Web sustav upravljanja bazom podataka MySQL

Topolovec, Filip

Sveučilište
Sjever

Završni rad br. 451/MM/2015

Web sustav upravljanja bazom podataka MySQL

Filip Topolovec, 4705/601

Varaždin, listopad 2015. godine

Sveučilište
Sjever

Odjel za multimediju, oblikovanje i primjenu

Završni rad br. 451/MM/2015

Web sustav upravljanja bazom podataka MySQL

Student

Filip Topolovec, 4705/601

Mentor

Vladimir Stanisavljević, mr.sc.

Varaždin, listopad 2015. godine

Sažetak

U ovom radu obrađuju se tehnologije potrebne za dinamično upravljanje MySQL bazom podataka: PHP, SQL, PDO, AJAX i jQuery. PHP čini temeljnu programsku arhitekturu, u SQL-u se strukturiraju upiti za komunikaciju sa bazom, PDO izvršava te upite, a AJAX i jQuery su odgovorni za dinamično upravljanje bazom bez ponovnog učitavanja čitavog sadržaja nakon neke akcije.

Kao praktični dio završnog rada izrađen je primjer sustava za dinamično upravljanje bazama podataka korištenjem navedenih tehnologija sa osnovnim CRUD funkcionalnostima (engl. *create, read, update, delete*).

Popis korištenih kratica

engl.
Engleski

DBMS
Sustav za upravljanje bazama podataka (engl. Database Managament System)

OOP
Objektno orijentirano programiranje (engl. Object Oriented Programming)

PHP
U početku Personal Home Page, kasnije PHP: Hypertext Preprocessor

MVC
engl. Model-View-Controller

PDO
PHP objekti podataka (engl. PHP Data Objects)

AJAX
Asinhroni JavaScript i XML (engl. Asynchronous JavaScript and XML)

SQL
Strukturirani upitni jezik (engl. Structured Query Langauge)

CRUD
Funkcionalnosti izrade, čitanja, ažuriranja i brisanja

HTTP
engl. HyperText Transfer Protocol

HTML
Prezentacijski jezik za izradu web stranica (engl. HyperText Markup Language)

CSS
Kaskadne stranice stilova (engl. Cascading Style Sheets)

XML
Jezik za proširivo označavanje podataka (engl. Extensible Markup Language)

URL
jedinstveni lokator resursa (engl. Uniform Resource Locator)

Sadržaj

1. Uvod

Broj korisnika na internetu neprekidno raste iz dana u dan te je premašio tri milijarde korisnika još 2014. godine [1]. Upravo tom činjenicom potaknuta je izrada završnog rada o aplikaciji za dinamično upravljanje bazama podataka jer povećanjem broja korisnika povećava se i količina informacija na internetu što zahtjeva razvoj sustava za upravljanje bazama podataka.

Internetska baza podataka skup je organiziranih i međusobno povezanih podataka smještenih na web poslužitelju [2]. Podacima na web poslužitelju pristupa se preko tzv. DBMS (engl. *Database Managament System*) sustava. Najpopularniji poslužitelj baza podataka je MySQL kojim se najčešće komunicira PHP programskim jezikom. Razlog njihove popularnosti je u besplatnom korištenju i razvoju (engl. *Open Source*). Kada tim tehnologijama dodamo JavaScript i CSS sa namjerom poboljšanja korisničkog doživljaja dobivamo sve komponente potrebne za razvoj aplikacije za dinamično upravljanje bazama podataka.

U prvom dijelu rada teoretski će se obraditi tehnologije PHP, MySQL i JavaScript, uz njih usko vezane tehnologije AJAX i jQuery, razvojni okvir (engl. *Framework*) PHP MINI, i objektno orijentirano programiranje.

U drugom dijelu rada opisati će se izrada aplikacije za dinamično upravljanje bazama podataka korištenjem svih navedenih tehnologija u teoretskom dijelu rada. Aplikacija bi trebala izvršavati osnovne CRUD funkcionalnosti (engl. *Create, read, update, delete*) nad MySQL bazom podataka korištenjem PHP programskog jezika sa naglaskom na dinamičnost.

Ključne riječi u završnom radu: **PHP, MySQL, JavaScript, AJAX, jQuery, PHP MINI.**

2. PHP (PHP: Hypertext Preprocessor)

PHP je interpreterski programski jezik koji se obrađuje na poslužitelju. Sintaksa PHP programskog jezika orijentirana je prema sintaksi programskih jezika C i Perl [3]. PHP je namijenjen isključivo izradi dinamičnih web stranica i aplikacija. Ne koristi se za programiranje desktop aplikacija namijenjenih jednom korisniku, već se PHP kod obrađuje na poslužitelju, a korisnici poslužitelju pristupaju putem web preglednika, takav način komunikacije nazivamo **klijentsko-poslužiteljska arhitektura** (engl. *Client-server architecture*) [4].

PHP se ističe širokom podrškom raznih baza podataka i internet protokola kao i raspoloživosti brojnih programerskih knjižnica. Važno je naglasiti da se radi o proizvodu otvorenog koda (engl. *open source*), što znači da postoji pristup izvornom kodu kojeg je moguće koristiti, mijenjati i dalje distribuirati potpuno besplatno. U praktičnom dijelu rada PHP-om će biti izrađena temeljna programska arhitektura.

2.1. Povijest PHP-a

PHP kakvog danas poznajemo zapravo je nasljednik proizvoda zvanog PHP/FI. kojeg je 1994. godine konstruirao Grenlandski programer Rasmus Lerdorf. Inicijalna verzija PHP-a upotrebljavala se u počecima kao brojač posjeta, a paket skripti R. Lerdorf nazvao je "Personal Home Page Tools" [5].

S vremenom se pojavila potreba za većom funkcionalnošću pa je R. Lerdorf reprogramirao PHP alate i time omogućio širu i bogatiju implementaciju PHP alata. Novi model alata bio je sposoban spojiti se i komunicirati sa bazom podataka te se koristio kao razvojni okvir (engl. *Framework*) na kojem su korisnici razvijali dinamične web stranice.

U lipnju 1995. Godine R. Lerdorf izdao je izvorni PHP kod za javnost čime je potaknuo programere na slobodno korištenje, ispravljanje grešaka i u kranjem

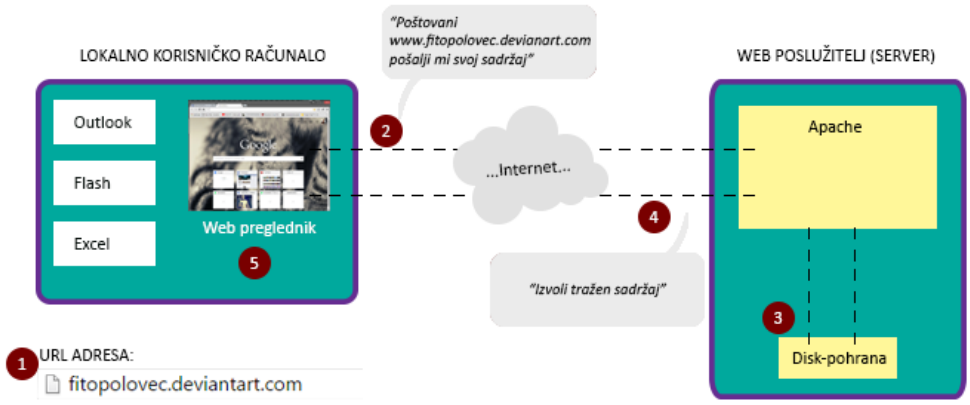
slučaju razvoj programskog jezika koji je danas temelj brojnih dinamičkih web sjedišta i aplikacija [5].

PHP 2.0	1997. godine, ugrađena podrška za DBM, mSQL i Postgres95 baze podataka, podrška za kolačiće i korisnička podrška
PHP 3.0	1998. godine, umjesto PHP/FI preimenovan u PHP **(PHP: Hypertext Preprocessor)**, unaprijeđena sintaksa, podržava objektno orijentirano programiranje, spoj na više baza itd.
PHP 4.0	2000. godine, poboljšano izvođenje koda (performance), ugrađena podrška za veći broj servera, HTTP sesija, sigurnosna poboljšanja, dodani neki novi jezični konstrukti.
PHP 5.0	2004. godine, srž ove verzije PHP-a je Zend Engine 2.0, dodane mnoge nove funkcije, konstantno u razvoju, trenutna inačica je PHP 5.6.7.

Tablica 1. Inačice PHP programskog jezika [5]

2.2. **Klijentsko-poslužiteljska arhitektura**

Kada sjedimo ispred kompjutera i surfamo internetom nekim web preglednikom kao što su Google Chrome ili Mozilla Firefox, potičemo razne komunikacijske procese koji se zbivaju između našeg računala i udaljenog računala koje nam pruža sadržaj koji pregledavamo. Navedena komunikacija i njezin rezultat ilustrirani su na *Slika 1.*



Slika 1. Klijentsko-poslužiteljska arhitektura

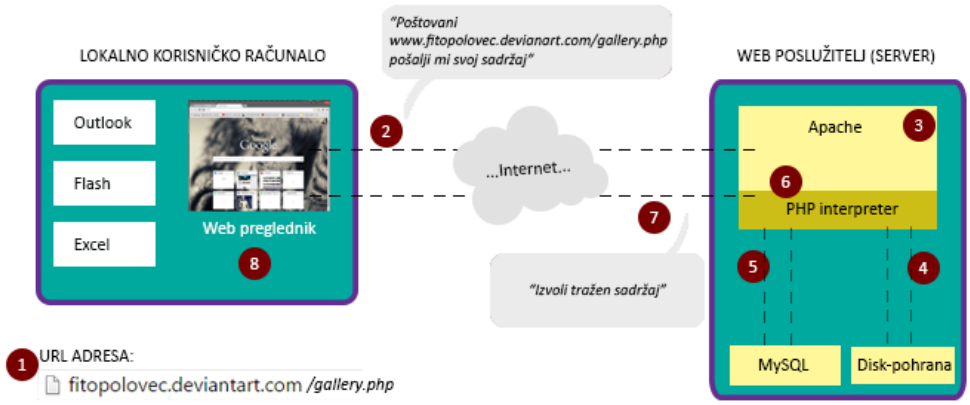
Na pobrojanim mjestima dijagrama događa se sljedeće:

1. Upišemo u adresnu traku URL adresu [1]stranice koju želimo posjetiti.
2. Web preglednik pošalje poruku preko Interneta računalu na adresi fitopolovec.deviantart.com i traži njen sadržaj.
3. Apache[2], poslužiteljski program, pročita poruku i izvadi tražen sadržaj sa diska na kojem je sadržaj pohranjen.
4. Apache šalje povratni sadržaj preko Interneta našem lokalnom korisničkom računalu kao odgovor na pitanje preglednika.
5. Web preglednik prikazuje stranicu na ekranu na način da renderira HTML kod koji je stigao u poruci.

[1] URL adresa je jedinstveni lokator resursa (engl. Uniform Resource Locator)

[2] Apache je besplatni HTTP web poslužitelj

Potrebno je naglasiti da se u prethodnom primjeru radi o dohvaćanju *.html* datoteka. Ukoliko sa poslužitelja dohvaćamo PHP sadržaj, poslužitelj ima nešto više posla za odraditi prije nego nam pošalje odgovor. Pa to izgleda ovako:



Slika 2. Klijentsko-poslužiteljska arhitektura uz PHP

1. Upišemo u adresnu traku URL adresu stranice koju želimo posjetiti.
2. Web preglednik pošalje poruku preko Interneta računalu na adresi fitopolovec.deviantart.com i traži sadržaj *gallery.php.*
3. Apache, poslužiteljski program, pročita poruku i proslijedi ju PHP interpreteru sa pitanjem „Kako izgleda *.../gallery.php?"*.
4. PHP interpreter [3]čita datoteku */usr/local/www/devians/gallery.php* sa diska na poslužitelju.
5. PHP interpreter izvršava naredbe u *gallery.php* datoteci, dok izvršava naredbe moguće je da komunicira sa MySQL bazom podataka.

