Izrada mrežnog servisa za čavrljanje primjenom MVC-a i PHP-a

Petrović, Darko

Sveučilište
Sjever

Završni rad br. 685/MM/2020

Izrada mrežnog servisa za čavrljanje primjenom MVC-a i PHP-a

Darko Petrović, matični broj: 2229/336

Varaždin, rujan 2020. godine

Sveučilište
Sjever

Multimedija, oblikovanje i primjena

Završni rad br. 685/MM/2020

Izrada mrežnog servisa za čavrljanje primjenom MVC-a i PHP-a

Student

Darko Petrović, matični broj: 2229/336

Mentor

Vladimir Stanisavljević, pred.,dipl.inf.

Varaždin, rujan 2020. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL Odjel za multimediju

STUDIJ preddiplomski stručni studij Multimedija, oblikovanje i primjena

PRISTUPNIK Darko Petrović | MATIČNI BROJ 2229/336

DATUM 02.09.2020. | KOLEGIJ Programski alati 3

NASLOV RADA Izrada mrežnog servisa za čavrljanje primjenom MVC-a i PHP-a

NASLOV RADA NA ENGL. JEZIKU Implementation of a chat web service using MVC and PHP

MENTOR mr.sc. Vladimir Stanisavljević | ZVANJE Viši predavač

ČLANOVI POVJERENSTVA

1. izv.prof.dr.sc. Dean Valdec, pred. - predsjednik
2. doc.dr.sc. Andrija Bernik - član
3. mr.sc. Vladimir Stanisavljević, v.pred. - mentor
4. mr.sc. Matija Mikac, v. predavač - rezervni član
5.

Zadatak završnog rada

BROJ 685/MM/2020

OPIS

Objektno orijentirano programiranje (OOP) i MVC (model-view-controler) su značajne paradigme u programskom inženjerstvu. Premda spada u proceduralne jezike, programski jezik PHP u novijim inačicama podržava većinu uobičajenih OOP konstrukcija, a neki razvojni okviri i MVC. U ovom radu potrebno je:

* opisati osnove objektno orijentiranog programiranja i pregled osnovnih OOP mogućnosti u programskom jeziku PHP i kroz kraće primjere pokazati kako se oni koriste
* posebnu pozornost obratiti na PHP Data Objects (PDO) za perzistenciju podataka i koristeći ih pohraniti jednostavne i složene objekte u neku bazu podataka
* korištenjem prikazanih tehnologija izvesti mrežni servis za čavrljanje (chat)
* detaljno opisati programski kod i tehnologije koje su korištene u radu

REPUBLIKA HRVATSKA
SVEUČILIŠTE SJEVER

ZADATAK URUČEN 17.09.2020. | POTPIS MENTORA Vlado

Sažetak

Programiranje i razvoj aplikacija postaju sve zahtjevniji zadaci. Razvojem novih tehnologija, mogućnosti i nastajanjem sve kompleksnijih problema, potrebno je uložiti više vremena za izradu projekta.

Fokus rada je analiza Model-View-Controller (MVC) arhitekture (izumljena je 70-tih godina) u PHP-u, jednom od najpoznatijih jezika u domeni web-tehnologija. PHP ima podršku za objektno orijentirano programiranje i za rad s bazama podataka. Tek od verzije 5.0, PHP podržava objektno orijentirano programiranje (OOP). OOP pomaže programerima u pisanju čišćeg kôda kojeg je lakše mijenjati i ponovno koristiti.

Za praktični dio završnog rada izrađena je PHP aplikacija za slanje poruka, pronalaženje ljudi i izradu personaliziranog profila.

Ključne riječi u završnom radu: **PHP, objektno orijentirano programiranje, baze podataka, MVC, MVC arhitektura.**

Abstract

Programming and application development are becoming increasingly demanding tasks. With the development of new technologies, opportunities and the rise of increasingly complex problems, more time needs to be invested to develop the project.

The focus of this work is the analysis of model-view-controller (MVC) architecture (invented in the 1970s) in PHP, one of the most famous programming languages in the domain of web technologies. PHP has support for object-oriented programming and for working with databases. PHP started supporting object-oriented programming (OOP) in version 5.0. OOP helps developers write cleaner code that is easier to change and reuse.

For the practical part of this work a PHP application for messaging, finding people and creating a personalized profile has been created.

Keywords: **PHP, objektno orijentirano programiranje, baze podataka, MVC, MVC arhitektura.**

Popis korištenih kratica

URL	Jedinstveni lokator resursa (eng. Uniform Resource Locator)
WEB	eng. World Wide Web
HTTP	eng. Hypertext Transfer Protocol
HTML	Sintaksa za obilježavanje hipertekstualnih dokumenata (eng. HyperText Markup Language)
CSS	eng. Cascading Style Sheets
AJAX	Asinkroni JavaScript i XML (engl. Asynchronous JavaScript and XML)
PHP	eng. Personal Home Page (PHP: Hypertext Preprocessor)
OOP	Objektno Orijentirano Programiranje (eng. Object Oriented Programming)
DBMS	Sustav za upravljanje bazom podataka (eng. Database Management System)
CRUD	eng. Create, read, update, delete
SQL	eng. Structered Query Language
PDO	PHP objekti podataka (eng. PHP Data Objects)
MVC	eng. Model View Controller
DAO	eng. Data Access Object
DTO	eng. Dana Transfer Object
API	eng. Application Programming Interfaces
MC	eng. Model Controller

Sadržaj

1. Uvod

Model-view-controller (MVC) je arhitekturalni uzorak (engl. Architectural pattern) koji se koristi u programskom inženjerstvu. Uspješna uporaba uzorka izolira poslovnu logiku od korisničkog sučelja, što rezultira programom u kojem je lakše modificirati vizualni izgled aplikacije ili osnovna poslovna pravila bez utjecaja na druge dijelove aplikacije [1].

U prvom dijelu rada će biti kratki opis jezika (HTML, CSS, JavaScript, Jquery i PHP) korištenih u praktičnom dijelu rada. S obzirom na usku povezanost MVC arhitekture sa objektno orijentiranim programiranjem (OOP), teoretski će se razraditi bitni koncepti OOP i radu s MySQL bazama podataka u PHP jeziku. Ostatak prvog dijela rada će biti iskorišten za teoretsku razradu MVC i MC arhitekture, usporedbe MVC radnih okvira za PHP i SOLID principa.

U drugom dijelu rada se opisuje izrada aplikacije za slanje poruka, pronalaženje ljudi i izradu personaliziranog profila korištenjem nekih od navedenih tehnologija u teoretskom dijelu rada. Fokus drugog dijela rada će biti stavljen na dio aplikacije zadužen za slanje poruka (ostatak je više kao potpora ovom dijelu). Aplikacija bi trebala biti izrađena koristeći MVC arhitekturu.

2. HTML i CSS

2.1. HTML

Hypertext Markup Language (HTML) je jezik za označavanje dizajna dokumenta kako bi se prikazivao u web-pregledniku. HTML je osnova svake web stranice. On web pregledniku daje podatke o sadržaju i strukturi web stranice, preglednik dalje oblikuje stranicu ovisno o primljenim podacima. Kod prikazivanja mu mogu pomoći tehnologije kao što su *Cascading Style Sheets* (CSS) i skriptni jezici kao što je JavaScript.

2.1.1. Povijest HTML-a

Tim Berners-Lee (zaposlenik CERN-a) je 1991. godine napisao dokument „HTML tags“ koji je bio prvi javno dostupan opis HTML-a. Dokument je sadržavao opis prvih 18 elemenata HTML-a (prvi puta su neke stvari standardizirane). Berners-Lee se smatra izumiteljem World Wide Web (WEB). 1994. godine pokrenuo je World Wide Web Consortium (W3C) čija svrha je standardizacija tehnologija koje se koriste na web-u. Od tada se konstantno radilo na HTML-u i izlaskom svake sljedeće verzije dodane su nove mogućnosti i pojednostavnile su se neke stvari.

Prva službena verzija HTML-a je izbačena 1993. godine. Cilj prve verzije bio je jednostavnije strukturiranje podataka, odnosno teksta. Verzija 2.0 HTML-a je bila prva imenovana verzija, no nije proglašena standardnom. 1995. godine W3C objavljuje verziju 3.0 HTML-a kojoj je glavna značajka bila mogućnost definiranja tablica. U prosincu 1997. godine je predstavljen HTML4, 2 godine kasnije bila je izbačena konačna verzija 4.01 HTML-a. Godine 2000. predstavljen je XHTML (kombinacija XML3 i HTML) koji je skup pravila XML-a primijenjen na HTML-u. Suradnjom W3C-a i HWATWG-a godine 2008. objavljen HTML5. Razvoj HTML5 dovršen je godine 2014. te postaje službeni HTML standard [2].

2.1.2. Sintaksa HTML-a

Elementi su blokovi od kojih se sastoji HTML kôd. Oni određuju na koji će način internetski preglednici prikazivati određene dijelove sadržaja. Njih se prepoznaje po tome što imaju početnu <> i krajnju </> oznaku. Atributi se nalaze između ovih oznaka. Tag-ovi se uvijek nalaze između znakova < i >.

Kako bi se HTML kôd moga interpretirati, potrebno je pretraživačima dati do znanja da se radi o HTML-u:

- Mora se izradit HTML dokument. U editoru se otvara nova datoteka i sprema se sa ekstenzijom *.html*. Početna stranica uglavnom ima naziv *index.html*.
- HTML dokument mora sadržavati određene elemente koji pregledniku govore da se radi o HTML dokumentu.

Osnovna struktura HTML kôda *Kôd 2.1 Primjer HTML kôda klase*:

```
<!doctype html>

<html>

<head>

    <title> Naslov </title>

</head>

<body>

    <p> Sadržaj </p> <!-- Komentar -->
     <!—
Komentar
-->

</body>

</html>
```

Kôd 2.1 Primjer HTML kôda

Više o HTML-u se može pronaći na stranicama W3C.

2.2. CSS

Nakon što se stranica napuni sadržajem, potrebno je cijeli taj sadržaj stilizirati. Za to je zadužen Cascading Style Sheets (CSS). Web pretraživači napisani kôd pretvaraju i interpretiraju u ono što korisnik vidi na ekranu. Bez CSS-a web stranice bi bile samo hrpa crnog (U slučaju da je link onda plave boje i podcrtani) teksta na bijeloj pozadini, istog fonta i bez lijepo izgleda.

2.2.1. Povijest CSS-a

Prije CSS-a postojali su mnogi nestandardizirani stilski jezici. CSS je nastao kombinacijom CHSS-a5 i SSP-a6 jer su se ova dva stilska jezika pokazala kao najboljima. Bert Bos i Hakon

Wium Lie su počeli raditi zajedno kako bi razvili CSS. Njihov razvojni prijedlog je bio predstavljen na konferenciji "Mosaic and the Web" 1994. i 1995. godine. Tek na kraju 1996. godine bila je predstavljena prva standardizirana verzija nazvana CSS1. CSS2 se prvi puta pojavljuje u 1997. godini, no kao službeni standard se počeo primjenjivati 1998. godine [3]. Već 1998. se krenulo sa razvojem CSS3, no prvi puta se pojavljuje u 1999. godini. Za razliku od prijašnjih verzije, CSS3 je podijeljen na više odvojenih modula. Ti moduli se implementirani odvojeno i međusobno su neovisni. Iako je CSS3 aktualni standard i dalje se razvija.

2.2.2. Sintaksa CSS-a

CSS se može pisati na 3 načina: linijski, unutarnji i vanjski. Linijski i unutarnji način pisanja se pišu zajedno sa HTML-om u istom dokumentu, dok se vanjski CSS piše u odvojenom dokumentu koji se sprema sa nastavkom *.css*.

- Linijski CSS – kôd će se isključivo odnositi na HTML element uz kojeg je napisan. Određenom HTML elementu se dodaje atribut *style*. Tom atributu se dodijeli vrijednost koja poštuje sintaksu CSS-a. Dobro je izbjegavati ovaj način pisanja CSS.
- Unutarnji CSS – Piše se po standardnoj CSS sintaksi unutar HTML dokumenta (mora biti pisan unutar oznake <style></style> kao atribut). Nedostatak je to što napisan kôd vrijedi samo za taj dokument.
- Vanjski CSS – Najpraktičniji način pisanja CSS. CSS se piše u odvojenom dokumentu koji se sprema sa nastavkom *.css*. Glavna prednost je to što je primjenjiv u bilo kojem HTML dokumentu. Nije potrebno pisati istu stvar više puta za svaki HTML dokument.

Kao i svaki drugi jezik, CSS isto ima svoja pravila pisanja kôda i sintaksu. Svaki blok kôda mora sadržavati selektor, svojstvo i vrijednost. Selektor pretraživaču govori na koji HTML element se taj blok odnosi. Jedan selektor može sadržavati više svojstva i vrijednosti. Svojstvo govori koje će stilsko sredstvo biti primijenjeno. Svakom svojstvu je dodijeljena vrijednost te mu govori što da se primjeni (npr. koje boje će neki HTML element biti).

Struktura CSS kôda (klasa) *Kôd 2.2 Primjer CSS kôda klase*:

```
.timesNewRoman{
    font-family: "Times New Roman"; /* komentar */
    font-style: Normal;
    font-size: 18px;
    color: rgba(252, 255, 235, 1);
}

/*
komentar
*/
```

Kôd 2.2 Primjer CSS kôda klase

U danom primjeru selektor je „timesNewRoman“, svojstva su „font-family“, „font-style“, „font-size“ i „color“. Svakom svojstvu je dodijeljena vrijednost, npr. Svojstvu „font-family“ je dodijeljena vrijednost „Times New Roman“.

Više o CSS-u se može pronaći na W3C-u.

3. JavaScript

JavaScript je jednostavan, interpretiran programski jezik namijenjen ponajprije razvoju interaktivnih HTML-stranica. Jezgra JavaScripta uključena je u većinu današnjih preglednika (Internet Explorer, Google Chrome, Mozilla Firefox, Opera, Safari i drugi) [4].

Svaki jezik koji se interpretira (izvršava ga *interpreter*) naziva se skriptni jezik. Kod skriptnog jezika se skripte ne trebaju prevoditi u strojni jezik. JavaScript je klijentski jezik, jer se kod interpretira na klijentskoj strani u pregledniku (klijentu). Glavni predstavnik klijentskih jezika je JavaScript [4].

3.1. Povijest JavaScript-a

Netscape je u godini 1995. nekoliko prvih inačica JavaScript-a. Nešto kasnije Microsoft objavljuje sličan jezik JavaScript-u pod nazivom JScript. Organizacija ECMA je zadužena za standardizaciju skriptnih jezika. ECMAScript je naziv pod kojim se ti standardi objavljuju. Prva inačica standarda je nastala 1997 godine. Taj standard je od strane međunarodnog ISO standarda odobren u 1998. godini.

