Objektno orijentirano programiranje u PHP-u

Gal, Petra

Sveučilište
Sjever

Multimedija, oblikovanje i primjena

Završni rad br. 643/MM/2019

Objektno orijentirano programiranje u PHP-u

Student

Petra Gal, matični broj: 3685/601

Mentor

Vladimir Stanisavljević, mr.sc.

Varaždin, rujan 2019. godine

Sveučilište
Sjever

Multimedija, oblikovanje i primjena

Završni rad br. 643/MM/2019

Objektno orijentirano programiranje u PHP-u

Student

Petra Gal, matični broj: 3685/601

Mentor

Vladimir Stanisavljević, mr.sc.

Varaždin, rujan 2019. 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 Petra Gal | MATIČNI BROJ 3685/601

DATUM 16.09.2019. | KOLEGIJ Programski alati 3

NASLOV RADA Objektno orijentirano programiranje u PHP-u

NASLOV RADA NA ENGL. JEZIKU Object oriented programming in PHP

MENTOR mr.sc. Vladimir Stanisavljević | ZVANJE Viši predavač

ČLANOVI POVJERENSTVA

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

Zadatak završnog rada

BROJ 643/MM/2019

OPIS

Objektno orijentirano programiranje (OOP) je značajna paradigma u programskom inženjerstvu. Premda spada u proceduralne jezike programski jezik PHP u novijim inačicama podržava većinu uobičajenih OOP konstrukcija. U ovom radu potrebno je:
* opisati osnove objektno orijentiranog programiranja
* napraviti pregled 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
* napraviti osnovna mjerenja perfomansi učitavanja i pohranjivanja podataka
* detaljno opisati programski kod i pozive koji su korišteni u radu

ZADATAK URUČEN 20.09.2019. | POTPIS MENTORA

REPUBLIKA HRVATSKA
SVEUČILIŠTE SJEVER
KOPRIVNICA
SVEUČILIŠTE SJEVER

Sažetak

PHP skriptni jezik ima podršku za objektno orijentirano programiranje i za rad s različitim bazama podataka. To su dvije mogućnosti ovog jezika koje su bitne za programere radi konkurentnosti na tržištu rada. OOP pomaže programerima u pisanju čišćeg koda kojeg je lakše mijenjati i ponovno koristiti. OOP počiva na izradi klasa i primjeraka (instanci) tih klasa koje se zovu objekti. Objekt koji ima sposobnost da nadživi proces koji ga je kreirao naziva se trajni (perzistentni) objekt (eng. *persistent object*). Jedna od najpopularnijih tehnika za ostvarivanje perzistencije koja se koristi i u praktičnom djelu rada je pohranjivanje podataka u relacijske baze podataka.

Za praktični dio završnog rada izrađena je PHP aplikacija za praćenje dolazaka studenata na vježbe ili predavanja koja upotrebljava principe i tehnologije obrađene u teoretskom dijelu na njihovoj osnovnoj razini.

Ključne riječi: **PHP, objektno orijentirano programiranje, baze podataka, CRUD funkcionalnosti**

Abstract

PHP is a scripting language with support for object oriented programming and handling multiple databases. These two PHP features are important because they increase programmer's competitivness on the job market. Using OOP helps in writing cleaner code that is easier to maintain and reuse. OOP fundamentals lie in creating classes and their instances called objects. Persistent objects are objects with a lifespan longer than the duration of a process that created them. One of the most frequently used ways of achieving object persistence, also implemented in the practical part of this paper, is storing data in relational databases.

Practical part of the paper describes a PHP application for taking student attendance made by using principles and technologies described in the theoretical part at their basic level.

Keywords: **PHP, object oriented programming, databases, CRUD functionalities**

Popis korištenih kratica

ENG	Engleski
OOP	Objektno orijentirano programiranje (eng. Object Oriented Programming)
PHP	Personal Home Page, PHP: Hypertext Preprocessor
PDO	PHP objekti podataka (eng. PHP Data Objects)
SQL	Strukturirani upitni jezik (eng. Structured Query Langauge)
DSN	Ime izvora baze podataka (eng. Database Source Name)
CRUD	Funkcionalnosti za stvaranje, iščitavanje, ažuriranje i brisanje podataka (eng. create, read, update, delete)
HTML	Standardizirani jezik za izradu web stranica (eng. HyperText Markup Language)
URL	Jedinstveni lokator resursa (eng. Uniform Resource Locator)

Sadržaj

1. Uvod

PHP (skraćenica za PHP: Hypertext Preprocessor) je popularan skriptni jezik otvorenog koda (eng. *open source*) namijenjen za mrežni razvoj (eng. *web development*). Razlikuje se od klijentskih skriptnih jezika poput JavaScripta jer se izvršava na poslužitelju.