[3] PHP interpreter je poslužiteljski program koji prati i izvršava naredbe napisane u PHP programskom jeziku.

6. PHP interpreter uzima vraćeni rezultat i šalje ga na Apache kao odgovor na „Kako izgleda *.../gallery.php?"*.
7. Apache šalje povrati sadržaj PHP interpretera preko Interneta našem lokalnom računalu kao odgovor na pitanje preglednika.
8. Web preglednik prikazuje stranicu na ekranu na način da renderira HTML kod koji je stigao u poruci.

PHP se izvršava na poslužitelju kao što je prikazano na *Slika 2*. Programski jezici i tehnologije kao što su JavaScript, jQuery ili Flash izvršavaju se na klijentskoj strani (npr. prijenosno računalo ili mobitel) [2]. Kod izvršavanja PHP naredbi, PHP interpreter vraća rezultat koji se zatim šalje putem Interneta klijentu i prikazuje na ekranu. Sa druge strane, klijentska tehnologija kao što je JavaScript daje svoje naredbe direktno lokalnom web pregledniku koji zatim te naredbe izvršava, na primjer otvaranje skočnog prozora.

Upravo takav primjer klijentsko-poslužiteljske arhitekture koristiti će se u praktičnom dijelu završnog rada kod dinamične komunikacije korisnika sa MySQL bazom podataka.

2.3. **Objektno orijentirano programiranje (OOP) u PHP-u**

Objektno orijentirano programiranje je programska paradigma, odnosno način/stil programiranja [6]. Postoje razni načini/stilovi programiranja: strukturalno, nestrukturalno, imperativno, deklarativno, proceduralno, funkcionalno itd.

U PHP programskom jeziku često se koristi proceduralno programiranje. Mana proceduralnog programiranja je odvojenost ponašanja od podataka što znači da se strukture koriste za sadržavanje podataka, a procedure za rad nad tim podacima [7]. Zbog te odvojenosti iznimno je teško održavati i graditi velike sustave.

2.4. Povijesni razvoj OOP – pojam klase

U prošlom stoljeću software-ski inženjeri zaključili su da je moguća kombinacija tehnologija izrade software-a i napretka u programskim jezicima. Rezultat te spoznaje u programskim jezicima bio je pojam klase (engl. *class*) [8]. Klasa implementira koncepte apstrakcije, modularizacije i skrivanja podataka. To radi na način da grupira definirane tipove podataka, zajedno s procedurama i funkcijama koje se mogu primijeniti na njih. Klasu je moguće naslijediti, (engl. *inheritance*) što omogućava ponovno iskorištavanje već postojećeg koda (engl. *reuesment*), klasa i struktura podataka u klasama [6].

»Klasa je u programskim jezicima napravila revoluciju sličnu onoj koju je u elektronici napravio razvoj integriranih krugova. Kao što se elektronički sustavi mogu izrađivati korištenjem već postojećih IC-ova, te po potrebi njihovim modificiranjem, tako se i u izradi software-skih sustava može raditi korištenjem ili modifikacijom klasa. « [8]

Prvi objektno orijentirani jezik bio je SIMULA, razvijen sredinom i kasnih '60-ih godina u Norveškoj [9]. SMALLTALK je programski jezik razvijen početkom '70-ih godina, a popularizirao je pojam objektno orijentiranog programiranja. Koristio se uvelike kod istraživanja umjetne inteligencije.

Sedamdesetih godina prošlog stoljeća ovi jezici su bili dostupni isključivo istraživačkim laboratorijima. Tek početkom '80-ih godina javlja se pojam komercijalnih programa. Tada je javnosti predstavljen SMALLTALK-80. Ostali objektno orijentirani jezici koji su postali komercijalno dostupni: C++, Objective C, Object Pascal itd. Među njima C++ proizlazi kao dominantni objektno orijentirani programski jezik [9].

Ključni koncepti objektno orijentiranog programskog jezika [8]:

- baziranost na objektima i klasama
- podržavanje nasljeđivanja
- podržavanje polimorfizma

2.5. **Osnovni pojmovi OOP u PHP-u**

2.5.1. **Objekt**

Osnovu objektno orijentiranog programiranja čini objekt. Objekt je u svakodnevnom svijetu definiran kao „nešto“ s granicama. On objedinjava podatke (engl. *properties*) i funkcijske članove ili metode (engl. *methods*). Podatkovni članovi opisuju stanje objekta [8].

Objekt prima poruke iz okoline i izvršava jednu od svojih metoda kao rezultat prijema poruke iz okoline (okolina može biti recimo neki drugi objekt). Poruka kaže što objekt treba napraviti, dok metoda kaže kako to treba napraviti [6]. Dakle, objektno orijentirana aplikacija je skup objekata koji rade zajedno radi postizanja ukupnog cilja. U praktičnom dijelu završnog rada aplikacija će vraćati tražene podatke iz baze u obliku objekta.

2.5.2. **Klasa**

Pomoću klasa se u objektno orijentiranim jezicima opisuju objekti. Klasa definira objekte i metode pojedinih objekata. Programski jezici trebaju omogućiti kreiranje objekata iz njihovih klasa. Klasa mora omogućiti rukovanje memorijom (engl. *memory-management*), alokaciju i dislokaciju memorijskog prostora objekta, oslobađajući programera tog zadatka [8].

Važno je da klasa postane strukturalni entitet za sve programe. Opis klase djeluje kao modul koji omogućava podjelu programa u manje dijelove. Podjela programa u manje dijelove biti će vidljiva u praktičnom dijelu završnog rada na primjerima CRUD funkcionalnosti.

2.5.3. **Metoda**

Metode objekata imaju pristup podatkovnim članovima objekata, tj. međusobno dijele podatke objekata [10]. Pomoću metoda u praktičnom dijelu završnog rada dohvaćat će se i manipulirati podacima.

2.6. **Osnovni principi OOP u PHP-u**

2.6.1. **Apstrakcija**

Sastavni dijelovi programa su objekti, koji su instance nekih klasa. Da bi se riješio programski zadatak potrebno je modelirati klase i njihove međusobne interakcije. Na taj način se sav kod u objektno orijentiranom programiranju nalazi unutar jedne ili više klasa.

2.6.2. **Enkapsulacija**

Važan princip OOP u PHP-u je postupak objedinjavanja stanja i ponašanja objekta u jednu cjelinu. Objekte organizirane na taj način lako je kontrolirati i onemogućiti neovlašten pristup. Korisnici sa objektom komuniciraju kontrolirano koristeći javne metode i ne mogu neovlašteno mijenjati unutarnja stanja objekta. Samo unutarnjim metodama objekta omogućen je pristup tim stanjima [10].

2.6.3. **Nasljeđivanje**

Nasljeđivanje je karakteristika objektno orijentiranog programiranja koja omogućuje izvođenje novih klasa iz postojećih i njihovu izmjenu. Samim tim povećava se produktivnost jer se koriste već postojeće biblioteke programskog koda.

Klase i njihove pod klase formiraju hijerarhiju klasa. Instanca izvedene klase je isto tako instanca svih njenih super-klasa (klasa koje su više po hijerarhiji). [10]

2.6.4. **Polimorfizam**

»Polimorfizam u prijevodu znači „moći poprimiti više oblika“, također je važna karakteristika objektno orijentiranog programiranja. Polimorfizam je baziran na pretpostavci da tip ili klasa varijable ne mora biti ista tipu ili klasi objekta na koju se varijabla odnosi. Dok sam objekt ne može biti polimorfan, varijabla koja ga predstavlja i koja može pokazivati na sam objekt može imati više tipova. Varijabla deklarirana da bude određene klase može se odnositi na objekt upravo te klase, ili objekt bilo koje od njenih izvedenih pod klasa.

Drugim riječima klasa varijable deklarira minimum zahtjeva koje objekt može imati. Polimorfizam omogućava specificiranje algoritama na višem, apstraktnijem nivou. Tipičan primjer je algoritam za printanje elemenata u listi. Kod polimorfizma algoritam može biti izveden na višem, apstraktom nivou: „pretraži listu i pošalji poruku za printanje svakom elementu“. Lista može sadržavati objekte različitih klasa, bitno je samo da svaka od tih klasa ima definiranu print metodu. « [11]

2.6.5. **Modularnost**

»Modularnost se odnosi na postupak razbijanja programa na manje dijelove koji mogu autonomno funkcionirati. Modularnost se primjenjuje kako bi se omogućila višestruka upotreba softverskih komponenti, odnosno takozvana ponovna upotreba koda (engl. *code reuse*). « [11]

2.7. Sintaksa objektno orijentiranog programiranja u PHP-u

```
class moja_klasa {
	$atribut = 'vrijednost';
	function moja_metoda() {
	echo 'zivot je generalno ljep';
	}
}
$moj_objekt = new moja_klasa();
$moj_objekt->moja_metoda();
echo $moj_objekt->atribut;
```

Kod 1. Sintaksa u OOP u PHP-u

U nastavku slijedi objašnjenje sintakse OOP (objektno orijentirano programiranje) u PHP-u na primjeru *Kod 1.***:**

- Ključnom riječi *class* počinje deklaracija klase.
- Tijelo deklaracije nalazi se unutar vitičastih zagrada.
- Atributi klase se navode tako da se nanižu njihova imena, može im se odmah postaviti i vrijednost kao što smo napravili u gornjem primjeru.
- Metode se pišu kao što se pišu funkcije u klasičnom PHP kodu samo što se ovdje pišu unutar deklaracije klase.
- Nakon što smo deklarirali klasu kreiramo objekt tipa *moja_klasa* sa ključnom riječi new.
- Liniju ispod pozivamo metodu *moja_metoda* na objektu *moj_objekt*.
- Linija ispod je ispis vrijednosti atributa po imenu atribut.
- Kada ovako pojednostavnjeno gledamo, objekt je u biti spremište varijabli i funkcija.

3. MySQL

MySQL je poslužitelj baza podataka (engl. *database server*). Drugim riječima, radi se o softwareu kojem se može pristupiti preko mreže na sličan način kao i web (HTTP) poslužiteljima, sa tom razlikom da se MySQL-u obično pristupa pomoću korisničkog imena i lozinke [12]. MySQL ima dobro dokumentirane module i ekstenzije te podršku od brojnih programskih jezika: PHP, Java , Perl, Python itd.

MySQL baze su relacijskog tipa, koji se pokazao kao najbolji način skladištenja i pretraživanja velikih količina podataka i u suštini predstavljaju osnovu svakog informacijskog sustava, tj. temelj svakog poslovnog subjekta koji svoje poslovanje bazira na dostupnosti kvalitetnih i brzih informacija.

Na serveru može postojati veći broj baza podataka koje su potpuno samostalne, no unutar jednog projekta se može baratati podacima iz više baza na serveru. Svakom korisničkom računu na serveru je moguće dodijeliti razna administracijska prava na cijeli server ili pojedine baze. Neka od prava bi bila stvaranje novih baza, pravo pristupa postojećim bazama, pravo uređivanja (unosa ili izmjena podataka) postojećih baza itd. Pri instalaciji MySQL-a se stvara tzv. superadministartor (obično se zove *root*) koji ima sva administracijska prava [12].

Jedna od velikih prednosti MySQL-a je što postoje verzije za sve važnije operacijske sustave, te ih se distribuira pod GPL licencom (engl. *General Public Licence*). Poveznica na više informacija o GPL licenci nalazi se ovdje http://dokumentacija.linux.hr/GPL.html. U poglavlju 5.1 opisan je postupak izrade MySQL baze podataka za praktični dio rada.