3.2. Sintaksa JavaScript-a

JavaScript se ovako može uključiti u HTML-dokument:

- pisanjem kôda kao HTML-atributa (npr. onclick ili onmouseover)
- pisanjem kôda između oznaka <script></script>
- JavaScript datoteka (ekstenzija „.js“) se importa u HTML dokument uporabom atributa „src“ u oznaci <script> *Kôd 3.1 Import JavaScript-a u HTML*.

```
<script src="javascript/js.js"></script>
```

Kôd 3.1 Import JavaScript-a u HTML

JavaScript varijable su definirani kao *mixed*, što znači da nije stroga definicija tipa kojeg ta varijabla mora biti. JavaScript, automatski mijenja vrstu varijable na osnovi podatka koji sadrži. Ako je vrijednost varijable 55 (*int*), tip te varijable će automatski biti postavljen na *int*.

Objekt je vrsta podataka koji skuplja više vrijednosti u jednu cjelinu. Određenom svojstvu u objektu se pristupa na način da se navede ime objekta, točka i na kraju ime svojstva.

```
//Komentar

funkcijaNesto("element_id"){}; // poziv funkcije

function funkcijaNesto(id){

    var i1 = document.getElementById(id);
    console.log(i1);

    if(i1){return true;}else{return false;}

}

var object = new Object();
```

Kôd 3.2 Primjer JavaScript kôda

3.3. HTML „event“

HTML „event “ je „stvar“ koja se dogodi, to jest JavaScript od reagira na taj „event“. HTML „event“ može biti nešto što sam preglednik učini (npr. Kada se stranica učita), ili nešto što korisnik učini (klikom na gumb).

```
<input type="button" value="doSomething" onclick="doSomething(this)">
```

Kôd 3.3 Primjer JavaScript event-a

3.4. Funkcija povratnog poziva (eng. callback)

Funkcija povratnog poziva je funkcija koja će se izvršiti nakon što se izvrši neka druga funkcija. U JavaScriptu funkcije su objekti, i zbog toga funkcije se mogu slati kao argumenti u druge funkcije. Funkcija koja je poslana kao argument se naziva „*callback function*“.

JavaScript će se uvijek odvijati i tragati za drugim event-ima. U slučaju da se pozivaju dvije funkcije istovremeno gdje prva funkcija čeka odgovor servera (npr. AJAX (engl. Asynchronous JavaScript and XML)) potrebnog za drugu funkciju [5].

```
funkcijaA("id"){}; // poziv prve funkcije
funkcijaB(){}; // poziv druge funkcije
```

Kôd 3.4 Primjer redosljeda izvršavanja funkcija u JavaScript

S obzirom da prvoj funkciji treba duže vrijeme da dobi odgovor, druga funkcija bi prva završila. Callback je način da se osigura da se druga funkcija tek počne odvijat nakon što prva funkcija završi *Kôd 3.5 primjer callback-a*.

```
function functionA(callback){
  console.log("a");
  callback();
}

function functionB(){
  console.log("b");
}

functionA(functionB);

/* Redoslijed odvijanja:
a
b
*/
```

Kôd 3.5 Primjer callback-a

3.5. jQuery

jQuery je posebna vrsta JavaScript biblioteke, razvijene sa namjenom da bude nadogradnja osnovnog JavaScript-a. jQuery pojednostavljuje njegovu sintaksu i omogućava bolju interakciju između JavaScript-a i drugih programskih jezika namijenjena razvoju web aplikacija [6].

Glavne prednosti JQuery-a su pojednostavljena sintaksa, kompatibilnost s više platforma i jednostavna upotreba AJAX-a.

3.6. AJAX

AJAX (eng. Asynchronous JavaScript and XML) je tehnologija koja se koristi za pozadinsku komunikaciju sa poslužiteljem. AJAX omogućava korisniku da od servera zatraži dodatne podatke bez osvježavanja cijele stranice. Sva komunikacija sa poslužiteljem izvršava se pomoću XMLHttpRequest objekta. Posljedica toga je dinamična i brza web stranica/aplikacija sa visokim stupnjem interakcije.

U praktičnom dijelu rada XMLHttpRequest pisan je u skraćenom obliku unutar jQuery tehnologije. Koristi se kao primarni način komunikacije aplikacije sa poslužiteljem i koristi se kao *routing* za aplikaciju. Glava mana je to što ne omogućava povratak na prethodnu stranicu.

Više o JavaScript-u i jQuery-u se može pronaći na W3C-u i W3Schools.

4. PHP

PHP (skraćenica za PHP: Hypertext Preprocessor) je popularan skriptni jezik otvorenog kôda (eng. open source) namijenjen za razvoj web stranica (eng. web development). Razlikuje se od klijentskog skriptnog jezika poput JavaScripta jer se izvršava na poslužitelju. Jedna od njegovih najznačajnijih mogućnosti je njegova podrška za različite baze podataka [7].

PHP je interpreterski programski jezik koji se obrađuje na poslužitelju (za razliku od JavaScripta koji se izvršava na klijentskoj strani). PHP je isključivo namijenjen za izradu web stranica i web aplikacija i ne koristi se za izradu desktop aplikacija.

4.1. Povijest PHP-a

Godine 1994. Grenlandski programer Rasmus Lerdorf izradio je jezik PHP/FI (Personal Home Page Tools / Forms Interpreter). Taj jezik bio je prethodnik PHP-u. S vremenom je dodavao funkcije iz programskog jezika C, npr. mogućnost komunikacije s bazama podataka. Kôd PHP/FI bio je javno objavljen, što znači da su ga svi slobodno mogli koristiti i sudjelovati u razvoju. Drugo izdanje PHP/FI-a (verzija 2.0) pojavilo se 1997. godine [1].

Godine 1998. Gutmans i Zeev Suraski su počeli rad na rekonstrukciji jezgre PHP-a. Ciljevi su bili poboljšati karakteristike kompleksnih aplikacija i još više povećati modularnost PHP-a. Novi PHP interpreter nazvan 'Zend Engine' (od Zeev and Andi), postigao je zadane ciljeve i predstavljen je prvi put sredinom 1999. Nova verzija, PHP 4.0, je bazirana na ovom interpreteru sa novim proširenjima. Osim poboljšanja jezgra i novih ekstenzija, značajne novosti su bile podrška za još više web poslužitelja, i sigurniji način rukovanja korisničkim podacima [8].

Godine 2004. izlazi verzija 5.0 PHP-a. Temelj ove verzije je Zend Engine 2.0 i u ovoj verziji je uvedena mogućnost objektno orijentiranog programiranja. Trenutna (datuma: 21.09.2020.) aktivna verzija PHP-a je 7.4.10 i jedna od značajnijih promjena je mogućnost određivanja tipova funkcija i parametra koje funkcija prima. U godini 2021 bi službeno trebala izaći verzija 8.0+ PHP.

4.2. Objektno orijentirano programiranje (OOP) u PHP-u

Objektno orijentirano programiranje (OOP) predstavlja pokušaj da se programi približe ljudskom načinu razmišljanja. Evolucija OOP kroz povijest se dogodila preko starijih pristupa programiranja, a to su proceduralno i modularno programiranje [9].

Postoje tri glavne tehnike programiranja, to su nestrukturirano, proceduralno i modularno programiranje.

Nestrukturirano programiranje se javlja kod početnika. Program se sastoji od samo jedne glavne „main“ funkcije. Ova tehnika je kontraproduktivna u slučaju kada program naraste. Izrazi koji se trebaju koristit više puta u programu se kopiraju. Glavna mana je nepreglednost i nečitkost samog kôda. Tu nastaje potreba da se dijelovi koji se više puta izvode odvoje u zasebnu cjelinu s načinom na koji ih se poziva i dobiva povratna informacija [10].

Proceduralni pristup programiranju zasniva se na promatranju programa kao niza funkcija, procedura, koje međusobno sudjeluju u rješavanju zadataka. Svaka procedura je konstruirana tako da obavlja jedan manji zadatak. Poziv procedure skače na mjesto u programu gdje je ona definirana te nakon što bude izvršena, program se nastavlja izvršavati od mjesta odakle je procedura pozvana [9].

Modularno programiranje sve procedure zajedničkih funkcionalnosti grupira u odvojene module. Posljedica toga je ta da je program podijeljen u više manjih dijelova koji su povezani preko poziva procedure. Svaki modul može imati svoje podatke što mu omogućava upravljanje unutrašnjim stanjima [10].

Za razliku od modularnog, kod OOP moduli grupiraju podatke i operacije nad njima te ih se predstavlja kao ključne elemente u razvoju programa, a nazivamo ih objektima. Svaki objekt ima svoje ponašanje, zadržava informaciju i može komunicirati s ostalim objektima [9].

4.2.1. Osnova OOP u PHP-u

Osnovni principi OOP-a su globalni, to jest varijable i funkcije su objedinjene u jednu cjelinu nazvanu objekt.

4.2.2. Klase i objekti

U OOP jezicima klasa je nacrt za objekte. Ona definira objekte i metode. Programski jezici moraju omogućiti kreiranje objekata iz njihove klase. Klasa ima vlastite funkcije (metode) i varijable (atribute) koje su zajedničke jednom ili skupu objekata (Ovisno koliko instanci postoji).

Klasa se stvara na način da se počinje ključnom riječi *class*, pa imenovanjem te klase. Ime klase može biti bilo koja riječ, osim ako je PHP pred definirana riječ. Standard je da naziv glavne klase unutar modula bude nazvana isto kao i modul sa velikim početnim slovom. Iza imena dolaze vitičaste zagrade unutar kojih se definiraju metode i atributi [11].

```
class SettingsController {

    private $model = null;
    private $view = null;

    protected $post;
    public $username;

    Public function __CONSTRUCT($post) {

        $this->model = new SettingsModel();
        $this->view = new SettingsView();
        $this->post = $post;
        $this->username = $_SESSION[["Username"];

    }

    protected function buildSettings(){

        $output["db"] = $this->model->selectQ1();
        $output["callback"] = true;

        return $this->view->toJSON($output);

    }

    public function build(){

        return $this->buildSettings();

    }

}
```

Kôd 4.1 Primjer klase u PHP-u

Inicijalno je postavljena vrijednost varijablama *$model* i *$view* na *null* *Kôd 4.1 primjer nasljeđivanja klase u PHP-u*.

4.2.3. Nasljeđivanje

Nasljeđivanje je bitan koncept u OOP-u, a predstavlja mogućnost izvođenja novih klasa iz postojećih.

Nasljeđivanje u OOP-u je način kreiranja novih klasa i objekata koristeći već definirane klase. Klasa koja nasljeđuje drugu klasu se naziva izvedena, a klasa koja je naslijeđena se naziva bazna klasa. Važne prednosti ovog koncepta OOP-a je iskorištavanje dosada napisanog koda

i smanjivanje kompleksnosti aplikacije. Izvedene klase preklapaju i nadopunjuju mogućnosti koje pruža bazna klasa [12].

Klase i njihove pod klase formiraju hijerarhiju klasa.

U PHP-u se klasa deklarira da je potklasa postojeće klase koristeći ključnu riječ *extends* *Kôd 4.2 Primjer nasljeđivanja klase u PHP-u*.

```
class UpdateProfilePicController extends HomeModel{}
```

Kôd 4.2 Primjer nasljeđivanja klase u PHP-u

4.2.4. Polimorfizam

Polimorfizam znači „moći poprimiti više oblika“ i važan je koncept kod OOP-a. U OOP-u polimorfizam je sposobnost varijable, metode ili objekta da preuzme više oblika.

Nadklasa sadrži metode zajedničke svim izvedenim klasama u hijerarhiji, ostavljajući mogućnost da pojedina izvedena klasa nadjača metodu svojom specifičnom implementacijom. Takve metode u izvedenoj klasi nazivaju se virtualnim metodama [13].

4.2.5. Kontrola pristupa

Kontrola pristupa je princip u OOP-u s kojim se ograničuje pristup podacima. Pristup podacima se ograničuje kako bi se osjetljivi podaci mogli sakriti i zaštititi ili kako ne bi došlo do slučajnih ili namjernih promjena.

U PHP-u postoje tri ključne riječi sa kojima se postiže kontrola pristupa:

- *public* (javna) – u ovom slučaju će metoda ili varijabla biti javna. Što znači da se može pristupiti od bilo kojeg dijela koda.
- *protected* (zaštićena) – Pristup ovoj metodi ili varijabli je omoguće samo unutar klase u kojoj se nalazi i unutar njezinih naslijeđenih klasa.
- *Private* (privatna) – privatnoj metodi ili varijabli je pristup ograničen samo u klasi u kojoj se nalazi.

Primjer kontrole pristupa u slici *Kôd 4.1 primjer klase u PHP-u*. varijabla *$username* postavljena je na *public*, varijabla *$post* na *protected*, dok su ostale varijable postavljene na *private*.

4.2.6. Varijabla *$this*

Varijabla *$this* korištena je samo unutar klasa. Ona se koristi za ukazivanje na varijablu, metodu i objekt koji poziva metodu ili varijablu unutar tog objekta.

U *Kôd 4.3 Primjer upotrebe varijable u PHP-u* koristeći varijablu *$this* vrijednost iz *id* se zapisuje u *array* pod ključem *loggedin*.

U *Kôd 4.3 Primjer upotrebe varijable u PHP-u * metoda *toJSON* (koja je definirana izvan klase u kojoj je pozvana) je pozvana iz objekta *view* koristeći varijablu *$this*.

```
$output["loggedin"] = $this->id;
$output["pageowner"] = $this->post["ProfileID"];

return $this->view->toJSON($output);
```

Kôd 4.3 Primjer upotrebe $this varijable u PHP-u

4.2.7. Magične metode

U PHP-u svaka klasa ima predefinirane metode. Predefinirane metode koje počinju znakom '__' nazivaju se magične metode. Vidljivost magičnih metoda je uvijek postavljena kao *public*. PHP sadrži veliki broj magičnih metoda, no u ovom radu će biti opisane samo najvažnije magične metode: *__construct(), __destruct(), __get(),* i *__set()*.

Instanciranjem klase automatski se poziva magična metoda *__construct* (konstruktor). Korištenje konstruktora nije obavezno. On se koristi za prijenos parametra kod stvaranja novog objekta i on ne vraća nikakvu povratnu vrijednost.

U primjeru *Kôd 4.1 primjer nasljeđivanja klase u PHP-u* konstruktor se koristi kod inicijalizacije klase kako bi postavio vrijednosti varijablama *$username* i *$post*. U varijablama *$model* i *$view* se instanciraju klase koje isto tako može ju imati svojeg konstruktora.

Magična metoda *__destruct* (destruktor) ima ulogu suprotnu konstruktoru. Ta magična metoda se automatski poziva prije uništenja objekta. On se koristi za brisanje podataka i oslobađanje memorije. Destruktor, isto kao i konstruktor, ne vraća nikakve povratne vrijednosti i ne prihvaća nikakve argumente.

Magična metoda *__get* pokreće se kada se pokušava pristupiti nedostupnoj varijabli.

Magična metoda *__set* pokreće se kada se nedostupnoj varijabli pokušava dodijeliti vrijednost.