Jedna od njegovih najznačajnijih mogućnosti je njegova podrška za različite baze podataka, a od verzije 5.0 PHP programski jezik ima i jaku podršku za objektno orijentirano programiranje. [1]

U vrijeme kada je programiranje sve traženije zanimanje, PHP će zadovoljiti i početnike zbog svoje jednostavnosti, ali i iskusne programere zbog raznih naprednih mogućnosti. Za programere je radi konkurentnosti na tržištu rada uvijek bitno pratiti najnovije mogućnosti i paradigme programskih jezika, stoga se ovaj rad bavi najtraženijim PHP mogućnostima - objektno orijentiranim programiranjem i radom s bazama podataka.

U prvom dijelu rada teoretski se obrađuju bitni konceptu u objektno orijentiranom programiranju i radu s MySQL bazama podataka u PHP jeziku.

U drugom dijelu rada se opisuje izrada aplikacije za praćenje dolazaka studenata na predavanja i/ili vježbe korištenjem svih navedenih tehnologija u teoretskom dijelu rada. Aplikacija bi trebala biti napisana sljedeći osnovne objektno orijentiranih principe i izvršavati osnovne CRUD funkcionalnosti (eng. *create, read, update, delete*) pomoću PHP-ove PDO ekstenzije.

2. Prednosti objektno orijentiranog programiranja

OOP ne rješava sve probleme pisanja lošeg kôda već samo usmjerava programere na određene standardne načine pisanja koda. Iako je OOP programiranje sporije od proceduralnog što se performansi tiče, programeri koristeći objektno orijentirane principe pišu „čišći" kod kojeg je lakše mijenjati i ponovno koristiti (eng. *reusable code*). Upravo je potonje vrlo važno u modernom razvoju programske potpore jer se unutar jednog tima ili firme određene programske komponente mogu ponovno koristiti čime se smanjuje ukupno vrijeme razvoja programa i povećava njegova vrijednost. Također, što više ljudi koristi neku programsku cjelinu, to je veća šansa da će se pronaći i riješiti greške (tzv. bugove). S vremenom će ta komponenta sve brže i brže postajati bliža savršenoj.

Ovaj način razvoja pogodan je za razvoj složenih programskih sustava. Sam razvoj je brži jer se omogućuje jednostavno korištenje postojećeg programskog koda. [2]

3. Osnove o objektima i klasama

Za razliku od proceduralnog programiranja, osnovni principi OOP-a su globalni. [3]

Podatci (varijable) i operacije (funkcije) koje manipuliraju podacima objedinjuju se u jednu cjelinu koja se naziva objekt. Program se sastoji od skupa objekata koji međusobnom komunikacijom rješavaju problem.

3.1. Što su to klase i objekti

Ako želimo pričati o objektima moramo prvo objasniti klasu bez koje objekti ne mogu postojati. Klasa ili razred je nacrt za jedan ili proizvoljni broj objekata koji imaju slične karakteristike definirane unutar te klase. Ako za primjer uzmemo objekt iz stvarnog života - kuću, klasa je nacrt za tu kuću.

Klasa ima vlastite funkcije (metode) i članske varijable (atribute) koje su zajedničke jednom ili skupini objekata, ovisno koliko instanci te klase imamo. Varijable određuju stanje, a metode ponašanje objekta.

Dakle, može se reći da je objekt jedan primjerak (instanca) klase, a tih instanci može biti proizvoljno mnogo, kao što uostalom i iz jednog nacrta za kuću možemo izgraditi nebrojeno mnogo kuća.

Za naš primjer klase, uzet ćemo klasu *Clan* čija svrha može biti praćenje ili izmjena stanja prijave korisnika na nekom web forumu. Definiciju svake klase započinje ključna riječ *class* nakon koje slijedi ime klase i vitičaste zagrade unutar kojih definiramo metode i atribute koji joj pripadaju. Naziv klase može biti koji, osim rezerviranih PHP riječi. Te riječi su predefinirani identifikatori koje programer ne može koristiti u svojim skriptama. Ispravan naziv klase počinje sa slovom ili donjom crtom, nakon čega mogu ići slova, brojevi i donje crte u količini koju programer želi. [4]

Možemo uočiti da klasa *Clan* već ima deklarirana dva atributa i tri metode.

```
class Clan
{
    $korisnickoIme = "";
    private $prijavljenost = false;

    function prijava()
    {
       $this->prijavljenost = true;
    }

    function odjava()
    {
       $this->prijavljenost = false;
    }

    function stanjePrijave()
    {
       return $this->prijavljenost;
    }
}
```

Kod 3.1: Klasa Clan

Vrijednost prve varijable, *$korisnickoIme* inicijalno je postavljena na prazan niz znakova. Varijabla *$prijavljenost* sprema stanje prijave korisnika i početna joj je vrijednost logički izraz *false*. Pristup varijabli definiran je kao *private*.