Postoje tri osnovne vrste podataka:

- tekstualni
- numerički
- datum/vrijeme

Tekstualni tipovi podataka	
CHAR( )	String fiksne dužine od 0 do 255 znakova.
VARCHAR( )	String promjenjive dužine od 0 do 255 znakova.
TINYTEXT	String maksimalne dužine 255 znakova.
TEXT	String maksimalne dužine 65535 znakova.
BLOB	String maksimalne dužine 65535 znakova.
MEDIUMTEXT	String maksimalne dužine 16777215 znakova.
MEDIUMBLOB	String maksimalne dužine 16777215 znakova.
LONGTEXT	String maksimalne dužine 4294967295 znakova.
LONGBLOB	String maksimalne dužine 4294967295 znakova.
Numerički tipovi	
TINYINT( )	-128 do 127 normal
	0 do 255 UNSIGNED
SMALLINT( )	-32768 do 32767 normal
	0 do 65535 UNSIGNED
MEDIUMINT( )	-8388608 do 8388607 normal
	0 do 16777215 UNSIGNED
INT( )	-2147483648 do 2147483647 normal
	0 do 4294967295 UNSIGNED
BIGINT( )	-9223372036854775808 do 9223372036854775807 normal
	0 do 18446744073709551615 UNSIGNED
FLOAT	Manji broj s pomičnim zarezom.
DOUBLE(,)	Veći broj s pomičnim zarezom.
DECIMAL(,)	DOUBLE spremljen kao string i to točno određene dužine..
Datum i vrijeme	
DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DD HH:MM:SS
TIMESTAMP	YYYYMMDDHHMMSS
TIME	HH:MM:SS
Ostali tipovi	
ENUM ( )	do 65535 znakova

	Svojevrsno nabrajanje, što znači da svaki ovakav stupac može imati samo jednu od prethodno definiranih vrijednosti
SET	do 64 znaka Sličan ENUM tipu podatka no svaki supac može sadržavati više prethodno definiranih vrijednosti

Tablica 2. MySQL tipovi podataka [13]

4. PHP MINI razvojni okvir i povezane tehnologije

PHP MINI je razvojni okvir (engl. *framework*) za PHP aplikacije baziran na MVC (*Model, View, Controller*) arhitekturi. Prednosti PHP MINI razvojnog okvira naspram ostalih profesionalnih razvojnih okvira su u njegovoj čistoj i jasnoj strukturi, čistim URL-ovima, ugrađenim primjerima za CRUD (*Create, Read, Update, Delete*) i AJAX funkcionalnosti, komentiranim kodom, primjerima PDO (*PHP Data Objects*) spojeva na bazu, i korištenjem izvornog objektno orijentiranog PHP koda. Iz tog razloga korisnici ne moraju učiti nikakve posebne funkcionalnosti samog razvojnog okvira.

PHP MINI jednostavan je za instalaciju. Komprimirani paket ovog razvojnog okvira dostupan je na GitHub-u te je besplatan i slobodan za korištenje. Nakon preuzimanja potrebno ga je postaviti na poslužitelj koji podržava PHP 5.3.0+ i MySQL i konfigurirati *config.php* datoteku sukladno podacima sa poslužitelja. [18]

4.1. Kako PHP MINI funkcionira?

Da bi objasnili kako ovaj razvojni okvir funkcionira potrebno je pogledati URL-ove standardne instalacije MINI razvojnog okvira. Recimo da smo ga instalirali u direktorij primjer.com. MINI koristi *mod_rewrite* modul kako bi kreirao i čitao jednostavne i uredne URL-ove koji izgledaju otprilike ovako:

http://www.example.com/home/exampleone

ili

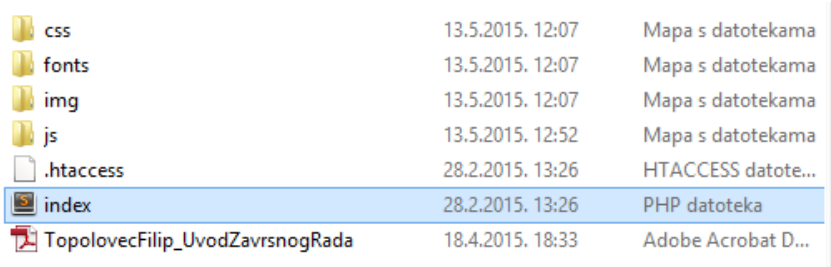
http://www.example.com/songs/editsong/17.

Detaljno u *mod_rewrite* modul nećemo ulaziti, ali *songs/editsong/17* je dio URL-a proslijeđen na index.php kao GET parametar na način *index.php?url=songs/editsong/17* i zatim razdijeljen po znaku kosoj crti (engl. „*slash*“). Nakon dijeljenja URL-a dobili smo elemente „songs“, “editsong“ i „17“ koji su potom proslijeđeni u aplikacijsku strukturu. Prvi element uvijek označava

ime kontrolera, u ovom slučaju bi to bio kontroler „songs“, dakle MINI učita *aplication/controllers/songs.php*. Drugi element uvijek označava ime akcije, koja je zapravo metoda unutar kontrolera, u našem slučaju to je „editsong“, dakle MINI poziva *editSong()* unutar *aplication/controllers/songs.php*. Svi sljedeći elementi URL-a predstavljaju parametre metodi *editSong()*, u našem slučaju to je broj „17“ [18]. To je ukratko cijela logika funkcioniranja ovog razvojnog okvira u pozadini.

4.2. Logika PHP MINI razvojnog okvira

Dva su glavna gradivna dijela aplikacije, **public** i **aplication**. Unutar *public* mape nalaze se HTML, CSS i JavaScript datoteke. Uz njih, tu su slikovni i drugi multimedijski materijali (fontovi, *.pdf* datoteke itd.). Važno je spomenuti glavnu datoteku *index.php* unutar koje se renderiraju najvažniji dijelovi aplikacije.



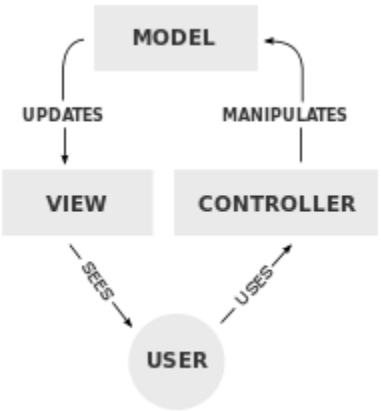
Slika 3. Strukrura public mape

Kao što sama riječ govori u ovoj mapi *aplication* nalazi se aplikacijska struktura potrebna za rad razvojnog okvira. Osnovna aplikacijska struktura sastoji se od sljedećih dijelova:

- **Config** (definirane konstante potrebne za spoj na bazu podataka)
- **Controller** (kontroleri koji se pozivaju prema potrebi aplikacije)
- **Core** (izvršava spoj na bazu i komunikaciju između svih ključnih dijelova MVC arhitekture)
- **Model** (modeli koje kontroler koristi za rad sa podacima)
- **View** (dio arhitekture odgovoran za prikaz obrađenih podataka na ekranu)

4.3. MVC (Model, View, Controler) arhitektura

U uvodu PHP MINI razvojnog okvira spomenuta je njegova baziranost na MVC arhitekturi. Svrha MVC arhitekture je razdvajanje korisničkog sučelja od poslovne logike čime je poboljšana organizacija i preglednost programskog koda [14]. Time je izravno olakšano održavanje kao i nadogradnja aplikacija koje koriste MVC arhitekturu.



Slika 4. MVC arhitektura

MVC arhitektura razdvaja aplikaciju na tri osnovna dijela – **Model, View ili pogled i Controller ili upravitelj** [15]. Svaki dio ima svoja svojstva i zadaću koju mora izvršiti. Sve zajedno predstavlja jedan izlaz (output) u odnosu na zahtjeve korisnika.

Prednosti korištenja MVC tehnologije:

- Lakše održavanje, testiranje, te nadograđivanje aplikacije.
- Jednostavnije dodavanje novih klijenata sa vlastitim objektima View i Controller
- Fleksibilnost kod planiranja i implementiranja objekata Model, omogućena višestruka iskoristivost i modularnost.
- Omogućen paralelan razvoj objekata Model, View i Controller.
- To sve aplikaciju čini proširivom i skalabilnom.

4.3.1. **Model**

Model je dio MVC arhitekture koji je odgovoran za rad s podacima. Neke od situacija gdje će se koristiti u praktičnom dijelu rada su konekcija na bazu podataka, izvršavanje upita, dohvaćanje podataka itd [15]. Dakle, Model odgovara na zahtjeve koji stižu iz Controller objekta, priprema i obrađuje podatke, te obavještava registrirane View objekte da ažuriraju svoj prikaz sa novim podacima.

4.3.2. **View (pogled)**

View uzima podatke od objekta Controller, formatira ih, te prosljeđuje pregledniku. Zadužen je za prikaz podataka i dodaje funkcionalnosti koje su potpuno izolirane od složenih operacija nad podacima. View kao sučelje za interakciju sa sustavom može biti [15]: Web forma, HTML, XHTML, XML/XSLT, WML, Windows forma, itd.

4.3.3. **Controller (upravitelj)**

Controller zadužen je za upravljanje akcijama. Kada korisnik šalje zahtjev sustavu preko korisničkog sučelja (objekt View), objekt Controller prosljeđuje zahtjev Modelu. Model obrađuje i priprema podatke, te nakon toga vraća odgovor. U tom slučaju objekt Controller poziva sve potrebne objekte View koji uzimaju te podatke i prilagođavaju svoj prikaz [15].

4.4. AJAX (Asynchronous JavaScript and XML)

AJAX (Asynchronous JavaScript and XML [2]) je tehnologija koja se koristi za pozadinsku komunikaciju sa poslužiteljem. PHP MINI koristi AJAX za dinamično ažuriranje baze podataka u svom demo primjeru. Takav pozadinski/asinkroni način komunikacije s poslužiteljem omogućuje korisnicima da od servera traže samo dodatne podatke tako da se ne osvježava cijela stranica (npr. Gmail, AHyCo, Facebook) [16].

Slika 5. Ajax princip rada [16]

Sva komunikacija sa poslužiteljem izvršava se pomoću **XMLHttpRequest** [2] objekta što rezultira dinamičnom i brzom web stranicom/aplikacijom sa visokim stupnjem interakcije [16]. U praktičnom dijelu rada XMLHttpRequest pisan je u skraćenom obliku unutar jQuery tehnologije [17] a koristi se kod ispisivanja sadržaja tablica na stranicu, ažuriranja, brisanja te dodavanja novog sadržaja.

5. Izrada aplikacije za dinamično upravljanje MySQL bazom podataka (praktični dio)

Kao praktični primjer završnog rada izradio sam aplikaciju za dinamično upravljanje MySQL bazom podataka koja uključuje sve prethodno obrađene tehnologije na njihovoj osnovnoj razini.

Aplikacija koju sam izradio ima sljedeće funkcionalnosti:

- Izrada MySQL tablice kroz korisničko sučelje
- Ispis postojećih tablica iz MySQL baze podataka
- Prikaz odabrane tablice i njezinih podataka
- Mogućnost brisanja redaka odabrane tablice
- Mogućnost ažuriranja redaka odabrane tablice
- Mogućnost dodavanja novog retka u odabranu tablicu

Važno je napomenuti da se svaka od navedenih radnji procesira u realnom vremenu bez osvježavanja kompletnog sadržaja stranice pomoću AJAX-a. Aplikacija koristi MVC arhitekturu koja razdvaja ključne dijelove aplikacije radi lakše izrade, manipulacije i nadogradnje, te najbitnijeg razumijevanja. Razumijevanje je glavni razlog zašto ova aplikacija koristi MVC arhitekturu, pošto je aplikacija relativno jednostavna i nema osobitu potrebu za korištenjem MVC arhitekture, ali na taj način predstavlja jako pojednostavljenu verziju nekih poznatijih i znatno kompliciranijih aplikacija koje svakodnevno koristimo.