4.2.8. Apstraktne klase i metode

Apstraktna klasa definira se ključnom riječi *abstract*. Apstraktna klasa mora sadržavati barem jednu apstraktnu metodu. Apstraktna metoda je deklarirana ali nije implementirana. Klasa koja nasljeđuje apstraktnu klasu implementirati apstraktnu metodu (isti naziv, argumenti i pravo pristupa) iz apstraktne klase.

```
Abstract class ParentClass {

    Abstract public function someMethod1($values);
    Abstract protected function someMethod2();

}
```

Kôd 4.4 Primjer apstraktne klase u PHP-u

4.2.9. Sučelja (interface)

Sučelja ne mogu biti instancirana i služe kao nacrt ili ugovor koji klasa mora poštivati. Oni služe za definiranje funkcionalnosti klase koja ih implementira. Sučelja se definiraju ključnom riječi *interface*, primjer u *Kôd 4.4 Primjer sučelja u PHP-u*.

```
interface DAOinterface {

    public function insert_($values);
    public function update_id($values, $id);
    public function delete_id($id);
    public function select_id($id);
    public function select_all();

}
```

Kôd 4.5 Primjer sučelja (sučelje DAOinterface) u PHP-u

Svaka klasa može implementirat više različitih sučelja i samo sučelje može naslijediti jedno ili više sučelja.

U sučelju se samo deklarira metode (u samom sučelju metode ne sadrže svoj blok kôda) i može ju se odrediti argumenti koje metoda prihvaća. Pravo pristupa tim metodama mora biti javno. Sučelja mogu sadržavati i konstante (*const*), ali ne mogu sadržavati varijable.

U primjeru koda *Kôd 4.5 Primjer sučelja u PHP-u* prikazano je sučelje *DAOinterface* sa deklariranim metodama *insert_*, *update_id*, *delete_id*, *select_id*, *select_all*. Svaka klasa koja implementira sučelje *DAOinterface* mora deklarirati metode koje se nalaze u sučelju.

```
class NotificationsDB implements DAOinterface{

private $c = null;
private $table;

Public function __CONSTRUCT() {

    $this->c = new DaoUser();
    $this->table = "notifications";
}

    public function insert_($values){
        // Izvršni kôd...
    }

    public function update_id($values, $id){
        // Izvršni kôd...
    }

    public function delete_id($id) {
        // Izvršni kôd...
    }

    public function select_id($id) {
        // Izvršni kôd...
    }

    public function select_all() {
        // Izvršni kôd...
    }
}
```

Kôd 4.6 Primjer implementacija sučelja u PHP-u

U primjeru koda *Kôd 4.6 Primjer implementacije sučelja u PHP-u* prikazana je klasa *NotificationsDB* koja implementira sučelje *DAOinterface* što znači da ta klasa mora implementirati metode navedene u sučelju. Način na koji će ta klasa implementirati te metode ovisi o samoj klasi.

4.3. Rad sa bazom podataka kod OOP-a u PHP-u

Bazu podataka promatramo kao zbirku podatkovnih zapisa pohranjenih na računalu koja je lako dostupna korisnicima i aplikacijama. Sastoji se od skupa međusobno povezanih podataka, pohranjenih zajedno, bez štetne ili nepotrebne zalihosti [14].

Program koji izvodi sve operacije (npr. brisanje, mijenjanje, unošenje, dohvaćanje podataka,…) nad bazom podataka naziva se sustav za upravljanje bazom podataka (eng. Database Management System) (DBMS).

Osnovne funkcionalnosti koje relacijske baze podataka mora sadržavat su takozvane CRUD (eng. *create, read, update, delete*) funkcionalnosti.

4.3.1. MySQL

MySql je poslužitelj baza podataka (*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 korisničkom imena i lozinke. [15]

MySql je jedan od najpopularnijih sustava za upravljanje relacijskim bazama podataka. Karakteristike MySQL-a:

- Pokreće se na strani poslužitelja
- Otvoreni kod (eng. *open source*)
- Podržava višekorisnički pristup bazama podataka

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. superadministrator (obično se zove root) koji ima sva administracijska prava [15].

Postoje MySQL verzije za sve važne operacijske sustave, a distribuirane su pod GPL licencom (engl. General Public Licence).

Postoje 3 osnovne vrste podataka:

- Tekstualni
- Numerički
- Datum/vrijeme

MySQL ima podršku za veliki broj programskih jezika. Jedna od popularnijih kombinacija je MySQL i programski jezik PHP. PHP ima podršku za različite baze podataka, a tome pridonosi PHP-ova objektno-orijentirana PDO (PHP Data Objects) ekstenzija.

4.3.2. PDO (PHP Data Objects)

Za ostvarivanje veze sa bazom podataka inicijalizira se novi objekt klase PDO. Potrebno je prenesti mu argumente DNS, korisničko ime i lozinku. DSN je akronim za *Database Source Name*, a prevodio bi se kao Ime izvora baze podataka. On sadrži niz znakova (eng. string) koji opisuju vezu. *Kôd 4.7 Instanca klase PDO*

```
$this->conn = new PDO(
 "mysql:host=" . $this->servername . ";dbname=" . $this->DB . "",
 $this->dbusername,
 $this->dbpassword
);

$this->conn->setAttribute(
 PDO::ATTR_DEFAULT_FETCH_MODE,
 PDO::FETCH_ASSOC
);
```

Kôd 4.7 Instanca klase PDO

Nakon što se uspostavi veza, može se započeti sa unošenjem i pretraživanjem podataka.

```
private function conn_exec($query, $data){

 $stmt = $this->conn->prepare($query);
 $stmt->execute($data);

 return $stmt;

}
```

Kôd 4.8 Izvršavanje upita

SQL upiti mogu se izvršavati pomoću metode *query*. Metoda *query* nije u potpunosti siguran naći izvršavanja upita, zato se koriste metode *prepare* i *execute*. *Kôd 4.8 Izvršavanje upita*

Nakon izvršavanja upita koriste se metode *fetch, fetchall, rowCount*,… kako bi se dohvatili podaci i spremili u varijablu.

```
 $query = "SELECT PathImg FROM backgroundimg WHERE
UserID=:userID";
 $data = array(":userID" => $this->id);
```

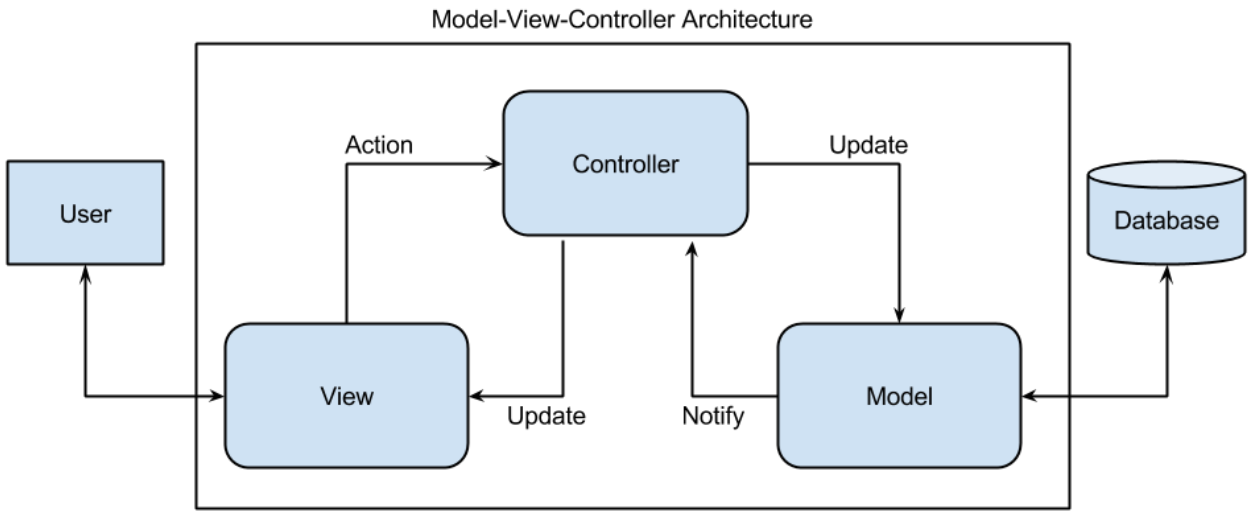
Kôd 4.9 Priprema podataka i upita za izvršavanje

U primjeru *Kôd 4.9 Priprema podataka i upita za izvršavanje* upit je spremljen u varijablu *$query*. Upit sadrži uvjet *WHERE* gdje se postavlja uvjet :*userID* na stupac *UserID*. Uvjetu se dodaje vrijednost u posebnoj varijabli koja je *array*.

Više o detaljima i specifikacijama OOP u PHP-u se može pronaći na službenim stranicama PHP-a.

5. MVC arhitektura

MVC arhitektura temelji se na razdvajanju korisničkog sučelja (ono što korisnik vidi) od poslovne logike kako bi se poboljšala organizacija i preglednost kôda. Aplikacije koje koriste MVC arhitekturu se lakše održava i nadograđuje.



Slika 5.1 MVC arhitektura

(izvor: http://www.patricksoftwareblog.com/tag/mvc/)

MVC struktura razdvaja aplikaciju na tri osnovna dijela – Model, View i Controller. Svaki dio ima svoja svojstva i zadaću koju mora izvršiti. Na kraju cijela struktura predstavlja jedan izlaz (*output*) u odnosu na zahtjeve krajnjeg korisnika [16].

MVC je prvi opisao Trygve Reenskaug. Postoji nekoliko izvedenica MVC uzorka, a jedan od najpoznatijih (zbog toga što je korišten od strane Microsofta) je Model View Presenter uzorak koji se pojavio 1990 godine i koji je bio dizajniran kao evolucija MVC uzorka, međutim Model-View-Controller i dalje ostaje u vrlo velikoj upotrebi [1].

- Prednosti MVC arhitekture:
- Lakše održavanje i nadograđivanje aplikacije
- Paralelan razvoj svake komponente (Model, View i Controller)
- Lakše testiranje svake komponente
- Fleksibilnost kod planiranja i implementiranja objekata Model, omogućena višestruka iskoristivost i modularnost.

Rezultat je konfigurabilna i podatna aplikacija s grafičkim sučeljem.

MVC arhitektura se primjenjuje u slučaju kada se želi odvojiti logika programa od korisničkog sučelja i prikaza. Najčešće se primjenjuje kod izrade složenijih aplikacija.

5.1. Model

U MVC arhitekturi, model je taj koji je zadužen za rad sa podacima i bazom podataka. On izvršava upite (dohvata podataka iz baze, zapis podataka u bazu,…).

Model je taj koji sadrži funkcije za manipulaciju podataka.

5.2. View (pogled)

View (pogled) je zadužen za sav prikaz u aplikaciji. On se bavi svim grafičkim dijelovima aplikacije. On uzima podatke od upravitelja (*controller*), te ih formatirane šalje u preglednik. Po modelu može biti više pogleda (*view*), gdje svaki ima svoju svrhu. Pogled može biti u sljedećim oblicima: HTML, YHTML, XML/XSLT, Windows forma, itd.

5.3. Controller (upravitelj)

Controller je zadužen 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 [16].

5.4. Uporaba DAO arhitekture u MVC arhitekturi

Radi lakše organizacije kôda i daljnje mogućnosti ponovne uporabe kôda u velikim aplikacijama dobro je primijeniti druge arhitekture i principe. DAO (eng. *Data Access Object*) arhitektura se može primjenjivati uz MVC arhitekturu.

5.4.1. DAO arhitektura

DAO arhitektura se temelji na odvajanju poslovne logike od poslovnih podataka. Za svaku tablicu u bazi izrađuje se posebna klasa koja barata upitima te tablice.

Izrađuje se generičko sučelje koje sadrži metode za ostvarivanje funkcionalnosti CRUD (*Create, Retrieve, Update, Delete*) nad podacima u bazi. To generičko sučelje može se implementirati na sve DAO klase ili se može izraditi posebna generička klasa. Ta generička klasa bi sadržavala osnovne CRUD funkcionalnosti koje bi se mogle primijeniti na svaku tablicu. Za

kompleksniju manipulaciju nad pojedinačnu tablicu potrebno bi bilo izraditi klasu specifičnu samo za tu tablicu.

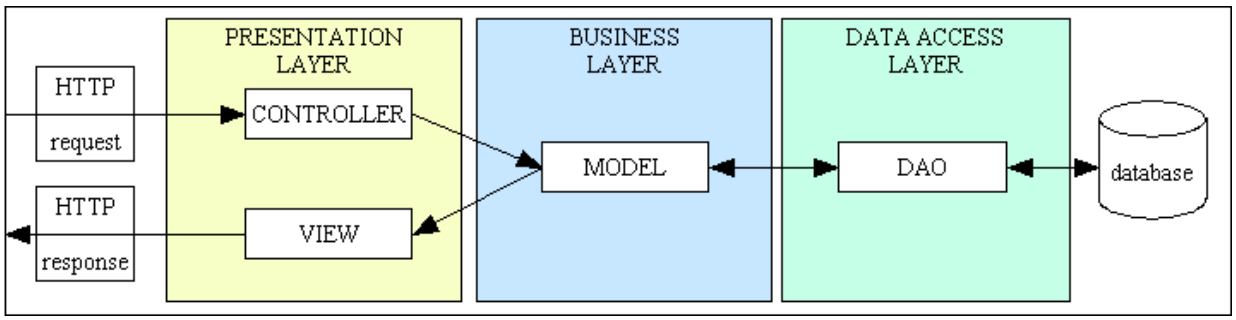
U jezicima sa strogo definiranim tipovima varijabla, uz MVC ili DAO arhitekturu bi se koristila i DTO (eng. *Data Transfer Object*) arhitektura. DTO se temelji na tome da se izrade klase nazivom tablice koje reprezentiraju. Ta tablica bi bila popunjena varijablama istog naziva i tipa kao stupci tablice. Dohvaćeni podaci bi se preslikali u listu od klase (postupak *auto mapping*). Osim toga, DTO može biti bilo koji modul koji ne sadrži logiku. Namjena takvo modula je prijenos podataka.

Potrebno je napomenuti da bi se kod većih projekata na kojima radi veći broj ljudi potrebno koristiti DTO arhitekturu.

5.5. MVC – DAO

Glavni smisao DAO arhitekture je mogućnost ponovne upotrebe kôda. Stoga svaki puta kada se treba izvršiti upit za određenu tablicu se samo instancira DAO klasa zadužena za tu tablicu.

Upravitelj podatke šalje modelu. Ako je potrebno model će izraditi *WHERE* dio upita i proslijedit će uvjete u DAO klasu. Ta klasa će izvršiti upit i vratiti povratnu informaciju.



Slika 5.2 MVC i DAO arhitektura

(izvor: https://www.tonymarston.net/php-mysql/model-view-controller.html)

Ovaj princip omogućava da se u modelu upit izvrši u samo jednu do tri linije koda u slučaju da se ne radi o kompleksnom upitu ili ako nema potrebe za kalkulacijama i provjerama podataka.