Prva metoda, *prijava()* postavlja stanje prijave na *true*. Dok sljedeća metoda, *odjava()* postavlja stanje prijave opet na *false*. Zadnja metoda, *stanjePrijave()* provjerava status prijave korisnika, tj. trenutnu vrijednost varijable *$prijavljenost*.

Objašnjenju uloge varijable *$this* i ključne riječi *private* posvećena su kasnija poglavlja. Da bismo uopće mogli koristiti atribute i metode koje sadrži ta klasa prvo moramo stvoriti objekt.

3.2. Instanciranje

Kreiranje objekata dane klase zovemo instanciranjem objekata, a same objekte zovemo instancama dane klase. Kako bi se kreirala instanca klase, koristi se ključna riječ *new* uz naziv klase unutar koje se objekt želi kreirati. Pogledajmo to na primjeru klase *Clan*:

```
$Ivan = new Clan;
```

Kod 3.2: Objekt Ivan klase Clan

Stvorili smo instancu klase *Clan* koju smo pohranili u varijablu *$Ivan*. Sada možemo varijablama klase *Clan* dodijeliti određene vrijednosti. To radimo tako što napišemo ime varijable u koju smo pohranili objekt, zatim koristimo operator -> i ime varijable čiju vrijednost želimo izmijeniti - važno je napomenuti da uz varijablu nakon strelice ne koristimo simbol *$*. [5]

```
$Ivan = new Clan;
$Ivan->korisnickoIme = 'carrot123';
```

Kod 3.3: Postavljanje atributa korisnickoIme klase Clan

Korisničko ime korisnika $Ivan postavili smo na vrijednost "*carrot123"*. Napravit ćemo novog korisnika $Jana čiji će *$korisnickoIme* biti "*mrkvica45"* i postaviti vrijednost varijable *$prijavljenost* za korisnika $Jana *true* (Kod 3.4).

```
$Ivan = new Clan;
$Ivan->korisnickoIme = 'carrot123';

$Jana = new Clan;
$Jana->korisnickoIme = 'mrkvica45';
$Jana->prijava();
```

Kod 3.4: Objekt Jana klase Clan

Važno je zapamtiti da je klasa pripadajućih objekata samo nacrt. Stvarni podaci sadržani su u pojedinom objektu, a ne u njoj. Stoga, svaki objekt ima svoje podatke, a pojedinih objekata može biti proizvoljno mnogo.

Za poziv metode vrijedi sličan princip kao i kod pristupa varijabli objekta, jedina razlika su obavezne zagrade nakon imena metode, kao što smo mogli vidjeti u primjeru *$Jana->prijava()*. *$Jana->prijava()* poziva funkciju *prijava()* koja varijablu *$prijavljenost* postavlja na *true*, tako da je trenutno *$prijavljenost* status korisnika $Jana *true*, a *$prijavljenost* status korisnika $Ivan *false*. Značenje varijable *$this* objasnit ćemo u sljedećem poglavlju.

3.3. $this varijabla

Varijabla *$this* koristi se samo unutar klase i to za ukazivanje na objekt koji poziva funkciju koja pripada toj klasi. Opet ćemo se poslužiti primjerom klase *Clan* prikazanom u kodu 3.1.

Metoda *prijava()* je definirana unutar klase *Clan* i, kao što je objašnjeno u prijašnjem poglavlju, svrha joj je postavljanje varijable *$prijavljenost* na *true*.

U trenutku kada smo instancirali objekt $Jana i pozvali metodu *prijava()*, varijabla *$this* počinje predstavljati objekt $Jana (Kod 3.4). Stoga, objektu $Jana metoda *prijava()* zapravo izgleda ovako: [3]

```
class Clan
{
    $korisnickoIme = "";
    private $prijavljenost = false;

    function prijava()
    {
       $Jana->prijavljenost = true;
    }
}
```

Kod 3.5: Objašnjenje korištenja varijable $this

3.4. Enkapsulacija i kontrola pristupa

Učahurivanje (ili enkapsulacija) bitan je princip objektno-orijentiranog programiranja. Programer bi kod kreiranja klase trebao osigurati da osjetljivi podaci budu sakriveni ili zaštićeni kako ne bi došlo do slučajnih ili namjernih promjena. Zbog toga se podaci enkapsuliraju. Enkapsulacijom klasa skriva neke atribute i metode od ostalih klasa.

Kontrola pristupa vrši se kroz upotrebu tri ključne riječi:

• *public* (javna) – podrazumijeva se da je varijabla ili metoda javna ako se ne definira drukčije. To znači da joj može pristupiti bilo tko i iz bilo kojeg dijela koda.

• *private* (privatna)– ako je varijabla ili metoda definirana kao privatna pristup je moguć samo unutar klase u kojoj se ona nalazi.