Aplikaciju sam izradio kroz četiri koraka:

1. Izrada baze podataka
2. Instalacija PHP MINI razvojnog okvira i spajanje sa bazom
3. Izrada korisničkog sučelja i dizajn aplikacije
4. Zatim redom sve od navedenih CRUD funkcionalnosti

5.1. Izrada MySQL baze podataka

Da bi se izradila MySQL baza podataka potrebno se je spojiti na sučelje za administraciju relacijskih baza podataka phpMyAdmin dodijeljenim korisničkim imenom i lozinkom te izraditi novu bazu podataka.
Nakon uspješne prijave u sustav potrebno je iz lijevog izbornika odabrati „New“ te potom upisati ime baze, odabrati način kodiranja znakova i odabrati „Create“. Izrađena baza podataka potom je vidljiva u lijevom izborniku. Izrada tablice unutar baze u ovom slučaju nije potrebna pošto će se tablice kreirati dinamičkim putem kroz korisničko sučelje aplikacije koja će se izraditi.

5.2. PDO spajanje na bazu i instalacija PHP MINI frameworka

Kako bi instalirali PHP MINI razvojni okvir potrebno je imati web poslužitelj (engl. *Hosting*) koji podržava PHP 5.3+, MySQL te ima *mod_rewrite* modul aktiviran. Ukoliko poslužitelj ne podržava *mod_rewrite* moguće je instalirati vrlo sličan razvojni okvir TINY koji radi bez te opcije. Nakon što smo provjerili serverske zahtjeve i preuzeli razvojni okvir za službene stranice ili GitHuba, potrebno ga je prenijeti u javni direktorij na poslužitelj.
Za spajanje na bazu podataka koristiti će se PDO tehnologija. PDO je sučelje za pristup bazama podataka u PHP-u te je objektno orijentirano, što znači da je moguće naslijediti i implementirati naprednije metode. Da bi se spojili pomoću PDO-a potrebno je konfigurirati *config.php* datoteku i njezine parametre.

```
define('DB_TYPE', 'mysql');
define('DB_HOST', '127.0.0.1');
define('DB_NAME', 'mini');
define('DB_USER', 'root');
define('DB_PASS', 'zavrsni123');
define('DB_CHARSET', 'utf8');
```

Kod 2. Konstante za spoj na MySQL bazu

U priloženom kodu definiraju se konstante pomoću PHP funkcije *define()*. Specifično je za konstante da se njihova vrijednost ne može mijenjati tijekom izvršavanja koda [19].

- DB_TYPE: definira tip baze podataka
- DB_HOST: definira host na koji se spajamo
- DB_NAME: definira ime baze
- DB_USER: definira korisničko ime
- DB_PASS: definira lozinku za pristup bazi podataka
- DB_CHARSET: definira sustav kodiranja znakova

U mapi *application/core* nalazi se datoteka controller.php u kojem se nalazi metoda *openDatabaseConnection()* sa dva atributa. Prvi atribut *options* definira način na koji će PDO dohvaćati podatke iz baze, ta se radnja izvršava putem funkcije *PDO::ATTR_DEFAULT_FETCH_MODE*. Dakle, umjesto da se rezultat vraća u asocijativnom obliku *$result["user_name“]* sa *FETCH_ASSOC* funkcijom, ovdje će rezultat biti objekt i pristupat će mu se sa *$result->user_name*.

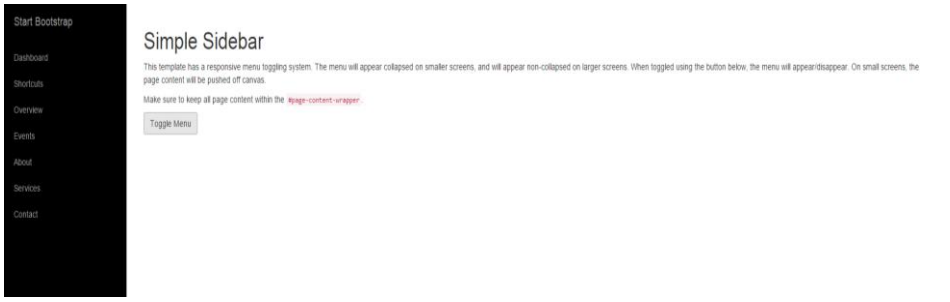
Drugi atribut generira novi PDO spoj na bazu koristeći parametre proslijeđene sa *config.php* datoteke koja je ranije konfigurirana.

```
private function openDatabaseConnection()
{
$options = array(PDO::ATTR_DEFAULT_FETCH_MODE =>
PDO::FETCH_OBJ, PDO::ATTR_ERRMODE => PDO::ERRMODE_WARNING);
$this->db = new PDO(DB_TYPE . ':host=' . DB_HOST . ';dbname='
. DB_NAME . ';charset=' . DB_CHARSET, DB_USER, DB_PASS,
$options);
}
```

Kod 3. Funkcija openDatabaseConnection()

5.3. **Korisničko sučelje i dizajn aplikacije**

U svrhu bržeg razvoja aplikacije nije rađen dizajnerski predložak od nule već je iskorišten predložak sa web stranice Startbootstrap. Predložak koristi istoimeni frontend razvojni okvir Bootstrap koji ima fleksibilnu i jednostavnu strukturu, velik broj elemenata i funkcionalnosti spremnih za korištenje. Potrebno je samo otići na službenu web stranicu, pronaći HTML element koji je spreman za korištenje, te ga kopirati u vlastiti projekt te ga modificirati po želji. Isto vrijedi za JS funkcionalnosti i još mnogo toga. Bootstrap je razvojni okvir otvorenog koda (engl. *open source*).



Slika 6. Bootstrap predložak (original)

Odabrani predložak nudi mogućnost sakrivanja bočnog izbornika što je dobro jer je tada fokus na funkcionalnostima aplikacije. Nakon obrade predloška, izmijenjene su neke boje, dodani elementi i stavke izbornika. Uređeni predložak sprema se sa svim povezanim mapama u *public* mapu MINI razvojnog okvira. Pojedinačne stranice koje će koristiti predložak izrađivati će se unutar *View* objekta MVC arhitekture.

Slika 7. Bootstrap predložak (uređen)

Nakon što je osnovni predložak izrađen potrebno je testirati stranicu. Kako bi mogli prikazivati stranice u MVC arhitekturi potrebno je izraditi kontroler koji će na korisnički zahtjev pozvati HTML sadržaj iz *aplication/view* mape u kojoj su spremljeni predlošci za pojedine stranice web mjesta. Stoga je potrebno u navedenoj mapi izraditi podmapu „home“ i u njoj datoteku *index.php*. *Index.php* je klasična HTML stranica koja će u ovom slučaju sadržavati uvod završnog rada i link na *.pdf* dokument.

Nakon korisničkog zahtjeva kroz URL adresu poziva se kontroler „home“ koji sadrži metodu *index()* unutar koje su deklarirani atributi odgovorni za prikaz početne stranice.

```
    class Home extends Controller
{
    public function index()
    {
        require APP . 'view/_templates/header.php';
        require APP . 'view/home/index.php';
        require APP . 'view/_templates/footer.php';
    }
}
```

Kod 4. Klasa home - pozivanje osnovnog HTML predloška

Kod 4. prikazuje klasu *Home* koja odgovara imenu mape u *aplication/view* podmapi, a metoda *index()* odgovara imenu datoteke unutar mape *Home* kojoj pristupamo kontrolerom. Unutar metode *index()* postoje tri atributa koji pozivaju zaglavlje predloška, tijelo predloška (izrađeni *index.php* sa sadržajem stranice) i podnožje stranice.

Važno je spomenuti da se u ovom primjeru javlja princip **nasljeđivanja** (vidi poglavlje **2.6 Osnovni principi OOP u PHP-u**). Nasljeđivanje omogućuje da klasa *Home* preuzme atribute ili metode od klase *Controller*. Klasa *Controller* sadrži metodu za spajanje na bazu i učitavanje modela, na taj način bilo koja klasa koja naslijedi klasu *Controller*, nemora imati vlastiti spoj na bazu ili bilo koji drugi parametar koji je naslijedila.

5.4. CRUD funkcionalnosti

Create, *read*, *update* i *delete* su fundamentalne funkcije svake dinamične aplikacije, pa tako i ove za manipulaciju MySQL bazom podataka. Sve navedene funkcionalnosti zapravo su metode klase *Crud*. U nastavku je objašnjena izrada svake metode.

5.4.1. **Izrada dinamičkog kreiranja tablice**

Da bi dinamično kreirali tablice potrebno je imati obrazac u kojem je moguće generirati *n* broj redaka te određivati ime, duljinu i tip podataka (ta tri parametra potrebna su da bi se izvršio MySQL *CREATE* upit). Stoga je prvi korak izrada stranice u *View* direktoriju MVC arhitekture pod imenom *newTable.php* koja će sadržavati tablicu sa web formom (obrascem).

```
<td><input class="form-control" type="text"
name="columnName[]" required = "required" placeholder="Ime
```

Kod 5. Primjer polja obrasca

Dinamičko kreiranje tablice zahtjeva dvije metode, *newTable()* koja poziva stranicu *newTable.php* i *create()* koja sakuplja vrijednosti obrasca, pakira ih u niz sa funkcijom *implode*, te šalje na obradu modelu.

Vrijednosti polja spremaju se u niz *columnName[]* kako bi se mogao dodati *n* broj redaka. Nakon potvrde obrasca (engl. *submit*) podaci se šalju *POST* metodom na *create()* gdje se pomoću funkcije *implode* spremaju u odgovarajuće varijable i prosljeđuju modelu *createTable* kao parametri.

U modelu *createTable* vrijednosti parametara se rastavljaju (deserijaliziraju), spremaju u niz, te iteriranjem kroz *for* petlju ispisuju u SQL upit.

```
public function create()
    {
        $tableName = $_POST['tableName'];
        $columnName = implode (";",$_POST['columnName']);
        $dataType = implode (";",$_POST['dataType']);
        $size = implode (";",$_POST['size']);
        $successCreate = $this->model-
>createTable($tableName, $columnName, $dataType, $size);
        $tableName = $this->model->getTables();
    }
```

Kod 6. Metoda create() - dohvaćanje podataka obrasca

Create metoda (*Kod 6.*) dohvaća podatke obrasca proslijeđene metodom *POST*. Odabir metode prosljeđivanja podataka forme izvodi se navođenjem *method* argumenta *form* taga. Moguće vrijednosti method argumenta su *POST* i *GET*. Iz sigurnosnih razloga ne preporuča se korištenje *GET* metode ukoliko se podaci koriste za rad sa bazom. Kod *GET* metode podaci forme šalju se kroz komandnu liniju (query string, tj. iza znaka *?* u adresnoj traci internet preglednika) [4].

Odabirom metode *POST* podaci nisu vidljivi u komandnoj liniji već se šalju transparentno kroz zaglavlje *HTTP requesta*.

Dohvaćeni podaci spremaju se unutar odgovarajućih varijabli pomoću PHP funkcije *implode* koja spaja elemente niza u string, na način da ih odvaja proizvoljnim separatorom (u primjeru to je znak „**;**“). Tako uređene varijable spremne su za prosljeđivanje na model *createTable* kao parametri funkcije.

U primjeru *Kod 7.* na sljedećoj stranici prikazan je model *createTable*. Parametri metode pomoću funkcije *explode* rastavljaju se u niz te spremaju u odgovarajuće varijable. Zatim se funkcijom *count* izbroji koliko stavki sadrži niz kako bi *for* petljom mogli dinamično izraditi redove SQL upita *CREATE TABLE*. Unutar *for* petlje postoji i *if/else* petlja koja sprječava da zadnji red SQL *CREATE TABLE* upita ne završava sa znakom „**,**“ što bi rezultiralo sintaksnom greškom i upit se ne

bi izvršio. Na posljetku PDO metoda *prepare($sql)* vraća instancu *PDOStatement* za zadnji upit te se metodom *execute()* isti izvršava.

```
public function
createTable($tableName,$columnName,$dataType,$size)
    {
        $columnName = explode(";", $columnName);
        $dataType = explode(";", $dataType);
        $size = explode(";", $size);
        $countItems = count($columnName);
        $columnCount = $countItems;
        $linije = '';
        $sql = "CREATE TABLE ".$tableName." (id INT(3)
UNSIGNED AUTO_INCREMENT PRIMARY KEY, " ;
        for($x=0; $x < $columnCount; $x++) {
                    if($x != $columnCount-1) {
                            $linije = ',';
                     } else {
                            $linije = '';
                     }
                    $sql .= $columnName[$x]."
".$dataType[$x]."(".$size[$x].") NOT NULL".$linije;
                };
        $sql .= ")";
        var_dump("Izvršen je upit: ".$sql);
        $query = $this->db->prepare($sql);

        $query->execute();
    }
```

Kod 7. PHP createTable metoda (model)

5.4.2. **Izrada dinamičkog ispisivanja tablica i sadržaja**

Nakon što je moguće putem korisničkog sučelja izraditi tablice u MySQL bazi podataka, potrebno je iste prikazati. Cilj ovog segmenta aplikacije je prikazati popis postojećih tablica u bazi te korisničkom akcijom klika prikazati njihov sadržaj. Kasnije će se dodati funkcije za manipulaciju tim sadržajem.