5.6. MVC – Service – DAO

U slučaju MVC – Service – DAO, poslovna logika se premješta iz modela u *service. Service* će dalje proslijediti sve potrebne podatke Dao klasi koja će izvršiti upit.

5.7. M(V)C u API dizajnu

Aplikacija koja prati MVC arhitekturu ne mora nužno koristiti sve komponente te arhitekture. Aplikacija može koristiti samo upravljača i model (MC). U tom slučaju bi se izgradila zasebna aplikacija koja bi se ponašala kao pogled. Potrebno je napomenuti da bi i ta aplikacija mogla sadržavati sve tri komponente MVC arhitekture.

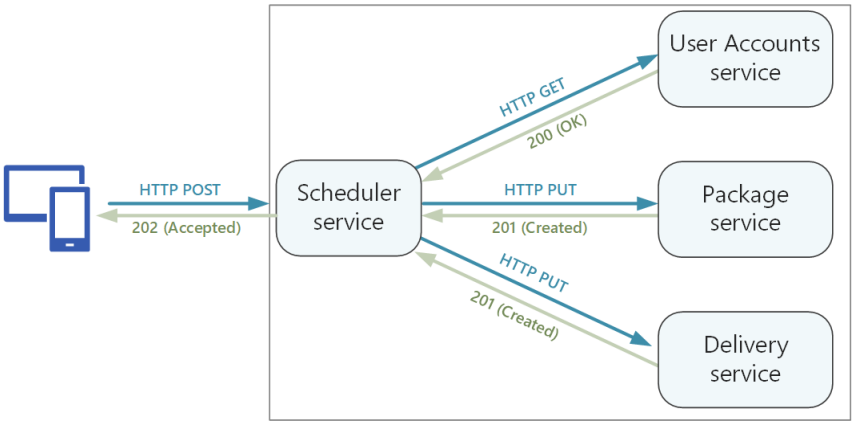
5.7.1. API dizajn

API (eng. *application programming interfaces*) je skup definicija i protokola za izgradnju i integraciju aplikacijskog software-a [17].

Glavna prednost API-a je to što omogućava komunikaciju sa drugim aplikacijama bez potrebe da se zna kako je su implementirane. Glavna prednost toga je što olakšava development i mogućnost uštede na vremenu i novcu.

API-i se ponašaju kao ugovori između dvije aplikacije. Aplikacija 1 šalje unaprijed definiran zahtjev koji aplikacija 2 prepoznaje i vraća odgovor ovisan o zahtjevu [17].

API-i olakšavaju integraciju nove aplikacije u već postojeću arhitekturu.



Slika 5.3 API

(izvor: https://docs.microsoft.com/en-us/azure/architecture/microservices/design/api-design)

Primjer *Slika 5.3 API* gdje korisnik sa web, mobilne ili *windows* aplikacije šalje zahtjev aplikaciji na serveru. Ta aplikacija će tada potvrditi korisnika i ovisno o zahtjevu će vratiti povratnu informaciju. Dalje je prepušteno aplikaciji koja je poslala zahtjev da radi što hoće sa primljenim podacima. Podaci se najčešće šalju HTTP metodama (GET, POST, PUT, PATCH, DELETE) kao JSON format, a vraćaju se najčešće kao JSON ili XML format.

5.7.2. Middleware

Middleware je software koji pruža uslugu aplikaciji koja je izvan dosega te aplikacije. On pruža način komunikacije između dvije aplikacije koje nemaju nikakav direktan međusoban način komuniciranja.

Middleware-i se koriste kako bi se brže i učinkovitije izradile aplikacije. Najbolji primjer middlewere-a je software za *authentication.*

5.7.3. MC arhitektura

Aplikacija koja se sastoji samo od upravljača i model zadužena je za interakciju sa podacima. Ona prima podatke od drugih aplikacija i ovisno o zahtjevu vraća povratne podatke.

Upravljač prima zahtjev (npr. neku od CRUD funkcionalnosti), poziva se odgovarajuća funkcija iz modela, i vraćaju se povratni podaci (potvrda o zapisanim podacima u bazu). Sa tim povratnim podacima će dalje baratati aplikacija koja je poslala zahtjev.

Najbolji primjer ovog pristupa je odvajanje frontend-a i backend-a. Aplikacija zadužena za prikaz (web, windows ili mobilna aplikacija) će izgraditi ono što korisnik vidi ovisno o povratnim informacijama od aplikacije koja je zadužena samo za interakciju sa bazom.

5.8. SOLID princip

„Dobro napisani kôd ne treba komentare“ je uzrečica koja često ide uz SOLID princip. Termin SOLID je iskova Robert C. Martin, poznati kao „Uncle Bob“. SOLID princip je zapravo skup 5 različitih principa koji zajedno olakšavaju programeru da izradi *softwre*. Korištenjem tog principa se olakšava izrada i održavanje kôda i istovremeno se smanjuje kompleksnost kôda.

SOLID je akronim koji predstavlja sljedeće principe:

- S – Single Responsibility
- O – Open/Closed
- L – Liskov Substitution
- I – Interface Segregation
- D – Dependency Inversion

ASP.NET (ASP.NET CORE) su radni okviri za C# koji podupiru objektno orijentirani pristup programiranju sa implementacijom SOLID principa sa MVC i DTO arhitekturom.

5.8.1. S – Single Responsibility Principle (SRP)

SRP tvrdi da svaki modul treba imati samo jedan razlog da se promjeni. Što znači da svaka klasa u kôdu treba imati samo jedan posao da obavi. Sve u toj klasi mora biti posvećeno i povezano sa tim poslom. Ovo ne znači da ta klasa mora imati samo jednu metodu ili varijablu. Može ih imati više, sve dok su povezane sa tim ciljem [20].

Ovaj princip potiče na razmišljanje na koje sve načine bi se klasa mogla mijenjati.

5.8.2. O – Open/Closed

Ovaj princip tvrdi da modul/klasa je omogućena za proširivanje, ali onemogućena za modifikacije. Što znači da je klasa izrađena na način da joj se nove funkcionalnosti mogu dodavat po potrebi, ali ono što je već izrađeno i prešlo testiranje (*unit testing*) se ne smije mijenjat osim ako ne postoji neka greška (*bug*) koja te treba popraviti. Dobar primjer iz života ovog principa je to što si čovjek ne mora skidati kožu svaki put kada si želi obleći kaput.

5.8.3. L – Liskov Substitution

Ovaj princip tvrdi da se može koristiti bilo koja podređena klasa u zamjenu za bilo koju drugu podređena klasu bez potrebe da se modificira bazna klasa. Time se osigurava da podređena klasa nema utjecaj na baznu klasu.

Primjer ovoga iz pravog života: Otac koji je doktor ima sina koji želi biti nogometaš. U ovom slučaju sin ne može zamijeniti bez obzira na to da spadaju istoj obitelji.

5.8.4. I – Iterface Segregation Principle

Ovaj princip tvrdi da klasa ne bi trebala implementirati dio sučelja koji ne koristi. Umjesto jednog velikog sučelja potrebno je izraditi više manjih i organizirati ih po nekim grupama Sučelje mora reprezentirati kôd koji ga koristi a ne kôd koji ga implementira. Klasa ne smije biti natjerana da ovisi o sučeljima koje ne koristi [20].

Isto kao klase i sučelja trebaju imati samo jedan smisao/cilj (SRP). Klasa ne smije implementirati sučelje koje ne dijeli istu svrhu sa njom. Što veće sučelje, to veća šansa da sadrži nešto što klasi koja ga implementira ne treba.

5.8.5. D – Dependency Inversion Principle

Ovaj princip tvrdi da moduli/klase veće razine ne bi smjele ovisiti o modulima/klasama niže razine. Svaki modul i klasa bi se trebali temeljit na apstrakciji. Potrebno je napomenuti da apstrakcija ne bi smjela ovisiti o detaljima.

Klasa veće razine će sadržavati logiku aplikacije, dok klase niže razine će baratati detaljima. Npr. u MVC-u upravljač poziva na model da dohvati nešto iz baze. Upravljaču dalje ne mora biti bino dali je Model dohvatio osamdeset redova ili nula. On će to dalje proslijediti pogledu koji će bez obzira na to što je dohvaćeno od baze izgraditi izgled.

Klase više razine ne bi trebale poznavati mnogo o klasama niže razine sa kojima ima interakciju. Kada klasa poznaje mnogo o implementaciji druge klase, tu dolazi do opasnosti da izmjenom te druge klase bi se mogla slomiti prva klasa. Zato je bolje temeljiti klase na apstrakciji nego da ovise jedna o drugoj.

5.8.6. SOLID princip i MVC

Korištenjem SOLID principa uz MVC arhitekturu rastereću odgovornost sa MVC modula na mnogo manjih. Manje bitnije stvari će se rasporediti će se u određene kategorije i module. To će još više poboljšati čitkost i održivost koda.

Daljnjom apstrakcijom tih modula i klasa će se olakšati ponovna iskoristivost kôda.

SOLID princip je jedan od načela programiranja kojeg bi sve svaki programer trebao držati ili u najgorem slučaju nekoliko od tih 5 pod-principa koji stvaraju SOLID princip.

6. PHP radni okviri (eng. frameworks) sa podrškom MVC arhitekture

Neki od popularnijih PHP radnih okvira koji će ju biti opisani u nastavku: Laravel, Symfony i PHP Mini.

6.1. Radni okviri

Radni okvir je alat koji se sastoji od mnoštva različitih komponenti koje programeru mogu olakšati posao i ubrzati proces izrade aplikacije. Okviri su kolekcije različitih pomoćnih funkcija, klasa, knjižnica i dr. koje pomažu nudeći već gotova rješenja za specifične i često korištene probleme. Programeri se mogu odlučiti za korištenje već popularnih gotovih rješenja, no ponekad mogu krenuti i u izradu vlastitog [18].

6.2. Laravel

Laravel je radni okvir temeljen na MVC arhitekturi za PHP. On je izrađen kako bi programerima olakšao početak na PHP projektima.

Laravel je jednostavan za instalaciju. Može se ga preuzeti sa web stranice www.laravel.com te je besplatan i slobodan za korištenje.

Novi Laravel projekt stvara se komandom „*Laravel New ime-projekta"*. Unutar novo kreirane mape *ime-projekta* kreirane su sljedeće mape: *app*, *bootstrap*, *config*, *database*, *public*, *resources*, *routes*, *storage*, *tests* i *vendor*. Uz njih su generirani određeni drugi moduli.

Mapa *routs* sadrži modul nazvan *web.php,* On je zadužen za čitanje i kreiranje svih URL-ova. Unutra putanje *ime-projekta/app/http* nalazi se mapa *controllers*. U toj mapi se nalaze svi upravljači za aplikaciju.

Unutar putanje *ime-projekta/resuorces* se nalazi mapa *view* unutar koje se nalaze svi moduli zaduženi za pogled.

Za pristup na bazu podataka, prvo je potrebno unutar *.env* datoteke zapisati onsnovne informacije (korisničko ime, lozinka, baza,…) o pristupu na bazu. U putanje *Illuminate/Database/Eloquent/Model* nalazi se primarni model kojeg svi drugi *extend*-aju. Moduli nazvani *migration* su zaduženi za stvaranje i modificiranje tablica u bazi.

6.3. Symfony

Symfony je PHP radni okvir temeljen na MVC arhitekturi. On je otvorenog kôda i tisuće web stranica i aplikacija ga koriste kao temelj. Čak i drugi PHP radni okviri kao Drupal i Laravel koriste komponente Sympfony-a. Laravel kôd sadrži oko 30% kôda od Symfony.

Symfony je set komponenta koje je lako ponovno koristit. Svaka od tih komponenta je testirana u brojnim projektima i može ju se koristiti u bilo kojem projektu odvojeno od Symfony radnog okvira.

Razlozi zašto koristiti Symfony radni okvir:

- Reputacija – profesionalci su ga brzo počeli koristiti nakon što je izašao. Danas je jedan od najprepoznatijim radnim okvirom za PHP i ima veliku zajednicu [20]
- Reference – tisuće web stranica i aplikacija koriste Symfony [20]
- Sredstva – Lako je naći odgovore na pitanja zbog velike zajednice koju ovaj radni okvir ima. Postoje i mnogo radova koji pomaže ju pri korištenju ovog radnog okvira [20]
- Fleksibilnost – Ovaj radni omogućuje lako korištenje samo nekoliko svojih dijelova bez potrebe da se koristi radni okvir [20]

Ovaj radni okvir se lako može preuzeti s poveznice: https://symfony.com/download

6.3.1. Struktura Symfony-a

Glavna struktura Symfony projekta se sastoji od sedam mapa, no uglavnom su korištene samo tri mape (*config*, *src* i *template*). Mapa *template* sadrži sav HTML projekta. Unutar mape *config* se nalazi mapa *routs* unutar koje se definiraju URL rute projekta. Unutar mape *src* se odvija cijela logika projekta. Inicijalno ona sadrži samo mapu *Controller* i datoteku *Kernel.php*. Korisnik ručno dodaje sve druge mape (npr. *Models*, *Services*…) unutar mape *src*.

Unutar mape *public* se nalazi *indeks.php*.

6.4. PHP Mini

PHP MINI je razvojni okvir za razvoj PHP aplikacija. On je jednostavan i laki za koristit kostur PHP aplikacije. On nije profesionalni radni okvir i ne sadrži neke stvari koje pravi radni okviri sadrže.

PHP MINI je jednostavan za instalaciju i može se preuzeti sa GitHub-a iz poveznice https://github.com/panique/mini. On je besplatan i slobodan za korištenje. Potrebno je napomenuti da je on izrađen za verzije PHP 5.3.0 i veće, te baratenjem MySQL bazom podataka.

Dva su glavna dijela aplikacije koja koristi ovaj radni okvir: *public* i *aplication*. Unutar mape *public* se nalaze HTML, CSS (SCSS) i JavaScript datoteke. Uz njih se nalaze i drugi resursi kao slike i fontovi. Tu se nalazi i glavna datoteka *indeks.php* unutar koje se renderira sav prikaz aplikacije.

Unutar mape *aplication* nalazi se sva logika zadužena za aplikaciju. Osnovna aplikacijska struktura sastoji se od sljedećih dijelova:

- Config - definirane konstante potrebne za spoj na bazu podataka
- Controller- tu se nalaze svi upravljači koji se pozivaju prema potrebi aplikacije)
- Core - izvršava spoj na bazu i komunikaciju između svih ključnih dijelova MVC arhitekture
- Model - tu se nalaze svi model-i
- View - tu se nalaze svi moduli zaduženi za prikaz

7. Izrada chat aplikacije (praktični dio rada)

Za praktični dio završnog rada izradio sam PHP aplikaciju za slanje poruka, pronalaženje ljudi i izradu personaliziranog profila. Većina praktičnog dijela će se fokusirati na izradi dijela aplikacije zaduženog za slanje i primanje poruka *Chat*.

