating the need of manual, paper-based record keeping. The application is web based and does not require users to install additional programs on their computer or smartphones Inventory can be managed online and accessed anywhere and anytime.

Key words: SQL, PHP, MVC, database, web application