Prvo je potrebno dodati novu stranicu u *View* direktorij MVC arhitekture pod imenom *read.php* čija će se HTML struktura sastojati od dva glavna dijela. Lijevi *DIV* element u kojem će se generirati popis postojećih tablica baze podatka, te desni u kojem će se sadržaj odabrane tablice prikazivati.

U klasi *Crud* nalazi se metoda *read()* koja je odgovorna za prikaz stranice *read.php* te prikaz postojećih tablica baze. Pomoću argumenta *$tableName* poziva se metoda *getTables()* koja sakuplja informacije o postojećim tablicama

```
public function getTables(){
$sql = "SELECT TABLE_NAME FROM information_schema.TABLES
WHERE TABLE_TYPE='BASE TABLE' AND TABLE_SCHEMA='mini';";
        $query = $this->db->prepare($sql);
        $query->execute();
        return $query->fetchAll();
    }
```

Kod 8. Metoda za dohvaćanje informacija o tablicama iz baze podataka

pomoću SQL upita.

Nakon što su podaci u modelu dohvaćeni i proslijeđeni na kontroler, potrebno ih je ispisati u *read.php*. To je odrađeno pomoću *foreach* petlje (*Kod 9.*) gdje se svaka stavka vraćenog upita ispisuje kao redak tablice.

```
foreach ($tableName as $table ) {
echo '<tr><td class="ajaxTable" style="cursor:pointer;">'.
$table->TABLE_NAME
.'</td></tr>';}
```

Kod 9. Ispisivanje podataka pomoću foreach petlje

Ispis tablica je uspješan, sada je potrebno dodati korisničku akciju klik kojom će se sadržaj odabrane tablice prikazati u desnom dijelu aplikacije gdje trenutno stoji *„Odaberite tablicu sa lijeve strane“*.

Pošto svaki redak tablice ima HTML atribut *class*, jQuery akcijom nakon klika dohvatiti će se sadržaj retka (koji je ujedno i ime tablice) te pomoću AJAX zahtjeva proslijediti kontroleru na daljnju obradu.

Slika 8. Ispis tablica iz baze podataka

Nakon što se u kontroleru podatak dohvati *GET* metodom, šalje se na model metodi koja će izvršiti SQL upit i vratiti sadržaj tražene tablice. Vraćeni sadržaj filtrirati će se u kontroleru te po završetku pomoću AJAX-a ispisati na desnom dijelu stranice.

```
if ($('.ajaxTable').length !== 0) {
$('.ajaxTable').on('click', function(){
            var name = $(this).html();
            $.ajax(url + "crud/readAjax?name=" + name )
                  .done(function(result) {
                  $('#ajax-printTable').html(result);
```

Kod 10. Dohvaćanje vrijedosti retka i slanje AJAX requesta

Kod 10. prikazuje jQuery funkciju koja na klik dohvaća klasu *ajaxTable* te u varijablu *name* sprema njezinu vrijednost. Zatim se otvara novi AJAX zahtjev koji *GET* metodom, kroz URL, šalje ime tablice kontroleru, odnosno metodi *readAjax*.

```
$tableNameAj = $_GET['name'];
$returnTable = $this->model->getAjaxTable($tableNameAj);
```

Kod 11. Dohvaćanje podataka GET metodom

ReadAjax metoda dohvaća poslane podatke te ih prosljeđuje modelu, odnosno metodi *getAjaxTable (Kod 11.)*.

```php
public function getAjaxTable($tableNameAj)
    {
        $sql = "SELECT * FROM $tableNameAj";
        $query = $this->db->prepare($sql);
        $query->execute();
        return $query->fetchAll();
    }
```

Kod 12.Metoda getAjaxTable za dohvaćanje sadržaja tablice

Metoda *getAjaxTable (Kod 12.)* prima jedan parametar, ime tablice (*$tableNameAj*) te SQL upitom dohvaća sadržaj tražene tablice. Dohvaćeni podaci se zatim obrađuju u metodi *readAjax()*. Prvo se ispituje dali je išta vraćeno iz modela, ukoliko nema vraćenih podataka ispisuje se poruka *„Nema unesenih podataka. Tablica je zasad prazna.“*. Ukoliko je vraćen neki sadržaj onda se dinamički izrađuje tablica sa vraćenim sadržajem.

```
foreach($returnTable as $row) {
                $array = get_object_vars($row);
            }
            echo '<tr>';
            $arrayKeys = array_keys($array);
            foreach ($arrayKeys as $key) {
                echo '<th>'.$key.'</th>';
                 }
            echo '<th>Delete</th>';
            echo '<th>Edit</th>';
            echo '</tr>';
            $trId=0;
            foreach($returnTable as $row) {
                $trId++;
                $array = get_object_vars($row);
                $num = count($array);
                $counter = 0;
                $id = $row->id;
                echo '<tr id="'.$id.'">';
                $keyCounter=0;
                $data = '';
                foreach($array as $var) {
                    $counter++;
                    $arrayKeys = array_keys($array);
                    echo '<td>'.$var.'</td>';
                    if($num==$counter) {
                        echo '<td><button class="deleteRow"
data_table="'.$tableNameAj.'"
data_id="'.$id.'">DELETE</button></td>';
                        echo '<td><button type="button"
class="ajaxedit" value="Edit" id="'.$id.'"
data_table="'.$tableNameAj.'">EDIT</button></td>';
                    } $keyCounter++;
                }
                echo '</tr>';
```

Kod 13.Metoda readAjax() - dinamična izrada ispisa tablice

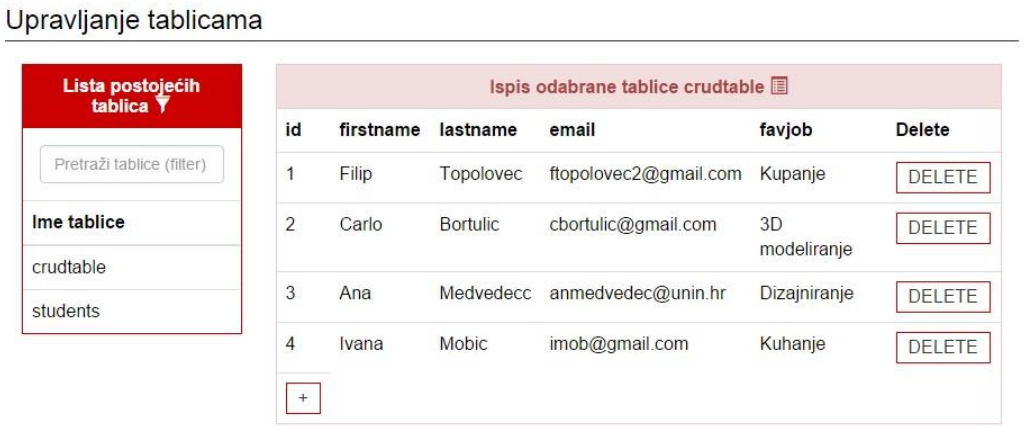
Kod dinamične izrade ispisa tablice *(Kod 13.)* koristi se *foreach* petlja kojom se dohvaćaju redovi vraćene tablice koji se spremaju u varijablu *$array*. Kod spremanja stavki koristi se funkcija *get_object_vars* koja omogućuje pristupanje podacima kao objektima. Potom se funkcijom *array_keys* izvlače imena polja tablice koja će biti prikazana u zaglavlju tablice kod ispisa. Potom se *foreach* petljom generira zaglavlje tablice ovisno o broju vraćenih polja i njihovim imenima. Uz dinamično generirana polja tu su još dva fiksna polja koja se dodaju na kraj svakog retka, to su polja u kojima se nalaze kontrole za manipulaciju retkom: *delete* i *edit*.

Nakon zaglavlja izrađuje se tijelo tablice. *Foreach* petljom dohvaćaju se redovi tablice i pohranjuju u varijablu *$array* koja se potom funkcijom *count* prebrojava. U varijablu *$id* sprema se *id* vrijednost pojedinog retka.

Sada je sve spremno za dinamičnu izradu retka. Nakon ispisa početnog taga retka sa *id*-jem *<tr id="'.$id.'">* dolazi *foreach* petlja koja ispisuje sve stavke niza *$array*.

Unutar *foreach* petlje nalazi se *if* kontroler tijeka koji na kraj retka dodaje dva polja u kojima se nalaze gumbi za brisanje i ažuriranje retka.

Zbog razumijevanja sljedeće funkcionalnosti brisanja sadržaja tablice važno je obratiti pozornost na atribute gumba delete. Atributi gumba *delete* su klasa, *data_table* i *data_id* koji se kod dinamičnog brisanja koriste za identifikaciju retka i njegovo brisanje pomoću AJAX-a.



Slika 9. Dinamični ispis tablice pomoću AJAX-a

5.4.3. **Izrada dinamičkog brisanja sadržaja tablica**

```
$('.deleteRow').on('click', function(){
    var i = $(this).closest('tr');
    $(i).fadeOut("slow","linear");
    var table = $(this).attr('data_table');
    var id = $(this).attr('data_id');
    jQuery.ajax(url + "crud/deleteRow?table=" + name +
"&id=" + id )
    .done(function(result){
        alert('Uspješno ste izbrisali red!');
    })
```

Kod 14. jQuery kod za brisanje retka AJAX-om

Prvi korak brisanja retka tablice je njegova identifikacija na korisničku akciju klik (*Kod 14.*). Nakon dohvaćanja retka pomoću identifikatora *class*, funkcijom se dohvaća najbliži *tr* tag koji se potom sakriva sa jQuery funkcijom *fadeOut*. Vrijednosti atributa *data_table* (koji sadrži ime tablice) i *data_id* spremaju se u pripadajuće varijable nakon čega se šalju metodi *deleteRow* kroz novi AJAX zahtjev *GET* metodom.

Metoda *deleteRow (Kod 15.)* dohvaća podatke poslane *GET* metodom, pohranjuje ih u varijable te prosljeđuje modelu kao parametre. U modelu se nalazi istoimena

metoda *deleteRow* sa dva parametra, imenom tablice i *id* vrijednosti koji se koriste kod SQL upita za brisanje retka tablice.

```
    public function deleteRow($tableName1,$id)
    {
        $sql = "DELETE FROM $tableName1 WHERE id = $id";
        $query = $this->db->prepare($sql);
        $query->execute();
       }
```

Kod 15. Metoda za brisanje retka tablice

5.4.4. **Izrada dinamičkog ažuriranja sadržaja tablica**

Dinamično ažuriranje sadržaja ostvareno je nadogradnjom tablice jQuery tehnologijom tako da se svako polje tablice na akciju klikom pretvara u *input* polje. Na taj način poboljšan je UX jer nije potrebno ugrađivati dodatne gumbe za akciju editiranja, niti dodatne obrasce, štedi se prostor i dobiva na preglednosti. Izmijenjen sadržaj serijalizira se zajedno sa *meta* podacima potrebnim za upis u bazu te se sve zajedno šalje AJAX zahtjevom uz korištenje metode *getJSON*. jQuery *getJSON()* metoda služi za dohvaćanje JSON podataka koristeći AJAX HTTP *GET* zahtjev.