Aplikacija ima sljedeće funkcionalnosti:

- Mogućnost izrade personaliziranog profila
- Mogućnost personalizirana profila (profilna slika, opis, objave)
- Mogućnost filtriranja postojećih korisnika ovisno o zadanim parametrima
- Mogućnost slanja poruka prijateljima

Aplikacija se može pronaći na poveznici: http://arwen.unin.hr/~dapetrovic/MVC/

Potrebno je napomenuti da se aplikacija temelji na AJAX pozivima. Mana toga je što se gubi sposobnost osvježivanja i pamćenje povijesti (sve se odvija unutar *indeks.php* datoteke). Prednost ovog pristupa je to što se ne treba razmišljat o *routing-u. Chat* dio aplikacije koristi MVC arhitekturu u kombinaciji sa Service i DAO arhitekturama i donekle prati SOLID principe.

Ostatak aplikacije prati MC arhitekturu gdje je prikaz izrađen na strani korisnika.

Aplikacija je izrađena u sljedećim koracima:

- Izrada baze podataka
- Izrada direktorija i strukture projekta
- Dizajn aplikacije
- Spajanje na bazu podataka
- Korištenje DAO arhitektura
- Podjela odgovornosti po mapama/klasama za *Chat*
- AJAX poziv

7.1. Izrada baze podatak

Stvorena je baza podataka pod imenom *pa3* u MySQL jeziku koristeći phpMyAdmin sučelje. To je besplatno sučelje za upravljanjem bazom podataka u PHP-u.

Unutar baze *pa3* stvorene su 12 tablica.

Tablica *users* ima sedam stupca. Stupac *UserID* je primarni ključ tipa BIGINT i vrijednost je postavljena na AUTO_INCREMENT. Stupac *Username* sadržava korisničko ime, dok stupac *Mail* sadržava elektronsku poštu korisnika. Stupac *Salt* sadrži ključ za šifriranje lozinke, dok stupac *PasswordENC* sadrži šifriranu korisničku šifru. Stupac *Date* sadrži datum registracije korisnika.

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	UserID	bigint(20)			No	*None*	AUTO_INCREMENT
2	Username	varchar(30)	utf8mb4_general_ci		No	*None*	
3	Mail	varchar(85)	utf8mb4_general_ci		Yes	*NULL*	
4	Salt	varchar(300)	utf8mb4_general_ci		No	*None*	
5	PasswordENC	varchar(300)	utf8mb4_general_ci		No	*None*	
6	Date	datetime(6)			No	CURRENT_TIMESTAMP(6)	
7	Prava	int(11)			No	0	

Slika 7.1 Tablica users

Tablica *usersdetail* se sastoji od pet stupaca. Stupac *id* je primarni ključ tablice, dok stupac *UserID* je strani ključ na primarni ključ iz tablice *users*. Ostali stupci zapisuju ime, prezime i grad korisnika.

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	id	bigint(20)			No	*None*	AUTO_INCREMENT
2	UserID	bigint(20)			No	*None*	
3	FirstName	varchar(85)	utf8mb4_general_ci		No	*None*	
4	LastName	varchar(85)	utf8mb4_general_ci		No	*None*	
5	City	varchar(85)	utf8mb4_general_ci		No	*None*	

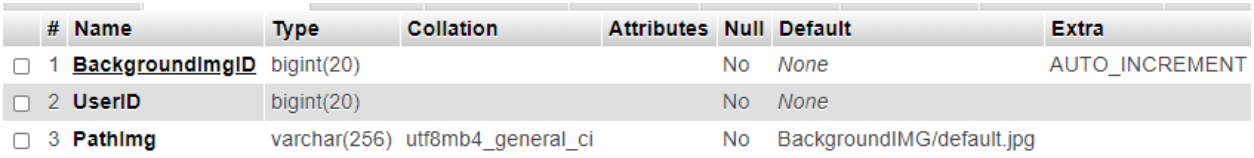
Slika 7.2 Tablica usersdetail

Tablica *userssettings* se sastoji od šest stupaca. Isto kao i u prošloj tablici ima stupce *id* i *UserID*. Ostali stupci sadrže brojčane vrijednosti koje se odnose na broj prikaza nečega (npr. koliko će se poruka u *chat*-u prikazat).

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	id	int(11)			No	*None*	AUTO_INCREMENT
2	UserID	bigint(20)			No	*None*	
3	UsersShownPerPage	int(11)			No	10	
4	ShowWallPosts	int(11)			No	10	
5	ShowNotifications	int(11)			No	10	
6	ShowMsg	int(11)			No	35	

Slika 7.3 Tablica userssettings

Tablica *backgroundimg* se sastoji od tri stupca. Stupac *BackgroundImgID* je primarni ključ tablice. Isto kao i u prošloj tablici i ova tablica ima stupac *UserID*. U stupac PathImg se zapisuje putanja slike za pozadinu profila.



#	Name	Type	Collation	Attributes	Null	Default	Extra
1	BackgroundImgID	bigint(20)			No	*None*	AUTO_INCREMENT
2	UserID	bigint(20)			No	*None*	
3	PathImg	varchar(256)	utf8mb4_general_ci		No	BackgroundIMG/default.jpg	

Slika 7.4 Tablica backgroundimg

Tablica *profileimg* je slična kao i prethodna tablica. Sastoji se od tri stupca. Stupac *ProfileImgID* je primarni ključ tablice, stupac *UserID* je strani ključ, dok stupac *PathImg* zapisuje putanju profilne slike.

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	ProfileImgID	bigint(20)			No	*None*	AUTO_INCREMENT
2	UserID	bigint(20)			No	*None*	
3	PathImg	varchar(256)	utf8mb4_general_ci		No	ProfileIMG/default.png	

Slika 7.5 Tablica profileimg

Tablica *descriptionposts* se sastoji od 3 stupca. Kao primarni ključ se koristi stupac *DescID*. Stupac *UserID* je strani ključ, dok stupac *DescTxt* sadrži opisni tekst korisničkog profila.

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	DescID	bigint(20)			No	*None*	AUTO_INCREMENT
2	UserID	bigint(20)			No	*None*	
3	DescTxt	varchar(1800)	utf8mb4_general_ci		Yes	*NULL*	

Slika 7.6 Tablica descriptionposts

Tablica *wallposts* se koristi za zapis objava korisnika na svojem profilu i na profilima drugih korisnika.. Ova tablica se sastoji od pet stupaca. Stupac *PostID* je primarni ključ tablice, dok stupac *UserID* je strani ključ iz tablice *users* (na stupac *UserID*) koji označuje na kojem se profilu nalazi objava. Stupac *TxtPosterID* je isto strani ključ iz tablice *users* (na stupac *UserID*) koji označuje tko je vlasnik objave. Stupac *CommentValue* sadrži tekst objave, a stupac *UpdateDate* sadrži datum kada je zadnji put izmjena bila napravljena na objavi.

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	**PostID**	bigint(20)			No	*None*	AUTO_INCREMENT
2	**UserID**	bigint(20)			No	*None*	
3	**TxtPosterID**	bigint(20)			No	*None*	
4	**CommentValue**	varchar(900)	utf8mb4_general_ci		Yes	*NULL*	
5	**UpdateDate**	datetime			No	CURRENT_TIMESTAMP	

Slika 7.7 Tablica wallposts

Tablica *notifications* se sastoji od devet stupaca i služi u svrhu baratanja obavijestima. Stupac *NotificationID* služi kao primarni ključ tablice. Stupci *UserID_toNotifie* i *UserID_didAction* su strani ključevi iz tablice *users* na stupac *UserID.* Stupac *UserID_toNotifie* se odnosi na korisnika kojega se treba obavijestiti o akciji, dok stupac *UserID_didAction* je korisnik koji je napravio tu akciju. Stupac *NotifieAction* se odnosi na to dali je korisnik označio obavijest kao pročitanu, dok stupac *DoesCount* se odnosi na to dali je obavijest viđena. Stupac *NotifieTypeID* zapisuje vrijednost od stranog ključa na kojeg će se odnositi, dok stupac NotifieTypeTable zapisuje tablicu na koju će se stupac *NotifieTypeID* odnositi. Stupac *NotifieTypeMsg* zapisuje poruku obavijesti.

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	**NotificationID**	bigint(20)			No	*None*	AUTO_INCREMENT
2	**ChangeDate**	datetime			No	CURRENT_TIMESTAMP	
3	**UserID_toNotifie**	bigint(20)			No	*None*	
4	**UserID_didAction**	bigint(20)			No	*None*	
5	**NotifieAction**	tinyint(1)			No	1	
6	**DoesCount**	tinyint(1)			No	1	
7	**NotifieTypeID**	bigint(20)			No	*None*	
8	**NotifieTypeTable**	varchar(50)	utf8mb4_general_ci		No	*None*	
9	**NotifieTypeMsg**	varchar(450)	utf8mb4_general_ci		No	*None*	

Slika 7.8 Tablica notifications

Tablica *friends* se sastoji od šest stupaca. Stupac *FriendID* je primarni ključ tablice, dok *UserID_a* i *UserID_b* su strani ključevi iz tablice *users* (na stupac *UserID*). Stupci *Active_a* i *Active_b* služe za označavanje odnosa između dva korisnika (npr. korisnik *a* je blokirao korisnika *b*). Stupac *LastUpdated* služi za bilježenje datuma zadnje izmjene.

	#	Name	Type	Collation	Attributes	Null	Default	Extra
☐	1	**FriendID**	bigint(20)			No	*None*	AUTO_INCREMENT
☐	2	**UserID_a**	bigint(20)			No	*None*	
☐	3	**UserID_b**	bigint(20)			No	*None*	
☐	4	**Active_a**	int(11)			No	0	
☐	5	**Active_b**	int(11)			No	0	
☐	6	**LastUpdated**	datetime			No	CURRENT_TIMESTAMP	

Slika 7.9 Tablica friends

Tablica *chatmsg* sastoji se od osam stupaca. Stupac *ChatID* je primarni ključ tablice, a stupac *InsertDate* sadrži datum kada je poruka poslana. Stupac FriendTypeID je strani ključevi iz tablice *friends* (na stupac *FriendsID*). *UserID_owner* i *UserID_friend* su stupci stranog ključa iz tablice *users* (na stupac *UserID*). Stupac *UserID_owner* se odnosi na korisnika koji je poslao poruku, dok stupac *UserID_friend* se odnosi na korisnika koji prima poruku. Stupac *Msg* sadrži kriptiranu poruku. Stupac *MsgNumb* sadrži redni broj poruke, dok stupac *MsgEncKey* sadrži unikatan ključ za dešifriranje poruke.

	#	Name	Type	Collation	Attributes	Null	Default	Extra
☐	1	**ChatID**	bigint(20)			No	*None*	AUTO_INCREMENT
☐	2	**InsertDate**	datetime(2)			No	CURRENT_TIMESTAMP(2)	
☐	3	**FriendTypeID**	bigint(20)			No	*None*	
☐	4	**UserdID_owner**	bigint(20)			No	*None*	
☐	5	**UserdID_friend**	bigint(20)			No	*None*	
☐	6	**Msg**	varchar(500)	utf8mb4_general_ci		No	*None*	
☐	7	**MsgNumb**	bigint(20)			No	*None*	
☐	8	**MsgEncKey**	varchar(75)	utf8mb4_general_ci		No	*None*	

Slika 7.10 Tablica chatmsg

7.2. Izrada direktorija i strukture projekta

Struktura projekta:

- Korijenska mapa
  - About
  - BackgroundIMG
  - Chat
  - Css
    - ❖ Style.css
  - DB
    - DAO
    - ❖ Daointerface.php
    - ❖ DbRoot.php
  - Dokumentacija
  - FindFriends
  - Home
  - Javascript
    - ❖ builds.js
    - ❖ jquery-3.5.1.min.js
    - ❖ js.js
  - LogIn
    - Controllers
    - Models
    - Views
  - Pics
  - ProfileIMG
  - Requests
  - Utils
  - ❖ HtmlExtras.php
  - ❖ HtmlHead.php
  - ❖ index.php
  - ❖ Init.php
  - ❖ SignOut.php

U korijenskoj mapi nalazi se datoteka *index.php* koja je zadužena za prikaz cijelog HTML-a aplikacije. Taj pristup se naziva *single page application*. Datoteka *HtmlExtras.php* sadrži zajedničke HTML elemente prikaza kao što je *footer* ili *modal window*. Datoteka HtmlHead.php sadrži sve atribute koji idi u HTML tag: *head*.

Datoteka *Init.php* je zadužena za inicijalizaciju aplikacije. Provjerava dali je korisnik prijavljen i ovisno o tome će učitat formu za login ili profil korisnika. Nakon što korisnik napravi re-direkciju na datoteku *SignOut.php*, ona će uništiti *Session* i napraviti re-direkciju na datoteku *indeks.php*.

Mape *BackgroundIMG* i *ProfileIMG* će sadržavati sve slike koje korisnici upload-aju. Mapa *pics* će sadržavati sve slike korištene u aplikaciji (npr. ikone za Facebook i Instagram). Mapa *Dokumentacija* će sadržavati dokumentaciju rada u *.docx* i *.pdf* formatu.

Mapa *Css* će sadržavati sve *.css* datoteke aplikacije. Sadrži datoteku *Style.scc* koja je zadužena za stiliziranje aplikacije.

Mapa *Javascript* sadrži sve *.js* datoteke. Datoteka *jquery-3.5.1.min.js* je knjižnica koja omogućuje korištenje jQuery-a, dok *js.js* je zadužena za svu logiku na korisničkoj strani. Datoteka *build.js* je zadužena za baratanjem izgleda stranice.

Unutar mape *Utils* će se nalaziti sve datoteke koje su često korištene preko cijele aplikacije, a nemaju nikakvu specijalnu klasifikaciju.

Mapa *LogIn* sadrži sve dijelove aplikacije zadužene za prijavu i registraciju.

Mape *About*, *Chat*, *FindFriend*, *Home* i Requests sadržavaju više mapa i datoteka zadužene za dio aplikacije za koje su vezane.