• *protected* (zaštićena) – ako je varijabla ili metoda zaštićena pristup je moguć unutar klase u kojoj se ona nalazi, ali i unutar njezinih naslijeđenih klasa.

Od članova klase Clan definiranu kontrolu pristupa ima samo varijabla *$prijavljenost*, čiji je pristup postavljen na *private*. Ostali članovi trenutno nemaju definiranu ni jednu ključnu riječ pa se podrazumijeva da je pristup njima javan. Nakon što i njima dodamo ključne riječi klasa će izgledati ovako:

```php
class Clan
{
    public $korisnickoIme = "";
    private $prijavljenost = false;

    public function prijava()
    {
       $this->prijavljenost = true;
    }

    public function odjava()
    {
       $this->prijavljenost = false;
    }

    public function stanjePrijave()
    {
       return $this->prijavljenost;
    }
}
```

Kod 3.6: Definiranje kontrole pristupa u klasi Clan

S obzirom da je pristup varijabli *$prijavljenost* postavljen na *private*, objekti koje smo stvorili u prijašnjim poglavljima - *$Jana* i *$Ivan*, nisu joj mogli izravno pristupiti.

Kako bi se moglo pristupiti zaštićenim i privatnim varijablama i izvan klase u kojima su definirane obično se koriste tzv. *geter(eng. getter)* i *seter (eng. setter)* metode. *Getter* metoda koristi se za čitanje podataka iz polja kojima nije moguć pristup. *Setter* metoda koristi se za kreiranje podataka unutar nedostupnih polja. [6]

Tu funkciju unutar klase *Clan* vrše javne metode *prijava(), odjava()* i *stanjePrijave()* koje omogućuju izmjenu i provjeru vrijednosti privatne varijable *$prijavljenost* i izvan klase *Clan*.

Kao što je već spomenuto, ključna riječ *protected* omogućuje klasama koje nasljeđuju tu klasu pristup njezinim varijablama ili metodama.

3.5.Nasljeđivanje

Još jedan bitan koncept u objektno orijentiranom programiranju je nasljeđivanje koje predstavlja stvaranje novih klasa i objekata koristeći se postojećim klasama. Ono omogućava iskorištavanje postojećeg koda i proširivanje njegove funkcionalnosti.

Ukratko, to je proces smanjivanja i optimiziranja programskog koda povezivanjem logičkih cjelina/klasa u hijerarhijsku vezu, u kojoj kreiramo nove objekte na osnovu klasa koje dijele iste ili slične atribute. [7]

Roditeljska ili bazna klasa je klasa koja se koristi kao baza za nasljeđivanje, a od nje klasa dijete, ili podređena klasa nasljeđuje svojstva i ponašanja. Potklase nasljeđuju sve atribute i metode svoje roditeljske klase, osim onih koji su deklarirani kao *private* i posebnu vrstu metoda - *konstruktore*. U većini slučajeva klasa dijete i nadopunjuje naslijeđeno svojim vlastitim svojstvima ili ponašanjima, a može i zamijeniti ili izmijeniti naslijeđeno ponašanje. Klase djeca mogu dalje imati svoju 'djecu' i tako dalje dokle god je potrebno za funkcioniranje aplikacije. No, klasa dijete može imati samo jednog direktnog roditelja.

U PHP-u se može prilikom stvaranja nove klase deklarirati da je ta klasa potklasa postojeće klase. Za to se koristi ključna riječ *extends*.

Za potrebe fiktivnog web foruma, dodat ćemo novu klasu *Administrator* koja će proširivati klasu *Clan*. Klasa *Administrator* imat će svoju vlastitu metodu s ulaznim argumentom - *zabraniPristupClanu()*.

```
class Administrator extends Clan
{
   public function zabraniPristupClanu( $clan)
   {
      echo "$this->korisnickoIme zabranio/la je pristup: $user-
>korisnickoIme";
   }
}
```

Kod 3.7: Klasa Administrator nasljeđuje klasu Clan

Klasa *Administrator* nasljeđuje sve javne atribute i metode klase *Clan*, također nadodaje svoju metodu. *$this->korisnickoIme* unutar metode *zabraniPristupClanu()* se odnosi na objekt koji je član klase *Administrator*, dok se *$user->korisnickoIme* odnosi na objekt koji je član klase *Clan*. Podatke o tom članu dobivamo preko argumenta *$clan* funkcije *zabraniPristupClanu()*.