Prvi korak u izradi *update* funkcionalnosti je izrada jQuery skripte koja će nadograditi postojeću HTML tablicu i omogućiti editiranje polja na klik. jQuery skripta *edit.js* sastoji se od dvije osnovne metode, *displayForm* koja radi izmjene na HTML tablici i *changeField* koja serijalizira podatke i izvršava AJAX zahtjev.

```
$(function(){
$('#ajaxprintTable').on('click','td#editable',function(){
displayForm($(this));
    });
});
```

Kod 16. JS funkcija - td event listener

Kod 16. prikazuje osnovnu funkciju koja dohvaća *div* element i prati klik akcije na poljima *(td)* tablice *editable*. Ukoliko dođe do akcije pokreće se funkcija *displayForm (Kod 17.)* koja je opisana na sljedećoj stranici. Unutar funkcije *displayForm* prvo se deklarira i inicijalizira nekoliko varijabli potrebnih za rad funkcije. Varijabla *column* pohranjuje atribut *class* elementa *td* koji se dohvaća kroz *cell* parametar iz glavne funkcije koja se pokreće na klik. Atribut *class* sadrži ime polja koje se dohvaća. Varijabla *id* dohvaća atribut *id tr* elementa retka koji se ažurira. Varijabla *cellWidth* dohvaća vrijednost širine *td* polja koja će se kasnije iskoristiti kao širina *input* elementa. Varijabla prevContent sprema vrijednost polja prije izmjene kako bi u slučaju odustajanja od ažuriranja prvobitan sadržaj mogao biti vraćen u polje. Varijabla *tname* dohvaća vrijednost atributa *data_table* koji sadrži ime tablice.

Posljednja varijabla *form* sadržava *html* obrazac. Atributu *action* postavljena je vrijednost *preventDefault* koja osigurava da obrazac ne osvježava čitavu stranicu nakon slanja kao što inače bi, već se u ovom slučaju podaci dohvaćaju *jQueryjem* i šalju AJAX zahtjevom. Zatim je izrađeno nekoliko input polja u koja su ubačene vrijednosti varijabli koje su ranije deklarirane i inicijalizirane, a služit će kasnije u *PHP*-u za obradu podataka i izradu SQL *update* upita.

```
function displayForm(cell) {

    var column = cell.attr('class'),
        id = cell.closest('tr').attr('id'),
        cellWidth = cell.css('width'),
        prevContent = cell.text(),
        tname = cell.attr('data_table'),
        form = '<form action="javascript:
this.preventDefault"><input type="text" name="newValue"
size"4" value="'+prevContent+'" /><input type="hidden"
name="id" value="'+id+'" />'+'<input type="hidden"
name="column" value="'+column+'" /><input type="hidden"
name="tableName2" value="'+tname+'"></form>';

        cell.html(form).find('input[type=text]')
            .focus()
            .css('width',cellWidth);

        cell.on('click', function(){return false;});

        cell.on('keydown', function(e){
            if (e.keyCode == 13) {//enter

                changeField(cell, prevContent);
            } else if (e.keyCode == 27){
                cell.text(prevContent);
                cell.off('click');
            }
        });
```

Kod 17. JS funkcija displayForm

Varijable su sada spremne za korištenje. Prvo je potrebno ubaciti HTML obrazac koji smo izradili unutar varijable *form* u polje tablice koje želimo ažurirati. Za tu radnju koristi se metoda *cell.html(form)* koja jednostavno uzima izrađen obrazac i ubacuje ga u *td* polje tablice, zatim metoda *.focus()* postavlja kursor unutar input polja tipa tekst i postavlja njegovu širinu prema varijabli *cellWidth*. Nakon toga slijedi jedna *on click* funkcija koja ništa ne vraća i time sprječava ponovno pokretanje funkcije unutar funkcije *displayForm*.

Naposljetku izrađena je funkcija koja prati akciju *keydown* nakon čega se provjerava vrijednost tipke koja je pritisnuta, ukoliko je to tipka sa ASCII kodom 13 koji odgovara tipki *Carriage return*, onda se pokreće funkcija ***changeField*** koja je u nastavku objašnjena. U protivnom je pritisnuta tipka sa ASCII kodom

27 koja odgovara tipki *Escape* ili je kliknuto izvan input polja, tada metoda *.text* vraća prijašnju vrijednost polja u *td* element.

```
function changeField (cell, prevContent) {

    cell.off('keydown');

    var input = cell.find('form').serialize();

    $.getJSON(url + "crud/updateRow?" + input, function(data)
{

        if (data.success)
            cell.html(data.value);
        else {
            alert('There was a problem updating data. Please
try again.');
            cell.html(prevContent);
        }
    });

    cell.off('click');
}
```

Kod 18. JS funkcija changeField

U funkciji *changeField (Kod 18.)* serijaliziraju se podaci obrasca u varijablu input. Nakon toga stvara se novi AJAX zahtjev koji će poslati serijalizirane podatke obrasca na PHP kontroler *updateRow*. Ukoliko je AJAX zahtjev uspješno izveden, polje će se ažurirati metodom *cell.html*. U protivnom će se ispisati greška i vratiti stari sadržaj.

Drugi dio aplikacije obavlja PHP programski jezik. Točnije metoda *updateRow (Kod 19.)* koja se nalazi u kontroleru aplikacije. Metoda *updateRow* dohvaća podatke poslane *GET* metodom te ih sprema u pripadajuće varijable. Nakon toga se prosljeđuju u model aplikacije metodi *updateTable* koja izvrši SQL upit *update*.

```
public function updateRow()
    {
            $column = $_GET['column'];
            $id = $_GET['id'];
            $newValue = $_GET['newValue'];
            $tname = $_GET['tableName2'];

     $response1 = $this->model-
>updateTable($column,$id,$newValue,$tname);

      echo $response1;
    }
```

Kod 19. PHP funkcija updateRow (kontroler)

Metoda *updateTable (Kod 20.)* dohvaća parametre *column,id,newValue* i *tname* koji su potrebni za izradu *update* SQL upita. Nakon što se upit izvrši rezultat se kodira PHP funkcijom *json_encode* koja formatira podatke u JSON format iz razloga jer su podaci poslani u JSON formatu metodom *getJSON()* koja učitava podatke sa servera poslane *GET* i *POST* zahtjevom.

```
public function updateTable($column,$id,$newValue,$tname)
    {

        $sql = "UPDATE $tname SET $column = '$newValue' WHERE
id = $id";
        $query = $this->db->prepare($sql);
        $response['success'] = $query->execute();
        $response['value'] = $newValue;

        $response = json_encode($response);
        return $response;
    }
```

Kod 20. PHP funkcija updateTable (model)

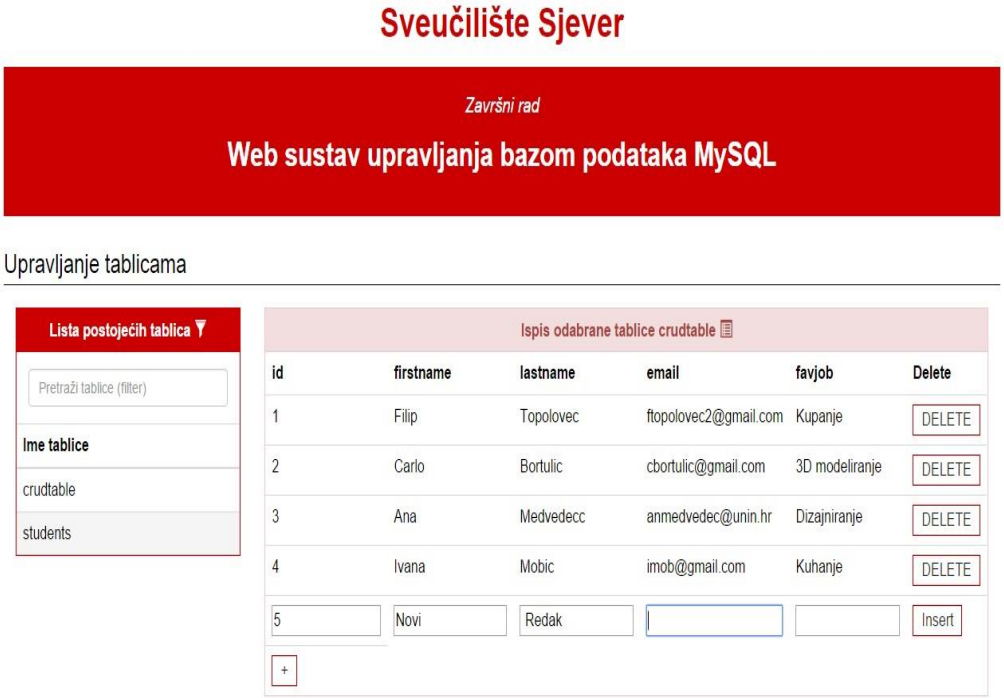
5.4.5. **Izrada dinamičkog dodavanja novog sadržaja tablici**

Dodavanje sadržaja u MySQL tablicu izvršava se SQL naredbom *INSERT INTO*. Sintaksa naredbe izgleda ovako:

```
INSERT INTO table_name (column1,column2,column3,...)
   VALUES (value1,value2,value3,...);
```

Kod 21. INSERT INTO SQL naredba

Iz priložene sintakse naredbe *INSERT INTO* vidljivi su neophodni parametri za izradu upita, to su ime tablice, imena polja tablice te vrijednosti istih.



Slika 10. Unos novog retka u MySQL tablicu

Prvi korak izrade unosa novog retka je kloniranje prethodnog retka u obliku input elemenata. Za to je zadužena jQuery funkcija *clone (Kod 22.)*. Dakle, funkcija klonira postojeći redak tablice te ga dodaje na začelje tablice, zamjenjuje

td vrijednosti elemenata *input* elementima te dodaje gumb „Insert“ kojim se pokreće obrada i AJAX zahtjev.

```
function clone() {
    var i = $('.cloneMe:first').clone(true, true);
        i.find('td').removeAttr('id');
        i.find('td').html('<input type="text"
class="insertFiled">');
        i.find('td:last').html('<button type="button"
onclick="insertRow()" class="insert">Insert</button>');
    $('.cloneMe:last').after(i);
};
```

Kod 22. jQuery funkcija clone

Nakon stvaranja input elemenata potrebna je funkcija za dohvaćanje poslanih podataka, njihovo pakiranje te slanje AJAX zahtjevom na PHP obradu.

```
function insertRow() {
    var i = $('.cloneMe:last');
    var data = [];
    var tableName = i.find('td').attr('data_table');
    i.find('td').each (function() {
        $(this).html($(this).find('input').val());
        data.push($(this).html());
    });
    i.find('td:last').html('<button type="button"
onclick="clone()">+</button>');
    data.pop();
    var request = $.ajax(url + "crud/InsertRow?dataArray=" +
data + "&tableName=" + tableName )
        .done(function(result) {
            alert('Uspješno ste unijeli red!');
        });

};
```

Kod 23. jQuery funkcija InsertRow

Funkcija *insertRow* (*Kod 23.*) dohvaća i sprema podatke u odgovarajuće varijable koje će se proslijediti PHP kontroleru na obradu i kasnije koristiti u izradi SQL *INSERT INTO* upita. U varijablu *tableName* pohranjuje se ime tablice, a nove vrijednosti spremaju se u niz *data*. Posljednji parametar su imena polja

tablice koja nema potrebe na bilo koji način dohvaćati ili obrađivati jQuery-jem pošto im se jednostavno može pristupiti PHP-om preko imena tablice. Nakon pohrane podataka u odgovarajuće varijable u posljednji *td* element dodaje se gumb za daljnje dodavanje novih redaka.