Struktura *Chat* mape:

- Chat
  - Controllers
    - ChatController.php
  - Encryption
    - SimpleChatEncryption.php
  - Error
    - ErrorFormat
      - ErrorFormat.php
    - ChatError.php
  - Imports
    - ChatControllerImports.php
  - Models
    - ChatModel.php
  - Services
    - ChatMsgService.php
    - FriendsService.php
    - NotificationsService.php
    - UsersService.php
  - Session
    - ChatSession.php
  - Utils
    - DateFormating.php
    - InputFormating.php
    - RandomGenerator.php
  - Validators
    - ChatInputValidator.php
    - ChatPostValidator.php
  - Views
    - ChatView.php

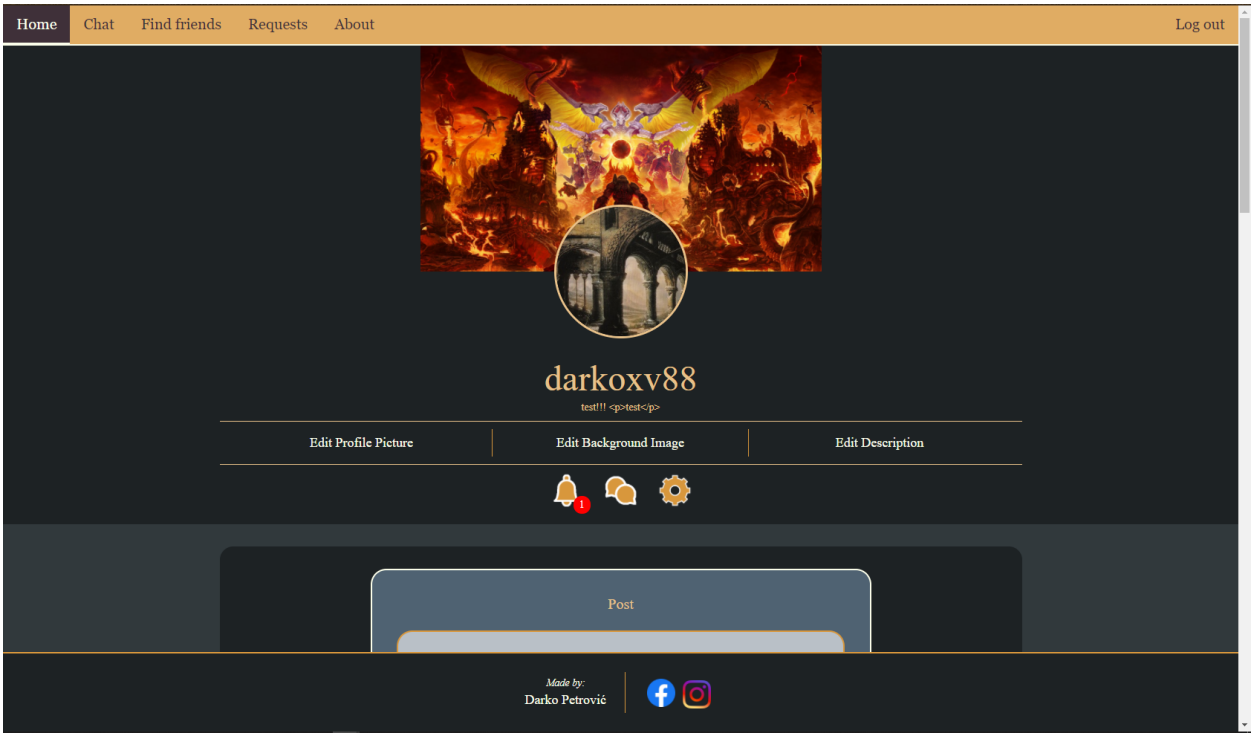
U poglavlju 7.4 ovog rada će se detaljnije opisati svrha svake datoteke vezane za *chat*.

7.3. Dizajn aplikacije

Dizajn aplikacije rađen je od nule u *css*-u. Odabrane su pet boje na kojima se temelji izgled stranice:

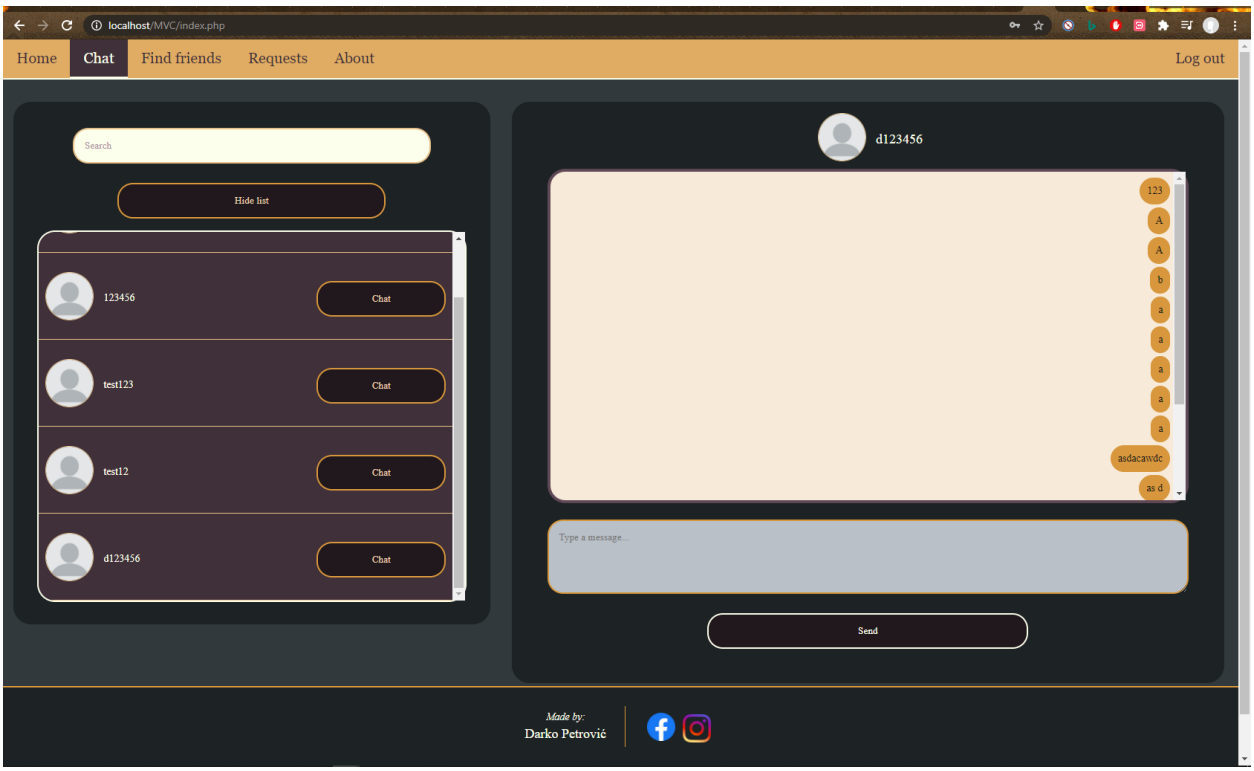
- rgba(49, 57, 60, 1)
- rgba(79, 98, 114, 1)
- rgba(216, 151, 60, 1)
- rgba(158, 120, 143, 1)
- rgba(252, 255, 235, 1)

U slici *Slika 7.11 Dizajn korisničkog profila* je prikazan dizajn korisničkog profila i okvirni dizajn aplikacije.



Slika 7.11 Dizajn korisničkog profila

U slici *Slika 7.12 Dizajn Chat-a* prikazan je dizajn *Chat*-a.



Slika 7.12 Dizajn Chat-a

7.4. Spajanje na bazu podataka

Za rad sa bazom podataka, prvobitno se je potrebno spojiti na poslužitelja baze podatak i odabrati bazu. U PHP-u se to ostvaruje pomoću PDO-a *Kôd 4.7 instanca klase PDO*.

```
class DbRoot {
 private $servername;
 private $dbusername;
 private $dbpassword;
 private $DB;
 public $conn;

 protected function __CONSTRUCT() {
  $this->servername = "localhost";
  $this->dbusername = "root";
  $this->dbpassword = "";
  $this->DB = "mvc";
  try {
   $this->conn = new PDO(
    "mysql:host=" . $this->servername . ";dbname=" . $this->DB . "",
    $this->dbusername, $this->dbpassword
   );
   $this->conn->setAttribute(
    PDO::ATTR_DEFAULT_FETCH_MODE,
    PDO::FETCH_ASSOC
   );}
  catch (PDOException $e) {
```

```
   echo "Error!: " . $e . " ...Connection failed";
  }
 }
}

class DaoUser extends DbRoot {

 public function __CONSTRUCT() {
  DbRoot::__CONSTRUCT();
 }

 private function conn_exec($query, $data){
  $stmt = $this->conn->prepare($query);
  $stmt->execute($data);
  return $stmt;
 }
 public function conn_fetch($query, $data) {
  $stmt = $this->conn_exec($query, $data);
  $row = $stmt->fetch();
  $stmt->closeCursor();
  return $row;
 }
 public function conn_fetchall($query, $data) {
  // izvršni kod
 }
 public function conn_rowCount($query, $data) {
  // izvršni kod
 }
 public function conn_insertedID($query, $data) {
  // izvršni kod
 }
}
```

Kd 7.1 Klase koje izvršavaju konekciju na bazu

Klasa DbRoot je zadužena za ostvarivanje konekcije na bazu pri instanciranju *Kôd 7.1 Klase koje izvršavaju konekciju na bazu *. Kod instanciranja klase DaoUser, automatski se instancira klasa DbRoot. Klasa DaoUser sadrži metodu *conn_exec* koja prima parametre *$query* i *$data*. Ona izvršava upit i vraća *statement*.

Metode *conn_fetch*, *conn_fetchall*, *conn_rowCount*, *conn_insertedID* su generičke metode koje primaju parametre *$query* i *$data*, te i šalju u metodu *conn_exec.* Iz povratnog *statement*-a dalje vade podatke dobivene iz upita.

7.5. Korištenje DAO arhitekture

Svaka DAO klasa implementira sučelje *DAOinterface* *Kôd 4.5 primjer sučelja (sučelje DAOinterface) u PHP-u*.

```
class ChatMsgDAO implements DAOinterface{

 private $c = null;
 private $table;

 Public function __CONSTRUCT() {
  $this->c = new DaoUser();
  $this->table = "chatmsg";
 }
 public function insertQ($values){
  $table = $this->table;
  $data = array();
  $set = null;
  $inserts = null;
  foreach($values as $k => $v){
   $set .= " " . $k . ",";
   $inserts .= " :" . $k . ",";
   $data[":".$k] = $v;
  }
  if($set != null) { $set = substr($set, 0, -1); }
  if($inserts != null) { $inserts = substr($inserts, 0, -1); }
  $query = "INSERT INTO $table ( $set ) VALUES ( $inserts )";
  return $this->c->conn_insertedID($query, $data);
 }
 public function update_id($values, $id){
  $table = $this->table;
  $set = null;
  foreach($values as $k => $v){
   $set .= " " . $k . "=:" . $k . ",";
   $data[":".$k] = $v;
  }
  if($set != null) { $set = substr($set, 0, -1); }
  $query = "UPDATE $table SET $set WHERE ChatID=:ChatID";
  $data[":ChatID"] = $id;
  return $this->c->conn_rowCount($query, $data);
 }
 public function delete_id($id) {
  $table = $this->table;
  $query = "DELETE FROM $table WHERE ChatID=:ChatID";
  $data = array(":ChatID" => $id);
  return $this->c->conn_rowCount($query, $data);
 }
 public function select_id($id) {
  $table = $this->table;
  $query = "SELECT * FROM $table WHERE ChatID=:ChatID";
  $data = array(":ChatID" => $id);
  return $this->c->conn_fetch($query, $data);
 }
 public function select_all() {
  $table = $this->table;
  $query = "SELECT * FROM $table";
  $data = array();
  return $this->c->conn_fetchall($query, $data);
 }
 public function select_chatMsg($id, $offset = 0, $newMsg = 0) {
  $chatMsgCount = $_SESSION["chatMsgCount"] + 1;
  $table = $this->table;
```

```
  $query = "SELECT *
   FROM $table WHERE FriendTypeID=:id AND ChatID>:ChatID
   ORDER BY InsertDate DESC LIMIT $chatMsgCount OFFSET $offset";
  $data = array(":id" => $id, ":ChatID" => $newMsg);
  return $this->c->conn_fetchall($query, $data);
 }
 public function select_messageMaxNum($friendTypeID) {
  $table = $this->table;
  $query = "SELECT MAX(MsgNumb)
   FROM $table WHERE FriendTypeID=:FriendTypeID";
  $data = array(":FriendTypeID" => $friendTypeID);
  return $this->c->conn_fetch($query, $data);
 }
}
```

Kôd 7.2 Primjer DAO klase (klasa ChatMsgDAO)

Svaka DAO klasa sadrži privatnu instancu klase *DaoUser* preko koje vrši konekciju na bazu podataka i ima CRUD funkcionalnosti.

Klasa *ChatMsgDAO* sa sastoji od sezam metoda i varijabli *$table* i *$c* *Kôd 7.1 Primjer DAO klase (klasa ChatMsgDAO)*. U varijablu *$this->c* se instancira klasa DaoUser, dok se u varijablu *$this->table* zapisuje naziv tablice koju klasa reprezentira.

Metoda *insertQ* prima *array* kao argument (mora sadržavati ključeve isto naziva kao i iz tablice i odgovarajuće vrijednosti). Iz *array*-a se ključevi i vrijednosti pretvore u *string* koji se dalje ubacuje u upit.

Metoda *update_id* prima parametre *$values* (koji se ponaša isto kao i parametar *$values* iz metode *insertQ*) i *$id* po kojem će se ažurirati red u tablici.

Metoda *delete_id* prima parametar *$id* po kojem će se obrisati red u tablici, dok metoda *select_id* prima parametar *$id* po kojem će se izvući red iz tablici. Metoda *select_all* će izvući svaki red iz tablice.

Metode *select_chatMsg* i *select_messageMaxNum* su unikatne metode za DAO klasu *ChatMsgDAO*. Metoda *select_chatMsg*. *Metoda select_chatMsg* izvršava upit kojim određeni broj poruka između dvije određene osobe. Metoda *select_messageMaxNum* vraća broj poruka između dvije određene osobe.

7.6. Podjela odgovornosti po mapama/klasama za *Chat*

U ovom poglavlju će se opisati kôd praktičnog dijela aplikacije (chat). Mape/klase će se objasnit po redoslijedu u kojem se javljaju u upravljaču. Upravljač će biti zadnji objašnjen jer spaja sve funkcionalnosti aplikacije.

Korišten znak „./ “ označava da se nalazi unutar mape *chat*

7.6.1. Mapa *Imports*

Datoteka *./Imports/ChatControllerImports.php* unosi sve datoteke navedene u kôdu *Kôd 7.3 Import-i datoteka (klasa) za upravljač* i izvozi ih u upravljač. Svaka Datoteka sadrži jednu klasa po kojoj je imenovana.

```
require_once('../../Utils/FriendStatus.php');
require_once('../Utils/InputFormating.php');
require_once('../Utils/DateFormating.php');
require_once('../Utils/RandomGenerator.php');
require_once('../Encryption/SimpleChatEncryption.php');
require_once('../Error/ErrorFormat/ErrorFormat.php');
require_once('../Error/ChatError.php');
require_once('../Validators/ChatInputValidator.php');
require_once('../Validators/ChatHTTPValidator.php');
require_once('../Session/ChatSession.php');
require_once('../../DB/DbRoot.php');
require_once('../../DB/DAOinterface.php');
require_once('../../DB/DAO/UsersDAO.php');
require_once('../../DB/DAO/FriendsDAO.php');
require_once('../../DB/DAO/NotificationsDAO.php');
require_once('../../DB/DAO/ChatMsgDAO.php');
require_once('../Services/UsersService.php');
require_once('../Services/FriendsService.php');
require_once('../Services/NotificationsService.php');
require_once('../Services/ChatMsgService.php');
require_once('../Models/ChatModel.php');
require_once('../Views/ChatView.php');
```

Kôd 7.3 Import-i datoteka (klasa) za upravljač

7.6.2. Mapa *Utils*

Mapa *./Utils* se sastoji od tri datoteke koje u sebi sadrže samo jednu klasu istog imena kao i datoteka.