U sljedećim primjerima instanciramo obje klase i koristimo dostupne metode i atribute:

```
//novi clan
$clan = new Clan;
$clan->korisnickoIme = "Jan";
$clan->prijava();

//novi administrator
$admin = new Administrator;
$admin->korisnickoIme = "Jana";
$admin->prijava();
```

Kod 3.8: Instanciranje klasa Clan i Administrator

Kao i u prijašnjim primjerima, instancirali smo novi objekt klase *Clan*. Korisničko ime smo postavili na vrijednost *Jan*, a status prijave, odnosno varijablu *$prijavljenost* na *true*. S obzirom na to da je klasa *Administrator* naslijedila i varijablu *$korisnickoIme* i metodu *prijava()*, isto možemo napraviti i za objekt klase *Administrator*. Stvorili smo novog administratora čije je korisničko ime *Jana*, a status prijave *true*. Nakon poziva metode *zabraniPristupClanu()* klase *Administrator* dobiti ćemo tekst - *Jana je zabranio/la pristup: Jan.*

```
//prikazuje tekst - Jana je zabranio/la pristup: Jan
$admin->zabraniPristupClanu( $clan);
```

Kod 3.9: Nasljeđivanje na primjeru metode zabraniPristupClanu

Jana je trenutno *$korisnickoIme* klase *Administrator*, dok je *$korisnickoIme* klase *Clan* - Jan. Prisjetimo se kako izgleda metoda *zabraniPristupClanu()* unutar klase Administrator:

```
public function zabraniPristupClanu( $clan)
 {
    echo "$this->korisnickoIme zabranio/la je pristup: $user-
>korisnickoIme";
 }
```

Kod 3.10: Metoda zabraniPristupClanu

Objekt klase *Clan* s podacima o korisničkom imenu prenosimo funkciji *zabraniPristupClanu()* preko argumenta.

U tekstu na početku poglavlja spomenuto je da klasa dijete može izmijeniti ili zamijeniti ponašanja roditeljske klase. Metode je moguće nadjačati (eng. *overriding*) što nas dovodi do sljedećeg poglavlja.

3.6. Polimorfizam i nadjačavanje

Izraz polimorfizam dolazi od dviju grčkih riječi koje u doslovnom prijevodu znače "mnogo oblika". U objektno orijentiranom programiranju to se odnosi na sposobnost varijable, funkcije ili objekta da preuzme više oblika. To znači da možemo stvoriti više metoda s istim imenom koje će se ponašati različito i u skladu s tipom objekata s kojima se koriste.

U OOP, polimorfizam, nasljeđivanje i nadjačavanje (eng. *override*) međusobno su povezani.

Metoda u podređenoj klasi treba imati isto ime, argumente i isto ili veće pravo pristupa kako bi nadjačala metodu iz roditeljske klase. Pisanjem metode s drugačijim parametrima od onog

u nadređenoj klasi ne nadjačava metodu iz roditeljske klase, već samo stvara novu metodu, jedinstvenu za potklasu [8]

Nadjačavanje omogućuje klasi dijete drugačiju implementaciju iste metode iz roditeljske klase. Kada nadjačamo metodu iz roditeljske klase, roditeljska metoda gubi svoju originalnu funkcionalnost u klasi dijete. No, u nekim slučajevima potrebno je zadržati tu funkcionalnost. To se čini pozivanjem nadjačane metode u metodi klase dijete. Ako želimo pristupiti metodama ili varijablama roditeljske klase u klase dijete, koristi se ključna riječ *parent* nakon koje slijede operator rezolucije opsega (eng. *scope resolution operator*) '::' i ime metode ili varijable.

```
class Administrator extends Clan
{
   public function prijava()
   {
      parent::login();
      echo "$this->username se prijavio/la";
   }
   public function zabraniPristupClanu( $clan)
   {
      echo "$this->korisnickoIme zabranio/la je pristup:
$user->korisnickoIme";
   }
}
```

Kod 3.11; Nadjačavanje

U primjeru iznad, funkcija *prijava()* klase *Administrator* nadjačava metodu *prijava()* roditeljske klase *Clan*, ali zadržava njezinu funkcionalnost pozivajući roditeljsku funkciju koristeći *parent::prijava()*. Također, nakon poziva roditeljske metode, dodaje i vlastitu funkcionalnost.

Dakle, funkcija *prijava()* koja pripada klasi *Clan* postavlja atribut $prijavljenost na *true* i ispisuje da se korisnik određenog korisničkog imena prijavio.

4. Magične metode

Magične metode (eng. *magic methods*) su ugrađene funkcije koje su jedinstvene po tome što počinju sa znakom '__' koji se kreira koristeći dvije podvlake. Pozivaju se automatski kada se ispune određeni uvjeti. Vidljivost magičnih metoda mora uvijek biti postavljena kao javna.

PHP omogućava veliki broj magičnih metoda poput *__construct(), __destruct() __get(), __set()* i *__call()*.