Svi podaci šalju se AJAX zahtjevom na PHP kontroler *InsertRow* (*Kod 24.*)gdje se podaci dalje obrađuju.

```
public function insertRow()
    {
        $dataArray = $_GET['dataArray'];
        $tableName = $_GET['tableName'];
        $dataArray = trim($dataArray,'"');
        $dataArray = explode(",", $dataArray);
        $getArrayKeys = $this->model-
>getArrayKeys($tableName);
        foreach($getArrayKeys as $row) {
                $array1 = get_object_vars($row);
        }
        $arrayKeys = array_keys($array1);

        $insertedRow = $this->model-
>insertRow($dataArray,$tableName,$arrayKeys);
    }
```

Kod 24. PHP metoda insertRow (kontroler)

Podaci poslani *GET* metodom dohvaćaju se u metodi *insertRow* te spremaju u odgovarajuće varijable. Dohvaćenim podacima brišu se navodnici PHP funkcijom *trim* kako ne bi dolazilo do problema sa dvostrukim navodnicima u izradi MySQL upita. Sada je potrebno dohvatiti nazive polja tablice kojoj se dodaje novi redak. Za tu svrhu izrađena je metoda *getArrayKeys* kojoj se prosljeđuje ime tablice.

```
    public function getArrayKeys($tableName)
    {
        $tableName = trim($tableName,'"');
        $sql = "SELECT * FROM $tableName";
        $query = $this->db->prepare($sql);
        $query->execute();
        return $query->fetchAll();
    }
```

Kod 25. PHP metoda getArrayKeys (model)

U metodi se jednostavno izvrši SQL upit koji selektira i vrati sve što tablica sadrži. Funkcijom *get_object_vars* omogući se pristupanje podacima kao objektima. Potom se funkcijom *array_keys* izvlače nazivi polja tablice i spremaju u zasebnu varijablu *arrayKeys* nakon čega se sve zajedno prosljeđuje metodi *insertRow (Kod 26.)* (dio aplikacije u modelu).

```
    public function
insertRow($dataArray,$tableName,$arrayKeys)
    {
        $tableName = trim($tableName,'"');
        $countKeys = count($arrayKeys);
        $linije = '';
        $sql = "INSERT INTO ";
        $sql .= $tableName ."(";
            for ($x=0; $x < $countKeys; $x++){

                if($x != $countKeys-1) {
                            $linije = ',';
                 } else {
                        $linije = '';
                 }
                 $sql .= $arrayKeys[$x]." ".$linije;
            };
        $sql .= ")";

        $countValues = count($dataArray);

        $sql .= "VALUES (";
            for ($x=0; $x < $countValues; $x++){

                if($x != $countValues-1) {
                            $linije = ',';
                 } else {
                        $linije = '';
                 }
                 $sql .= "'".$dataArray[$x]."'"."".$linije;
            };
        $sql .= ")";
        $query = $this->db->prepare($sql);
        $query->execute();
    }
```

Kod 26. PHP metoda insertRow (model)

Metoda *insertRow* prima tri parametra: *dataArray*, *tableName* i *arrayKeys*. Krajnji cilj ove metode je dinamično izraditi SQL *INSERT INTO* upit prema sintaksi *Kod 21*. Dakle, postoje tri parametra spremna za korištenje, potrebno je prebrojiti parametar *arrayKeys* i *dataArray* kako bi izradili odgovarajuća broj polja i vrijednosti za SQL upit. Varijabla *sql* se više puta dopunjava tijekom procesa izrade dinamičnog upita zbog nemogućnosti miješanja programskih struktura SQL upita i petlji. Prva inicijalizacija varijable *sql* je *INSERT INTO*, nakon čega slijedi nadopuna sa parametrom *tableName*. Slijedi *for* petlja koja ispisuje nazive polja dotične MySQL tablice, uz nju, tu je i *if/else* kontroler tijeka koji se brine da se zarez ne ispisuje nakon posljednje stavke upita. Isti proces ponovljen je i za drugi dio *INSERT INTO* upita u kojem se dinamično ispisuju vrijednosti polja, naravno sa drugim parametrima.

6. **Korištenje aplikacije na različitim poslužiteljima**

Aplikacija je u ovom stadiju spremna za korištenje na web poslužitelju. Ovisno o web poslužitelju (engl. *Web Server*) potrebno je znati FTP (engl. *File Transfer Protocol*) podatke za spajanje na poslužitelj. Nakon što se aplikacija sa svim sastavnim dijelovima prenese na poslužitelj potrebno je konfigurirati *config.php* datoteku koja se nalazi unutar *aplication/config* direktorija. Podaci potrebni za konfiguraciju su korisničko ime i lozinka za spajanje na bazu podatka i naziv baze podatka poslužitelja na kojem se baza podataka nalazi. Detaljnije objašnjenje *config.php* datoteke moguće je proučiti pod naslovom *8.2 PDO spajanje na bazu i instalacija PHP MINI frameworka.*

Kako bi aplikacija ispravno funkcionirala potrebno je promijeniti vrijednost *TABLE_SCHEMA* parametra unutar datoteke *model.php* koja se nalazi unutar *aplication/model* direktorija. Vrijednost koju je potrebno izmijeniti je naziv baze podataka koju će aplikacija koristiti.

Ukoliko su prethodni zahtjevi postavljeni prema podacima koji odgovaraju bazi podataka na poslužitelju aplikacija će prepoznati sve izrađene tablice unutar odabrane baze te ih ispisati u izborniku sa lijeve strane. Odabirom željene tablice njen će se sadržaj prikazati u desnom dijelu aplikacije gdje se mogu izvršavati akcije ažuriranja, dodavanja ili brisanja sadržaja. Aplikacija također omogućuje izradu nove tablice unutar baze podataka nakon čega joj je moguće dodati sadržaj.

7. **Zaključak**

Za izradu kvalitetne i dinamične web aplikacije potrebno je poznavati tehnologije obrađene u ovom završnom radu. MVC arhitektura koristi se u svim složenijim web sjedištima zbog svoje dobre organiziranosti koja omogućuje maksimalnu kontrolu nad web stranicama. Korištenje MVC arhitekture podrazumijeva korištenje objektno orijentiranog PHP-a kao i PDO tehnologije za spajanje na baze podataka. AJAX je *frontend* tehnologija koja svim navedenim *backend* funkcionalnostima poboljšava konačan korisnički doživljaj.

Ažuriranje podataka u završnom radu najbolji je primjer poboljšanog korisničkog doživljaja kod CRUD funkcionalnosti te dobro iskorištava potencijal i objedinjuje sve navedene *frontend* i *backend* tehnologije. Kako bi se bilo koji podatak tablice ažurirao dovoljno je izravno odabrati podatak mišem nakon čega on postaje editabilan, unijeti novu vrijednost te pritisnuti tipku *carriage return* (tipka za odlazak u novi redak), odnosno tipku *escape* kako bi odustali od ažuriranja. Takav način rada puno je jednostavniji i efikasniji od standardnih metoda ažuriranja koje često znaju biti kompliciranе, spore i neintuitivne.

Na kraju je važno napomenuti da praktični dio ovog završnog rada sadrži samo osnovne mehanizme potrebne za demonstraciju tehnologija za dinamično upravljanje MySQL bazom podataka te postoje mnoge mogućnosti za daljnji razvoj i širu implementaciju u razne sustave za rad sa bazama podataka.

8. **Literatura**

[1] W3C, »internetlivestats.com,« W3C, 2015. [Mrežno]. Available: http://www.internetlivestats.com/internet-users/#trend. [Pokušaj pristupa 4. 9. 2015].

[2] R. Nixon, Learning PHP, MySQL & JavaScript, 4th Edition, O'Reilly Media, 2014.

[3] Wikipedia, »PHP,« 26. 4. 2015. [Mrežno]. Available: https://hr.wikipedia.org/wiki/PHP. [Pokušaj pristupa 19. 3. 2015].

[4] D. Sklar, Learning PHP 5, O'Reilly Media, 2004.

[5] The PHP Group, »History of PHP,« 2015. [Mrežno]. Available: http://php.net/manual/en/history.php.php. [Pokušaj pristupa 21. 3. 2015.].

[6] L. Ullman, PHP Advanced and Object-Oriented Programming: Visual QuickPro Guide (3rd Edition), Peachpit Press, 2012.

[7] FER, »php uvod u objektno,« [Mrežno]. Available: http://php.fer.hr/predavanja/09_php_uvod_u_objektno.html#/. [Pokušaj pristupa 22. 3. 2015.].

[8] N. Plazibat, »Objektno orijentirano programiranje,« 10 2002. [Mrežno]. Available: http://www.oss.unist.hr/~jvrlic/dokumenti/OOP/oop0203.pdf. [Pokušaj pristupa 22. 3. 2015.].

[9] cplusplus.com, »History of C++,« 2015. [Mrežno]. Available: http://www.cplusplus.com/info/history/. [Pokušaj pristupa 4. 9. 2015.].

[10] M. Zandstra, PHP Objects, Patterns, and Practice, Apress; 4 edition, apress, 2013.

[11] M. Turkusic, »OOP predavanja,« 2014. [Mrežno]. Available: http://www.scribd.com/doc/229236032/OOP-Predavanja#scribd. [Pokušaj pristupa 4. 9. 2015.].

[12] php.com.hr, »Uvod u MySQL,« 2008. [Mrežno]. Available: http://php.com.hr/66. [Pokušaj pristupa 21. 5. 2015].

[13] M. Ivan, »MySQL Tipovi podataka – Data Types,« 10. 7. 2009. [Mrežno]. Available: http://www.hdonweb.com/programiranje/mysql-tipovi-podataka-data-types. [Pokušaj pristupa 4. 9. 2015.].

[14] J. Stump, »Understanding MVC in PHP,« 2015.. [Mrežno]. Available: http://archive.oreilly.com/pub/a/php/archive/mvc-intro.html. [Pokušaj pristupa 5. 9. 2015.].

[15] FER, »Logička arhitektura,« [Mrežno]. Available: http://docbook.rasip.fer.hr/ddb/res/46/Ch3.3.2.html. [Pokušaj pristupa 22. 3. 2015].

[16] M. Brajša, »Uvod u AJAX,« [Mrežno]. Available: http://web.zpr.fer.hr/ergonomija/2006/brajsa/index.html. [Pokušaj pristupa 22. 3. 2015].

[17] The jQuery Foundation, »jQuery.ajax(),« The jQuery Foundation, 2015.. [Mrežno]. Available: http://api.jquery.com/jquery.ajax/. [Pokušaj pristupa 4. 9. 2015.].

[18] DEV METAL, »MINI2, an extremely simple barebone PHP application on top of Slim,« 2015. [Mrežno]. Available: http://www.dev-

metal.com/mini2-extremely-simple-barebone-php-application-top-slim/. [Pokušaj pristupa 22. 3. 2015].

[19] The PHP Group, »Constants,« The PHP Group, 2015.. [Mrežno]. Available: http://php.net/manual/en/language.constants.php. [Pokušaj pristupa 7. 9. 2015.].