Klasa *DateFormating* je zadužena za baratanjem datumom. Metoda *dateFormat* prima parametar $date koji je *DATETIME (string)* tipa iz baze, te ga formatira u datum zadanog tipa (parametar *$type*). Metoda createDate će jednostavno stvoriti novu instancu datuma formata danog parametra *$type*.

```
class DateFormating {

 Public function __CONSTRUCT() {}
 public function dateFormat($date, $type = "jS \of F Y") { // string
  $dateCreated = date_create(date('F d, Y', strtotime($date)));
  return date_format($dateCreated, $type);
 }
 public function createDate($type = "Y-m-d H:i:s") { // string
  return date($type);
 }
}
```

Kôd 7.4 Klasa DateFormating

Klasa *InputFormating* je zadužena za formatiranje tipova podataka prije unosa u bazu. Metoda *formatParagraphString* prima parametar tipa string kojeg formatira (*escape*-a ga i miče višak *white space*-ova) i time ga pripremi za zapis u bazu.

```
class InputFormating {

 Public function __CONSTRUCT() {}
 public function formatParagraphString($data) { // string
  return htmlspecialchars(stripslashes(trim($data)));
 }
}
```

Kôd 7.5 Klasa InputFormating

Klasa *RandomGenerator* služi za generiranje nasumičnih brojeva i znakova. Metoda *generateRandomString* nasumično generira *string* od već unaprijed određenih znakova. Prima parametar *$lenght* koji definira koliko će znakova biti generirano. Metoda *randomOpenssl* nasumično generira heksadekadski broj koji se koristi kao ključ i sprema se u polje *MsgEncKey* u tablici *chatmsg*.

```
class RandomGenerator {
 Public function __CONSTRUCT() { }
 public function generateRandomString($length = 0) { // string
 $keyspace =
 '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ.!?';
  if ($length < 1) { $length = 64; }
  if ($length > 256) { $length = 256; }
  $pieces = [];
  $max = mb_strlen($keyspace, '8bit') - 1;
  for ($i = 0; $i < $length; ++$i) {
   $pieces[] = $keyspace[rand(0, $max)];
```

```
  }
  return implode('', $pieces);
 }
 public function randomOpenssl($length = 16, $secure = false) {
 // string
  return bin2hex(openssl_random_pseudo_bytes($length, $secure));
 }
}
```

Kôd 7.6 Klasa RandomGenerator

Za *chat* se još koristi datoteka *KorijenskaMapa/Utils/FriendStatus.php*. Ona služi za dobivanje statusa između dva korisnika (npr. dali su prijatelji, poslan zahtjev, blokani,…).

Sve ove klase prate S,O i/ili D dijelove SOLID principa. Temelje se na apstrakciji što znači da ih se može koristiti u svim slučajeva. Svaka klasa ima metode koje ostvaruju isti cilj i metode nije potrebno mijenjat.

7.6.3. Mapa *Encryption*

Mapa *./Encryption* se sastoji od samo jedne datoteke *SimpleChatEncryption.php*. Ta datoteka sadrži samo jednu isto imensku klasu.

Klasa *SimpleChatEncryption* barata šifriranjem i dešifriranjem poruke korisnika. Sadrži dvije varijable *$cipher* i *$iv*. Varijabla *$cipher* definira koji će tip šifriranja biti korišten, dok varijabla *$iv* je zajednički ključ (formata heksadekadski) korišten pri svakom šifriranju i dešifriranju. Ova klasa se sastoji od dvije metode.

Metoda *simpleEncrypt* kao parametre prima *$value* koji je željeni tekst za šifrirat i *$key* koji je nasumično generirani ključ (Klasa *RandomGenerator* metoda randomOpenssl). Ova metoda poziva funkciju *openssl_encrypt* koja će šifrirati tekst i vratiti ga.

Metoda *simpleDecrypt* kao parametre prima *$encryptedValue* koji je već neki šifrirani tekst i *$key* koji je ključ korišten za šifriranje teksta. Unutar te metode se poziva funkcija *openssl_decrypt* koja dešifrira tekst.

```
class SimpleChatEncryption {
 private $cipher;
 private $iv;
 public function __CONSTRUCT() {
  $this->cipher = "AES-256-CBC";
  $this->iv = hex2bin('Ključ'); // formata heksadekadski
 }
 public function simpleEncrypt($value, $key) { // string
```

```
  return openssl_encrypt($value, $this->cipher, $key, 0, $this->iv);
 }
 public function simpleDecrypt($encryptedValue, $key) { // string
  return openssl_decrypt(
   $encryptedValue, $this->cipher, $key, 0, $this->iv
  );
 }
}
```

Kôd 7.7 Klasa SimpleChatEncryption

7.6.4. Mapa *Error*

Unutar mape *./Error* se nalazi datoteka *ErrorHandler.php* i mapa *ErrorFormat* koja u sebi sadrži datoteku *ErrorFormat.php.*

Klasa *ErrorFormat* sadrži dvije varijable sa definiranim vrijednostima i jednu metodu. Metoda *getActionError* na poziv vrača tip *array* koji je predefinirani standard za prikaz grešaka korisniku unutar aplikacije.

```
class ErrorFormat {
 protected $msg;
 protected $title;
 Public function __CONSTRUCT() {
  $this->msg = "There was an unknown error";
  $this->title = "ERROR!";

 }
 public function getActionError() { // array
  $output["action"]["openpopup"]["type"] = "";
  $output["action"]["openpopup"]["data"] = $this->msg;
  $output["action"]["openpopup"]["title"] = $this->title;
  return $output;
 }
}
```

Kôd 7.8 Klasa ErrorHandler

Klasa *ChatError* nasljeđuje klasu *ErrorFormat* i sadrži samo jednu metodu. Metoda *getBasicError* prima opcionalni parametar *$msg* i ako je validan će ga postaviti unutar varijable *$this->msg* i pozvati metodu *getActionError* iz klase *ErrorFormat.*

```
class ChatError extends ErrorHandler {
 Public function __CONSTRUCT() {
```

```
  ErrorHandler::__CONSTRUCT();
 }
 public function getBasicError($msg = null) { // array
   if( $msg != null ) { $this->msg = $msg; }
   return $this->getActionError();
 }
}
```

Kôd 7.9 Klasa ChatError

7.6.5. Mapa *Validators*

Mapa *./Validators* sadrži dvije datoteke *ChatHTTPValidator.php* i *ChatInputValidator.php*.

Klasa *ChatHTTPValidator* služi za provjeru HTTP zahtjevai određuje koja će se funkcija zvati u upravljaču. Sadrži tri metode od kojih dvije provjeravaju tip zvanja o kojem ovisi koja funkcija će se pozvat u upravljaču.

Metoda *initPost* pretvara poslani *string* formata JSON u *array.*

```
class ChatHTTPValidator {
 Public function __CONSTRUCT() {}
 public function validateInitPost() { // boolean
  if ( (! array_key_exists("ajaxType", $_POST)) && (! array_key_exist
s("posted", $_POST)) ) { return false; }
  if ( $_POST["ajaxType"] != "ajax" ) { return false; }
  return true;

 }
 public function initPost($check) { // array
  if ( ! $check ) { return $_POST; }
  $_POST = json_decode($_POST["posted"], true);
  return $_POST[0];
 }
 public function getType() { // string
  if ( array_key_exists("ajaxType", $_POST) ) { return $_POST["ajaxTy
pe"]; }
  if ( array_key_exists("type", $_POST) ) { return $_POST["type"]; }
  return "";
 }
}
```

Kôd 7.10 Klasa ChatPostValidator

Klasa *./ChatInputValidator* zadužena je za provjeru točnosti poslanih vrijednosti u HTTP zahtjevu. Provjerava dali su poslane potrebne vrijednosti, dali su dobro tipa i dali su unutar zadanih

granica (npr. broj manji od minimalnog dopuštenog). Ova klasa se sastoji od šest metoda, i sve funkcioniraju slično jedna drugoj.

Metoda *validateType_loadChat* provjerava dali zahtjev (*$_POST*) sadržava ključ *UserID* tipa *intiger* većeg ili jednakog nuli. Metoda *validateType_filterUsers* jednostavno provjerava dali zahtjev sadrži ključ *f_Username.*

```
class ChatInputValidator {
 Public function __CONSTRUCT() {}
 public function validateType_ajax($inputs = array()) { // boolean
   // Izvršni kod
 }
 public function validateType_filterUsers($inputs = array()) { // boo
lean
  if ( ! array_key_exists("f_Username", $inputs) ) { return false; }
  return true;
 }
 public function validateType_loadChat($inputs = array()) { // boolea
n
  if ( ! array_key_exists("UserID", $inputs) ) { return false; }
  if ( ! strval($inputs["UserID"]) === strval(intval($inputs["UserID"
])) ) { return false; }
  if ( $inputs["UserID"] < 0 ) { return false; }
  return true;
 }
 public function validateType_sendMsg($inputs = array()) { // boolean
  // Izvršni kod
 }

public function validateType_timerMsg($inputs = array()) { // boolean
  // Izvršni kod
 }
 public function validateType_chatLoadNext($inputs = array()) { // bo
olean
  // Izvršni kod
 }
}
```

Kôd 7.11 Klasa ChatInputValidator

7.6.6. Mapa *Session*

Mapa *./Session* sadrži samo jednu datoteku *ChatSession.php*. Klasa *ChatSession* zadužena je za postavljanje podataka u *$_SESSION* potrebne za *chat* i sadrži samo jednu metodu.

Metoda *setChatSession* prima dva parametra *$friend* koji je *ID* korisnika za provjeru dali je prijatelj ili ne i *$data* koji je izvađen red iz tablice *friends* gdje jedno od polja *UserID (_a* ili *_b*) sadrži *ID* prijavljenog korisnika a drugo polje sadrži vrijednost istu kao i parametar *$friend.* U

slučaju da su dani dobri parametri, poziva se metoda *$this->friendStatus->getFriendStatusNoQuery($friend,$data)* koja će dobiti status između dva korisnika i zapisati u *$_SESSION["chat"]["FriendType"]*. Korisnici koji između sebe imaju status *friends* mogu komunicirati u *chat*-u.

```
class ChatSession {
 private $friendStatus;
 protected $id;
 Public function __CONSTRUCT() {
  $this->friendStatus = new FriendStatus();
  $this->id = $_SESSION["UserID"];
 }
 public function setChatSession($friend = 0, $data = null) { // void
  if ( $friend == 0 || $friend == $this->id || $data == null) {
  $_SESSION["chat"]["FriendID"] = $this->id;
  $_SESSION["chat"]["FriendTypeID"] = 0;
  $_SESSION["chat"]["FriendType"] = "owner";
  return;
 }
  $_SESSION["chat"]["FriendID"] = $friend;
  $_SESSION["chat"]["FriendTypeID"] = $data["FriendID"];
  $_SESSION["chat"]["FriendType"] = $this->friendStatus-
>getFriendStatusNoQuery(
   $friend,
   $data
  );
 }
}
```

Kôd 7.12 Klasa ChatSession

7.6.7. Mapa *Services*

Mapa *./Services* sadrži više datoteka i svaka je zadužena za baratanje logikom jedne od tablica u bazi. Klasa *ChatMsgService* je zadužena za baratanje logikom *ChatMsg* tablice. Sadrži pet varijabla u koje će se instancirati isto imenske klase i sastoji se od tri metode.

Metoda *db_getMsg* barata svom logikom za prije i poslije zvanja metode *select_chatMsg* iz klase *ChatMsgDAO*. Prvo postavi vrijednost varijable *$offset* koja se onda šalje kao parametar u metodu *select_chatMsg.* Nakon što se redovi izvade iz tablice u petlji *foreach* se prolazi kroz svaki red da bi se dešifrirala poruka koristeći metodu *simpleDecrypt* iz klase *SimpleChatEncryption.*

Metoda *db_insertMsg* poziva metodu *randomOpenssl* iz klase *RandomGenerator* kako bi se dobio ključ za šifriranje poruke. Poruka se šifrira koristeći metodu *simpleEncrypt* iz klase *SimpleChatEncryption,* No prije šifriranja poruka se formatira koristeći metodu

formatParagraphString iz klase *InputFormating*. Na kraju se svi podaci kao parametar šalju metodi *insertQ* iz klase *ChatMsgDAO* koja ih zapiše u bazu.

Metoda *db_getMessageMaxNumNext* jednostavno zove metodu *select_messageMaxNum* iz klase *ChatMsgDAO* i u slučaju da je vraćena vrijednost nevaljana postavi je u 0. Na kraju ta vrijednost se poveća za 1.

```
class ChatMsgService {
 private $cChat;
 private $dateFormating;
 private $inputFormating;
 private $randomGenerator;
 private $simpleChatEncryption;
 Public function __CONSTRUCT() {
  $this->cChat = new ChatMsgDAO();
  $this->dateFormating = new DateFormating();
  $this->inputFormating = new InputFormating();
  $this->randomGenerator = new RandomGenerator();
  $this->simpleChatEncryption = new SimpleChatEncryption();
 }
 public function db_getMsg($friendTypeID, $offset = 0, $newMsg = 0 )
{ // array (DB)
  $offset = $offset * $_SESSION["chatMsgCount"];
  $data = $this->cChat-
>select_chatMsg($friendTypeID, $offset, $newMsg);
  if ( count($data) < 1 ) { return null; }
  foreach( $data as $k => $v ) {
                      $data[$k]["InsertDate"] = $this->dateFormating-
>dateFormat($v["InsertDate"]);
                      $data[$k]["Msg"] = $this->simpleChatEncryption-
>simpleDecrypt($v["Msg"], $v["MsgEncKey"]);
  }
  return $data;
 }
 public function db_insertMsg($obj) { // array (DB)
  $key = $this->randomGenerator->randomOpenssl();
  $obj["MsgEncKey"] = $key;
  $obj["Msg"] = $this->simpleChatEncryption->simpleEncrypt(
  $this->inputFormating->formatParagraphString($obj["Msg"]), $key);
  $data = $this->cChat->insertQ($obj);
  return $data;
 }
 public function db_getMessageMaxNumNext($friendTypeID) { // int
  $data = $this->cChat->select_messageMaxNum($friendTypeID);
  if ( $data == null ) { $data["MAX(MsgNumb)"] = 0; }
  $data = $data["MAX(MsgNumb)"] + 1;
  return $data;
 }
}
```

Kôd 7.13 Klasa ChatMsgService

7.6.8. Mapa *Models*

Mapa ./*Models* spada u MVC arhitekturu i sadrži samo jednu datoteku *ChatModel.php*.