4.1. __construct

Konstruktor je magična metoda koja se automatski poziva kod instanciranja klase, odnosno kod stvaranja novog objekta klase. Može biti smješten bilo gdje u klasi, ali se obično piše pri samom vrhu, odmah nakon deklaracije klase. Kao što se može vidjeti iz prijašnjih primjera, konstruktori nisu obavezni već se koriste kada želimo prenijeti neke parametre odmah kod stvaranja objekta. S obzirom na to da je to metoda koja se koristi samo kod stvaranja novog objekta, konstruktor ne vraća povratnu vrijednost. Neke od mogućih upotreba su mu postavljanje inicijalnih vrijednosti atributima definiranima unutar klase ili učitavanje datoteka.

```
class Clan
{
    private $korisnickoIme = "";
    private $lokacija = "";
    private $prijavljenost = false;

    __construct( $korisnickoIme, $prezime)
    {
       $this->korisnickoIme = $korisnickoIme;
       $this->lokacija = $lokacija;
    }

    public function prikaziProfil()
    {
       echo "Korisničko ime: $this->korisnickoIme<br>";
       echo "Lokacija: $this->lokacija";
    }

    public function prijava()
    {
       $this->prijavljenost = true;
    }
```

```
    public function odjava()
    {
       $this->prijavljenost = false;
    }

    public function stanjePrijave()
    {
       return $this->prijavljenost;
    }
}
```

Kod 4.1: Konstruktor i nova metoda klase Clan

U primjeru iznad, klasi *Clan* dodali smo konstruktor koji kod inicijalizacije klase postavlja varijablu *$korisnickoIme* i novu varijablu *$lokacija* na vrijednosti koje mu se prenose kroz argumente. Klasa također sadrži i novu metodu – *prikaziProfil*, koja ispisuje vrijednosti atributa *$korisnickoIme* i *$lokacija*.

```
$clan = new Clan( "Jesus", "Los Angeles");
$clan->prikaziProfil();
```

Kod 4.2: Instanciranje klase Clan

Nakon toga, instancirali smo novi objekt klase *Clan*. Kod instanciranja objekta klase koja sadrži konstruktor, obavezno se koriste oble zagrade nakon imena klase, bez obzira da li se konstruktoru prenose vrijednosti kroz argumente ili ne. U ovom slučaju, konstruktor prima vrijednosti 'Jesus' i 'Los Angeles' koje zatim sprema u varijable *$korisnickoIme* i *$lokacija*. Dakle, korisničko ime korisnika/objekta spremljenog u varijablu *$clan* je 'Jesus', a njegova lokacija je *'Los Angeles'*. To se može provjeriti pozivom funkcije *prikaziProfil()*;

Ako klasa dijete nema definiran svoj konstruktor, vrijedi pravilo nasljeđivanja od roditeljske klase. Ako smo nadjačali konstruktor roditeljske metode u klasi dijete, ali svejedno želimo koristiti njegovu funkcionalnost, pozivamo ga unutar konstruktora koji ga je nadjačao kao bilo koju drugu nadjačanu metodu – koristeći *parent::ime_metode*, u ovom slučaju *parent::__construct()*.

4.2. __destruct()

Uloga destruktora je suprotna ulozi konstruktora. Destruktor je magična metoda koja se poziva automatski prije uništenja objekta. Može se koristiti za brisanje podataka i oslobođenje memorije ili spremanje nekih korisničkih podataka nakon završetka procesa. Kao i konstruktor,

ni destruktor ne vraća povratne vrijednosti. No, za razliku od konstruktora, destruktor ne prima argumente. [9]

4.3. __get() i __set()

Magične metode __get i __set automatski se pokreću kod pokušaja pristupa nedostupnim atributima. Ovdje se pod nedostupnim smatraju privatni atributi i atributi koji nisu nigdje deklarirani u klasi, tj. nepostojeći atributi.

__get	pokreće se kada pokušavamo pristupiti nedostupnom atributu	prima jedan argument – ime atributa koji je nedostupan
__set	pokreće se kada nedostupnom atributu pokušamo pripisati neku vrijednost	prima dva argumenta – ime nedostupnog atributa i vrijednost na koju ga pokušavamo postaviti

Tablica 4.1: Magične metode __get i __set

PHP omogućava stvaranje i pristup nepostojećim atributima upravo pomoću *__set* i *__get* metoda. Jedan od razloga zašto bi koristili *__set* i *__get* u tu svrhu je ako vrijednosti želimo pohraniti u jedno polje, umjesto u zasebne atribute. To ćemo u primjeru ispod napraviti s atributom *lokacija*.

```
class Clan
{
    private $korisnickoIme = "";
    private $podaci = array();

    public function __get( $atribut)
    {
       if( $atribut == "korisnickoIme")
       {
          return $korisnickoIme;
       }
       else
       {
          if( array_key_exists( $atribut, $this->podaci))
          {
             return $this->podaci[$atribut];
          }
          else
```

```
            {
                return null;
            }
        }
    }
    public function __set( $atribut, $vrijednost)
    {
        if( $atribut == "korisnickoIme")
        {
            $this->korisnickoIme = $vrijednost;
        }
        else
        {
            $this->podaci[$atribut] = $vrijednost;
        }
    }
}
```

Kod 4.3: Primjer magičnih metoda __get i __set

```
$Janko = new Clan;
$Janko->korisnickoIme = "Jan";
$Janko->lokacija = "Gorski kotar";
```

Kod 4.4: Primjer instanciranja klase s nepostojećim atributom

Klasu *Clan* smo očistili od suvišnoga koda, te se ona sada sastoji od privatne varijable *$korisnickoIme*, privatnog polja *$podaci* i *_get* i *_set* funkcija. U polje *$podaci* spremamo sve nedefinirane atribute.