Popis slika

Popis primjera kodova

Popis tablica

Prilog A – PHP kod

Datoteka *crud.php* (upravitelj, controller)

```
<?php

class Crud extends controller
{

    public function newTable()
    {
        //samo view u kojem je obrazac
        require APP . 'view/_templates/header.php';
        require APP . 'view/crud/newTable.php';
        require APP . 'view/_templates/footer.php';
    }

    public function create()
    {
        $tableName = $_POST['tableName'];

        $columnName = implode (";",$_POST['columnName']);
        $dataType = implode (";",$_POST['dataType']);
        $size = implode (";",$_POST['size']);


        $successCreate = $this->model->createTable($tableName,
$columnName, $dataType, $size);
        $tableName = $this->model->getTables();

    }

    public function read()
    {

        $tableName = $this->model->getTables();
        $returnTable = array();

        require APP . 'view/_templates/header.php';
        require APP . 'view/crud/read.php';
        require APP . 'view/_templates/footer.php';
    }



    public function readAjax()
    {
        $tableName = $this->model->getTables();

        $tableNameAj = $_GET['name'];
        $returnTable = $this->model->getAjaxTable($tableNameAj);
```

```
        if(!$returnTable) {
            echo '<p>Nema unesenih podataka. Tablica je zasad
prazna.</p>';
        } else {
            echo '<div class="panel panel-danger">
                    <div class="panel-heading">
                        <h3 class="panel-title">Ispis odabrane
tablice <b>'. $tableNameAj.'</b> <span class="glyphicon
glyphicon-list-alt" aria-hidden="true"></span></h3>
                    </div>
                    <table class="table table-hover" id="dev-
table">
                        <tbody>';

            foreach($returnTable as $row) {
                $array = get_object_vars($row);
            }
            echo '<tr>';
            $arrayKeys = array_keys($array);
            foreach ($arrayKeys as $key) {
                echo '<th>'.$key.'</th>';
                 }
            echo '<th>Delete</th>';
            echo '</tr>';

            $trId=0;
            foreach($returnTable as $row) {
                $trId++;
                $array = get_object_vars($row);
                $num = count($array);
                $counter = 0;
                $id = $row->id;

                echo '<tr id="'.$id.'" class="cloneMe">';
                $keyCounter=0;

                foreach($array as $var) {
                    $counter++;

                    echo '<td
class="'.$arrayKeys[$keyCounter].'" id="editable"
data_table="'.$tableNameAj.'">'.$var.'</td>';

                    if($num==$counter) {
                        echo '<td><button class="deleteRow"
data_table="'.$tableNameAj.'"
data_id="'.$id.'">DELETE</button></td>';
                    }
                    $keyCounter++;

                }

                echo '</tr>';
```

```
            }
            echo '<tr>';
                echo '<td><button type="button"
onclick="clone()">+</button></td>';
            echo '</tr>';
            echo '</tbody></table></div>';
        }
    }

    public function updateRow()
    {
            $column = $_GET['column'];
            $id = $_GET['id'];
            $newValue = $_GET['newValue'];
            $tname = $_GET['tableName2'];

            $response1 = $this->model-
>updateTable($column,$id,$newValue,$tname);
        echo $response1;
    }

    public function deleteRow()
    {
        $tableName1 = $_GET['table'];
        $id = $_GET['id'];
        $deleteRow = $this->model->deleteRow($tableName1,$id);

    }


    public function insertRow()
    {
        $dataArray = $_GET['dataArray'];
        $tableName = $_GET['tableName'];
        $dataArray = trim($dataArray,'"');
        //var_dump($dataArray);
        $dataArray = explode(",", $dataArray);
        $getArrayKeys = $this->model->getArrayKeys($tableName);
        foreach($getArrayKeys as $row) {
                $array1 = get_object_vars($row);
        }
        $arrayKeys = array_keys($array1);
        var_dump($dataArray);
        $insertedRow = $this->model-
>insertRow($dataArray,$tableName,$arrayKeys);
    }

}

?>
```

Datoteka *model.php* (model)

```
<?php

class Model
{
    /**
     * @param object $db A PDO database connection
     */
    function __construct($db)
    {
        try {
            $this->db = $db;
        } catch (PDOException $e) {
            exit('Database connection could not be
established.');
        }
    }

    public function
createTable($tableName,$columnName,$dataType,$size)
    {
        $columnName = explode(";", $columnName);
        $dataType = explode(";", $dataType);
        $size = explode(";", $size);

        $countItems = count($columnName);
        $columnCount = $countItems;

        $linije = '';

        $sql = "CREATE TABLE ".$tableName." (id INT(3) UNSIGNED
AUTO_INCREMENT PRIMARY KEY, " ;
        for($x=0; $x < $columnCount; $x++) {
                    if($x != $columnCount-1) {
                        $linije = ',';
                     } else {
                        $linije = '';
                     }
                    $sql .= $columnName[$x]."
".$dataType[$x]."(".$size[$x].") NOT NULL".$linije;
                };
        $sql .= ")";
        var_dump("Izvršen je upit: ".$sql);
        $query = $this->db->prepare($sql);

        $query->execute();
    }

    public function getTables(){

        $sql = "SELECT TABLE_NAME FROM information_schema.TABLES
WHERE TABLE_TYPE='BASE TABLE' AND TABLE_SCHEMA='mini';";
```

```
        $query = $this->db->prepare($sql);

        $query->execute();

        return $query->fetchAll();

    }

    public function getAjaxTable($tableNameAj)
    {
        $sql = "SELECT * FROM $tableNameAj";
        $query = $this->db->prepare($sql);
        $query->execute();
        return $query->fetchAll();
    }

    public function deleteRow($tableName1,$id)
    {
        $sql = "DELETE FROM $tableName1 WHERE id = $id";
        $query = $this->db->prepare($sql);
        $query->execute();
    }

    public function getEditRow($editRowId,$tableName)
    {
        $sql = "SELECT * FROM $tableName WHERE id = $editRowId";
        $query = $this->db->prepare($sql);
        $query->execute();
        return $query->fetchAll();
    }

    public function updateTable($column,$id,$newValue,$tname)
    {

        $sql = "UPDATE $tname SET $column = '$newValue' WHERE id
= $id";
        $query = $this->db->prepare($sql);
        $response['success'] = $query->execute();
        $response['value'] = $newValue;

        $response = json_encode($response);
        return $response;
    }

    public function getArrayKeys($tableName)
    {
        $tableName = trim($tableName,'"');
        $sql = "SELECT * FROM $tableName";
        $query = $this->db->prepare($sql);
        $query->execute();
        return $query->fetchAll();
    }
```

```
    public function insertRow($dataArray,$tableName,$arrayKeys)
    {
        $tableName = trim($tableName,'"');
        //var_dump($tableName);
        $countKeys = count($arrayKeys);
        $linije = '';
        $sql = "INSERT INTO ";
        $sql .= $tableName ."(";
            for ($x=0; $x < $countKeys; $x++){

                if($x != $countKeys-1) {
                          $linije = ',';
                 } else {
                      $linije = '';
                 }
                 $sql .= $arrayKeys[$x]." ".$linije;
            };
        $sql .= ")";

        $countValues = count($dataArray);

        $sql .= "VALUES (";
            for ($x=0; $x < $countValues; $x++){

                if($x != $countValues-1) {
                          $linije = ',';
                 } else {
                      $linije = '';
                 }
                 $sql .= "'".$dataArray[$x]."'"."".$linije;
            };
        $sql .= ")";
        $query = $this->db->prepare($sql);
        $query->execute();
        //var_dump($sql);
    }
}
```

Datoteka *read.php* (Pogled ili View objekt aplikacije za stranicu „Upravljanje tablicama“)

```
<!-- Page Content -->
        <div id="page-content-wrapper">
            <button type="button" class="hamburger is-closed"
data-toggle="offcanvas">
                <span class="hamb-top"></span>
                <span class="hamb-middle"></span>
                <span class="hamb-bottom"></span>
            </button>
            <div class="container-fluid">
                <div class="row">
                    <div class="col-md-8 col-md-offset-2">
                        <div class="jumbotron">
                            <h1 style="font-
size:24pt;">Upravljanje tablicama</h1>
                            <p>Ispis tablica iz MySQL baze</p>
                        </div>
                        <div class="col-md-3">
                            <div class="panel panel-primary">
                                <div class="panel-heading">
                                    <h3 class="panel-
title">Lista kreiranih tablica <span class="glyphicon glyphicon-
filter clickable filter" aria-hidden="true" data-
toggle="tooltip" title="Sakrij filter" data-
container="body"></span></h3>
                                </div>
                                <div class="panel-body">
                                    <input type="text"
class="form-control" id="dev-table-filter" data-action="filter"
data-filters="#dev-table" placeholder="Pretraži tablice
(filter)" />
                                </div>
                                <table class="table table-hover"
id="dev-table">
                                    <thead>
                                        <tr>
                                            <th>Ime tablice</th>
                                        </tr>
                                    </thead>
                                    <tbody>
                                    <?php
                                        foreach ($tableName as
$table ) {
                                            echo '<tr><td
class="ajaxTable" style="cursor:pointer;">'.$table->TABLE_NAME
.'</td></tr>';}
                                    ?>
                                    </tbody>
                                </table>
                            </div>
                        </div>
```

```
                            <div class="col-md-9" id="ajax-
printTable">
                            <p>Odaberite tablicu sa lijeve
strane</p>
                            </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
            <!-- /#page-content-wrapper -->
```

Prilog B – JS kod

Datoteka *application.js* (kod za ispisivanje tablica i brisanje redaka tablica)

```
$(function() {
    if ($('.ajaxTable').length !== 0) {
        $('.ajaxTable').on('click', function(){
            var name = $(this).html();
            $.ajax(url + "crud/readAjax?name=" + name )
                .done(function(result) {
                    $('#ajax-printTable').html(result);
                    $('.deleteRow').on('click', function(){
                        var i = $(this).closest('tr');
                        $(i).fadeOut("slow","linear");
                        var table = $(this).attr('data_table');
                        var id = $(this).attr('data_id');
                        jQuery.ajax(url +
"crud/deleteRow?table=" + name + "&id=" + id )
                        .done(function(result){
                            alert('Uspješno ste izbrisali
red!');
                        })
                    });
                })
                .fail(function() {
                    alert('Došlo je do pogreške!');
                })
                .always(function() {
                });
        });
            jQuery('#ajax-
printTable').on('click','.ajaxedit',function(){
            var editId = jQuery(this).attr('id');
            var tableName = jQuery(this).attr('data_table');
            jQuery.ajax(url + "crud/getEdit?editId=" + editId +
"&tableName=" + tableName)
            .done(function(result) {
                $('#ajax-editTable').html(result);
                })
            });
    }
});
```

Datoteka *edit.js* (kod za ažuriranje tablica)

```
$(function(){
    $('#ajax-printTable').on('click','td#editable',function(){
        displayForm($(this));
    });
});
function displayForm(cell) {
    var column = cell.attr('class'),
        id = cell.closest('tr').attr('id'),
        cellWidth = cell.css('width'),
        prevContent = cell.text(),
        tname = cell.attr('data_table'),
        form = '<form action="javascript:
this.preventDefault"><input type="text" name="newValue" size"4"
value="'+prevContent+'" /><input type="hidden" name="id"
value="'+id+'" />'+'<input type="hidden" name="column"
value="'+column+'" /><input type="hidden" name="tableName2"
value="'+tname+'"></form>';
        cell.html(form).find('input[type=text]')
            .focus()
            .css('width',cellWidth);
        cell.on('click', function(){return false;});
        cell.on('keydown', function(e){
            if (e.keyCode == 13) {//enter
                changeField(cell, prevContent);
            } else if (e.keyCode == 27){
                cell.text(prevContent);
                cell.off('click');
            }
        });
}
function changeField (cell, prevContent) {

    cell.off('keydown');

    var input = cell.find('form').serialize();
    console.log(input);
    $.getJSON(url + "crud/updateRow?" + input, function(data) {

        if (data.success)
            cell.html(data.value);
        else {
            alert('There was a problem updating data. Please try
again.');
            cell.html(prevContent);
        }
    });
    cell.off('click');
}
```

Datoteka *dodaj_red.js* (kod za dodavanje redaka u tablicu)

```
function clone() {
	var i = $('.cloneMe:first').clone(true, true);
		i.find('td').removeAttr('id');

		i.find('td').html('<input type="text"
class="insertFiled">');
		i.find('td:last').html('<button type="button"
onclick="insertRow()" class="insert">Insert</button>');
	$('.cloneMe:last').after(i);
};
//$('.cloneMe').on('click', '.insert', function()
function insertRow() {
	var i = $('.cloneMe:last');
	var data = [];
	var tableName = i.find('td').attr('data_table');
	i.find('td').each (function() {
		$(this).html($(this).find('input').val());
		data.push($(this).html());
	});
	i.find('td:last').html('<button type="button"
onclick="clone()">+</button>');
	data.pop();
	//var serialized = $(data).serialize();
	console.log(data + tableName);
	var request = $.ajax(url + "crud/InsertRow?dataArray=" +
data + "&tableName=" + tableName )
	  .done(function(result) {
		alert('Uspješno ste unijeli red!');
	  });

};
```

Prilog C – CD