Klasa *ChatModel* je zadužena za instaciranje svih *service* klasa koje će koristiti upravljač i sastoji se od sezam metoda. Svaka metoda zove jednu metodu od nekog *service*-a.

```
class ChatModel {
 private $usersService;
 private $friendsService;
 private $notificationsService;
 private $chatMsgService;
 Public function __CONSTRUCT() {
  $this->usersService = new UsersService();
  $this->friendsService = new FriendsService();
  $this->notificationsService = new NotificationsService();
  $this->chatMsgService = new ChatMsgService();
 }
 public function getFriendStats($friendID) { // array (DB)
  return $this->friendsService->db_FriendStatus($friendID);
 }
 public function getFriendsList($obj = array()) { // array (DB)
  return $this->usersService->db_friendsList($obj);
 }
 public function getChatMsg($friendTypeID, $offset = 0, $newMsg = 0)
{ // array
  return $this->chatMsgService-
>db_getMsg($friendTypeID, $offset, $newMsg);
 }
 public function getUserSimpleProfile($userID) { // array (DB)
  return $this->usersService->db_userSimpleProfile($userID);
 }
 public function updateNotificationRead($obj, $where) { // int
  return $this->notificationsService-
>updateNotifieForChat($obj, $where);
 }
 public function insertMessage($obj) { // int
  return $this->chatMsgService->db_insertMsg($obj);
 }
 public function getMessageMaxNumNext($friendTypeID) { // int
  return $this->chatMsgService-
>db_getMessageMaxNumNext($friendTypeID);
 }
}
```

Kôd 7.14 Klasa ChatModel

7.6.9. Mapa *Views*

Mapa ./*Views* spada u MVC arhitekturu i sadrži samo jednu datoteku *ChatView.php*.

Klasa *ChatView* zadužena je za HTML prikaz i sastoji se od pet metoda. U svakoj metodi se poziva funkcija *ob_start* nakon čega dolazi HTML kôd. Koristeći funkciju *ob_get_clean* se taj HTML kôd vadi kao *string* i zapisuje u varijablu.

```
class ChatView {
 public function loadPage($friends = array(), $chat = array(), $frien
dProfile = array()) { // string
  $list = $this->buildFriendsList($friends);
  $chat = $this->buildChatList($chat, $friendProfile);
  ob_start();
  ?>
   <!-- blok HTML-a -->
  <?php
  $content = ob_get_clean();
  ob_flush();
  return $content;
 }
 public function buildFriendsList($friends = array()) { // string
  // Izvršni kod
 }

 public function buildChatList($chat = array(), $friendProfile = arra
y()) { // string
 // Izvršni kod
 }
 public function ChatMsgDesign($chat = array(), $offset = 0, $load =
true) { // string
  // Izvršni kod
 }
 public function getLoadMoreForm($offset) {
  ob_start();
  ?>
   <!-- blok HTML-a -->
  <?php
  $btn = ob_get_clean();
  ob_flush();
  return $btn;
 }
}
```

Kôd 7.15 Klasa ChatView

7.6.10. Mapa *Controllers*

Unutar mape *./Controllers* (spada u MVC arhitekturu) se nalazi samo jedna datoteka *ChatController.php.* Pozivom (AJAX poziv) na ovu datoteku prvo se provjerava dali je *session* aktivan i ako nije onda se pokreće. Nakon toga dolaze svi importi potrebni da upravljač funkcionira i sama klasa *ChatController*. Nakon toga se provjerava dali je korisnik prijavljen i ako je prijavljen onda se instancira klasa *ChatController* i poziva metoda iz nje *index*. Odgovor od te metode se na kraju ispiše.

Klasa *ChatController* se sastoji od jedanaest varijabli i osam metodi. Sve privatne varijable se koriste za instanciranje klasa, dok sve privatne varijable se koriste unutar metoda. Pozvana *index* metoda jednostavno formatira u JSON oblik odgovor od pozvane metode *build* koja se nalazi u istoj klasi. Metoda *build* provjerava koja je vrijednost varijable *$this->type* (vrijednost se definira u instanciranju klase koristeći metodu *getType* iz klase *ChatHTTPValidator*) i ovisno o vrijednosti poziva odgovarajuću metodu.

```
if ( session_status() === PHP_SESSION_NONE ) session_start();
require_once('../Imports/ChatControllerImports.php');
class ChatController {
 private $dateFormating;
 private $chatError;
 private $hTTPValidator;
 private $inputValidator;
 private $chatSession;
 private $model;
 private $view;
 protected $post;
 protected $type;
 protected $id;
 protected $obj;
 Public function __CONSTRUCT() {
  $this->dateFormating = new DateFormating();
  $this->chatError = new ChatError();
  $this->chatSession = new ChatSession();
  $this->hTTPValidator = new ChatHTTPValidator();
  $this->inputValidator = new ChatInputValidator();
  $this->model = new ChatModel();
  $this->view = new ChatView();
  $this->type = $this->hTTPValidator->getType();
  $this->post = $this->hTTPValidator->initPost(
   $this->hTTPValidator->validateInitPost()
  );
  $this->id = $_SESSION["UserID"];
  $this->obj = array();
 }
 protected function loadPage() { // array
  // izvršni kod
 }
 protected function filterUsers() { // array
  // izvršni kod
 }
 protected function loadChat() { // array
  // izvršni kod
 }
 protected function sendMsg() { // array
  // izvršni kod
 }
 protected function timerMsg() { // array
  // izvršni kod
 }
 protected function chatLoadNext() { // array
```

```
  // izvršni kod
 }
 protected function build() { // array
  if ( $this->type === "ajax" ) { return $this->loadPage(); }
  if ( $this->type === "filterUsers" ) { return $this-
>filterUsers(); }
  if ( $this->type === "loadChat" ) { return $this->loadChat(); }
  if ( $this->type === "sendMsg" ) { return $this->sendMsg(); }
  if ( $this->type === "timerMsg" ) { return $this->timerMsg(); }
  if ( $this->type === "chatLoadNext" ) { return $this-
>chatLoadNext(); }
  return $this->chatError->getActionError();
 }
 public function index() { // string (JSON)
  return json_encode($this->build());
 }
}
if ( array_key_exists("Username", $_SESSION) ) {
 $class = new ChatController();
 $content = $class->index();
 echo $content;
}
```

Kôd 7.16 Klasa ChatController

Metode *loadPage, filterUsers, loadChat, sendMsg, timerMsg* i *chatLoadNext* su vrlo slične u smislu da jednostavno upravljaju i pozivaju metode drugih klasa, te rade neke provjere koristeći *if* izraz. Iz tog razloga, a i zbog same količine koda unutar svake metode ću samo objasniti metodu *loadChat* detaljnije a ostale ću samo ukratko opisati.

Metoda *loadChat* prvobitno inicijalizira varijable *$chatMSG* i *$friendProfile* kao *array* zatim pozivom na metodu *validateType_loadChat* iz klase *ChatInputValidator* i ovisno o odgovoru se metoda nastavlja ili prekida. U nastavku se poziva metoda *setChatSession* iz klase *ChatSession* kako bi se definiralo s kojim drugim korisnikom (i ako je prijatelj) trenutni korisnik komunicira. Dalje se provjerava dali je taj korisnik prijatelj i ako nije onda se jednostavno pozove metoda *buildChatList* iz klase *ChatView* i vrati se prazni izgled. U slučaju da je taj korisnik prijatelj, onda će se pozvati metode *getChatMsg* i *getUserSimpleProfile* iz klase *ChatModel* koje će dohvatiti podatke iz baze. Ti podaci će se dalje proslijediti u metodu *buildChatList* kako bi se izgradio prikaz tih poruka. Priprema se objekt koji se dalje prosljeđuje u metodu *updateNotificationRead* iz klase *ChatModel* kako bi ažurirao red u tablici *notifications.*

```
protected function loadChat() { // array
        $chatMSG = array();
        $friendProfile = array();
 if ( ! $this->inputValidator->validateType_loadChat($this-
>post) ) { return $this->chatError->getBasicError(); }
```

```
 $this->chatSession->setChatSession(
  $this->post["UserID"],
  $this->model->getFriendStats($this->post["UserID"])
 );
 $output["action"]["insert"][0]["id"] = "ChatScreen";
 if ( $_SESSION["chat"]["FriendType"] == "friends" && $_SESSION["chat
"]["FriendID"] != $this->id ) {
  $chatMSG = $this->model-
>getChatMsg($_SESSION["chat"]["FriendTypeID"]);
  $friendProfile = $this->model-
>getUserSimpleProfile($_SESSION["chat"]["FriendID"]);
  $this->obj = array();
  $this->obj["ChangeDate"] = $this->dateFormating->createDate();
  $this->obj["NotifieAction"] = 0;
  $this->obj["DoesCount"] = 0;
  $this->obj["NotifieTypeID"] = $_SESSION["chat"]["FriendTypeID"];
  $this->obj["NotifieTypeTable"] = "friends/chat";
  $where["UserID_toNotifie"] = $this->id;
  $where["NotifieTypeID"] = $_SESSION["chat"]["FriendTypeID"];
  $this->model->updateNotificationRead($this->obj, $where);
 }
  if ( $chatMSG == null ) { $chatMSG = array(); }
  if ( $friendProfile == null ) { $friendProfile = array(); }
  $output["action"]["insert"][0]["value"] = $this->view-
>buildChatList($chatMSG, $friendProfile);
  $output["callback_after"] = true;
  return $output;
}
```

Kôd 7.17 Metoda loadChat iz klase ChatController

Metoda *loadPage* se poziva kako bi inicijalizirala stranicu *Chat.*

Metoda *filterUsers* se poziva kako bi se filtrirala lista prijatelja od korisnika s kojima može razgovarati.

Metoda *sendMsg* se poziva kada korisnik želi poslati poruku, dok metoda *timerMsg* se poziva svakih tri sekunde kako bi se provjerilo dali je korisnik ili prijatelj poslao poruku i u slučaju da je poruka poslana prikazati je.

Metoda *chatLoadNext* se poziva kada korisnik želi učitati starije poruke.

7.7. AJAX pozivi

Za poziv upravljača se koristi AJAX poziv. AJAX koristi *XMLHttpRequest* kako bi poslao zahtjev na server i odgovor od servera se koristi unutar *success* objekta. U slučaju da dođe do nekakve pogreške onda se koristi *error* objekt. Objekt *type* definira kojeg tipa je AJAX zahtjev, objekt *url* definira link na koji se zahtjev šalje, a objekt *data* sadrži vrijednosti koje se šalju na zahtjev.

```
$.ajax({
 type: "post",
 url: php_file,
 data: theData,
 processData: false,
 contentType: false,
 traditional: true,
 success: function(data) {
  // izvršni kod
 },
 error: function(jqXHR, textStatus, errorThrown) {
  console.log(textStatus, errorThrown);
 }
});
```

Kôd 7.18 AJAX poziv sa jQuery-em

8. Zaključak

Ovaj završni rad nastao je kako bi se objasnili osnovni jezici web-a, objektno orijentirani pristup programiranju u PHP-u, korištenje MVC arhitektura, te kombinacije MVC arhitekture sa drugim arhitekturama i principima kako bi se postila lakša čitljivost samog kôda i smanjenja kompleksnost istog.

Ja sam u praktičnom dijelu ovog rada koristio OOP u PHP-u. Kao primarnu arhitekturu sam koristio MVC, dok sam ostale principe i arhitekture koristio kako bi si olakšao daljnji rad i smanjio kompleksnost kôda. Arhitekturu DAO sam koristio kako bi smanjio količinu upita kroz program, Service-e sam koristio kako bi za svaki upit na jednom mjestu imao logiku koja ide uz njega.

U praktičnom dijelu su bili korišteni principi SOLID-a po potrebi kako bi kôd bio čitki i kako bi se ga lakše moglo ponovno upotrijebiti (najčešće korišteni principi su bili S i D).

U Varaždinu, _____ _____

Potpis studenta

Sveučilište Sjever

SVEUČILIŠTE SJEVER

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, DARKO PETROVIĆ *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom IZRADA MREŽNOG SERVISA ZA ČAVRLJANJE PRIMJENOM MVC-A I PHP-A *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)
Darko Petrović
(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, DARKO PETROVIĆ *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom IZRADA MREŽNOG SERVISA ZA ČAVRLJANJE PRIMJENOM MVC-A I PHP-A *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)
Darko Petrović
(vlastoručni potpis)

9. Literatura

[1] Nikola Brežnjak: Upotreba MVC frameworka u izradi PHP aplikacija, Seminar, FER, Zagreb, 2009. https://nikola-breznjak.com/portfolio/CodeIgniter.pdf

[2] https://www.webdesignmuseum.org/web-design-history dostupno 11.6.2020.

[3] https://www.w3.org/Style/CSS20/history.html dostupno 14.6.2020.

[4] Denis Stančer: Osnove JavaScripta (C501), Zagreb 2015

[5] https://codeburst.io/javascript-what-the-heck-is-a-callback-aba4da2deced dostupno 14.6.2020.

[6] https://www.vvg.hr/ict/programiranje/jquery-sto-zasto-kako dostupno 17.6.2020.

[7] http://www.netakademija.hr/sto-je-php/ dostupno 18.06.2020.

[8] https://www.php.net/manual/en/history.php.php dostupno 17.06.2020. dostupno 18.06.2020.

[9] S. Flisar, Osnove objektno orijentiranog programiranja u C++, završni rad, Sveučilište Josipa Jurja Strossmayera u Osijeku, Osijek, 2016.

[10] N. Plazibat, Objektno orijentirano programiranje, ECSAT d.o.o., Split, listopad 2002

[11] https://www.php.net/manual/en/language.oop5.basic.php dostupno 18.6.2020.

[12] https://almirvuk.blogspot.com/2016/04/oop-nasljeivanje-objasnjeno.html dostupno 24.6.2020.

[13] https://www.fer.unizg.hr/_download/repository/4_Nasljedjivanje_Polimorfizam.pdf dostupno 26.6.2020.

[14] Tonči Carić i Mario Buntić, Uvod u relacijske baze podataka, Zagreb, 2015.

[15] http://php.com.hr/66 dostupno 3.7.2020

[16] http://docbook.rasip.fer.hr/ddb/res/46/Ch3.3.2.html dostupno 6.7.2020.

[17] https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces dostupno 10.7.2020.

[18] A. Smijulj, A. Meštrović: Izgradnja MVC modularnog radnog okvira Zbornik Veleučilišta u Rijeci, Vol. 2 (2014), No. 1, pp. 215-232

[19] https://www.c-sharpcorner.com/UploadFile/damubetha/solid-principles-in-C-Sharp/ dostupno 20.7.2020.

[20] https://symfony.com/six-good-reasons dostupno 6.8.2020.

[21] http://www.zemris.fer.hr/~ssegvic/pubs/ooup3MorePatterns.pdf dostupno 24.6.2020.

[22] A. Smijulj, A. Meštrović: Izgradnja MVC modularnog radnog okvira, Zbornik Veleučilišta u Rijeci, Vol. 2 (2014), No. 1, pp. 215-232

Popis kôda

Popis slika

Prilozi

Aplikaciji se može pristupiti preko poveznice: http://arwen.unin.hr/~dapetrovic/MVC/