Metoda *__get* kao ulazni argument prima ime nedostupnog atributa kojeg pokušavamo dohvatiti, zatim provjerava da li je ime atributa kojeg pokušavamo dohvatiti *korisnickoIme*. U slučaju da jest, vraća trenutnu vrijednost privatnog atributa *$korisnickoIme*.

U slučaju da ime atributa nije *korisnickoIme*, provjerava da li postoji ključ pod imenom atributa kojeg pokušavamo dohvatiti u polju $data. Ako postoji, vraća vrijednost atributa spremljenog u polje. Ako se ni jedan slučaj ne ispuni, vraća vrijednost *null*.

Metoda *__set* uvijek uzima dva argumenta, od kojih je prvi ime atributa kojeg pokušavamo dohvatiti, a drugi vrijednost na koju ga pokušavamo postaviti. U primjeru iznad, metoda *__set* provjerava da li je ime atributa čiju vrijednost želimo promijeniti - *korisnickoIme* i ako je tvrdnja istinita, postavlja atribut *$korisnickoIme* na željenu vrijednost.

Ako tvrdnja nije istinita, sprema novi atribut i njegovu vrijednost u polje $data, s imenom atributa kao ključem.

Ispod deklaracije klase, stvorili smo novog korisnika, odnosno novi objekt, pod imenom $Janko. Pomoću metode __set postavili smo vrijednost privatnog atributa *$korisnickoIme* na *Jan* i spremili nepostojeći atribut *lokacija* i njegovu pripadajuću vrijednost u polje *$data*. Uz pomoć metode *__get* možemo u bilo kojem trenutku dohvatiti te vrijednosti.

4.4. __call()

Prijašnje poglavlje bavilo se magičnim metodama koje se koriste za nedostupne atribute. Postoji i rješenje kod pokušaja pristupa nedostupnim metodama, a to je magična metoda *__call()*.

__call()	pokreće se kada pokušamo pristupiti nedostupnim metodama	Prima dva argumenata - prvi je ime metode kojoj pokušavamo pristupiti, a drugi je polje koje sadrži argumenti te metode

Tablica 4.2: Magična metoda __call

Metoda *__call* može biti korisna u slučaju kada želimo proslijediti metodu iz jedne klase u kojoj je ona nedostupna, drugoj, nevezanoj klasi koja ima tu metodu.

```
class Clan
{
    private $korisnickoIme = "";

    __construct( $korisnickoIme)
    {
       $this->korisnickoIme = $korisnickoIme;
    }

    public function dohvatiKorisnickoIme()
    {
       return $this->korisnickoIme;
    }
}

class nevezanaKlasa
{
    private $clan;

    public function __construct( $clan)
    {
```

```
        $this->clan = $clan;
    }

    public function __call( $metoda, $argumenti)
    {
        $this->clan->$metoda( $argumenti);
    }
}

$clan = new Clan( 'Janko');
$nevezanaKlasa = new nevezanaKlasa( $clan);
echo $nevezanaKlasa->dohvatiKorisnickoIme(); //ispisuje "Janko"
```

Kod 4.5: Primjer korištenja magične metode __call

U primjeru iznad, prvo smo stvorili instancu klase *Clan* kojoj prenosimo vrijednost "Janko" koja se sprema u privatnu varijablu *$korisnickoIme*. Nakon toga, stvorili smo instancu klase *nevezanaKlasa* kojoj prenosimo novostvoreni objekt klase *Clan - $clan*.

Kod pokušaja pozivanja nedefinirane metode *dohvatiKorisnickoIme* u klasi *nevezanaKlasa*, aktivira se *__call* metoda unutar koje se poziva metoda *dohvatiKorisnickoIme()* klase *Clan* i ispisuje se vrijednost korisničkog imena za objekt *$clan*.

5. Apstraktne klase i metode

Apstraktne (eng. *abstract*) klase služe samo kao nacrt za klase koje ih nasljeđuju i ne mogu biti instancirane, odnosno nije moguće stvoriti objekt apstraktne klase. Korisne su u slučajevima kada želimo izbjeći ponavljanje istih atributa i metoda u više klasa, ili kada želimo osigurati da klase implementiraju određenu metodu. U tu svrhu, apstraktne klase mogu sadržavati 'obične' i apstraktne metode.

5.1. Ključna riječ *abstract*

Deklaracija apstraktne klase započinje s ključnom riječi *abstract* nakon koje slijede ključna riječ *class* i ime klase.

```
abstract class Osoba
{

}
```

Kod 5.1: Deklaracija apstraktne klase Osoba

Klasa Osoba u primjeru iznad označena je kao apstraktna klasa i svaki pokušaj instanciranja javljao bi grešku. Deklaracija apstraktne metode je slična. Prije ključne riječi za kontrolu pristupa i riječi *function* dolazi ključna riječ *abstract*.

5.2.Apstraktne metode

Postoje dvije bitne razlike između apstraktnih i običnih, ne apstraktnih metoda:

- apstraktne metode mogu biti deklarirane samo unutar apstraktnih klasa
- apstraktne metode nemaju implementaciju, implementirati ih moraju klase djeca

Apstraktne metode su korisne u slučajevima kada želimo detalje implementacije metoda ostaviti svakoj pojedinoj klasi koja nasljeđuje apstraktnu klasu. Naslijeđene apstraktne metode moraju imati jednaku ili veću vidljivost u klasi dijete. Na primjer, ako je apstraktna metoda u apstraktnoj klasi označena s pravom pristupa *protected*, njezina implementacija u klasi dijete mora imati pravo pristupa *protected* ili *public*. Također, implementacija apstraktne klase u klasi dijete mora imati jednak broj argumenata kao i apstraktna metoda u apstraktnoj klasi. [10]

Primjera radi, stranici imaginarnog foruma iz ranijih poglavlja dodat ćemo i opciju stranice za kupovinu, tako da ćemo imati dvije različite vrste korisnika: članove foruma i kupce.

Apstraktnu klasu *Osoba* popunit ćemo zajedničkim karakteristikama i obaveznim implementacijama za klase *Clan* i *Kupac*.

```
abstract class Osoba
{
    $korisnickoIme = "";
    private $prijavljenost = false;

    function prijava()
    {
       $this->prijavljenost = true;
    }

    function odjava()
    {
       $this->prijavljenost = false;
    }

    function stanjePrijave()
    {
       return $this->prijavljenost;
    }
}
```

Kod 5.2: Apstraktna klasa Osoba

Osoba sadrži dvije *setter* i *getter* metode za privatne varijable *$ime* i *$prezime* i apstraktnu funkciju *prikaziPorukuPozdrava()* koju će implementirati klase *Kupac* i *Clan* zasebno.

```
class Clan extends Osoba
{
    public function prikaziPorukuPozdrava()
    {
       echo $this->dohvatiImePrezime.", dobrodošli na stranicu foruma!";
    }

    public function novaTema( $naslov)
    {
       //prima $naslov i stvara novu temu
    }
}
```

Kod 5.3: Clan proširuje apstraktnu klasu Osoba

Klasa *Clan* proširuje apstraktnu klasu *Osoba*, implementira metodu *prikažiPorukuPozdrava()*, čija je implementacija obavezna, i dodaje vlastitu metodu *novaTema()*.

```
class Kupac extends Osoba
{
    public function prikaziPorukuPozdrava()
    {
       echo $this->dohvatiImePrezime.", dobrodošli na stranicu za kupnju!";
    }

    public function dodajUKosaricu( $naslov)
    {
       //dodaje $stvar u košaricu za kupnju
    }
}
```

Kod 5.4: Kupac proširuje apstraktnu klasu Osoba

Klasa *Kupac* koja isto proširuje apstraktnu klasu *Osoba*, implementira *prikaziPorukuPozdrava()*, i dodaje vlastitu metodu *dodajUKosaricu()*.

Apstraktna klasa *Osoba* omogućila nam je izbjegnemo nepotrebno ponavljanje koda u klasama *Clan* i *Kupac*.

6. Sučelja

Apstraktne klase i sučelja(eng. *interfaces)* slična su po tome što ne mogu biti instancirana i služe samo kao nacrti. I iako dijele neke sličnosti, postoji nekoliko ključnih razlika:

- apstraktna klasa se definira s ključnom riječi *abstract,* dok se sučelje definira s ključnom riječi *interface*
- apstraktna klasa se nasljeđuje ili proširuje (eng. *extends*), a sučelje se implementira (eng. *implements*)
- klasa može naslijediti samo jednu apstraktnu klasu, ali implementirati više različitih sučelja.
- sučelje može naslijediti jedno ili više sučelja
- sučelje ne podržava implementaciju vlastitih metoda. U tom smislu je svaka metoda unutar sučelja apstraktna, iako se ne koristi ključna riječ *abstract*. Apstraktna klasa može, ali i ne mora sadržavati apstraktne metode
- pravo pristupa svake metode unutar sučelja mora biti postavljeno na javno
- sučelja ne mogu sadržavati varijable, mogu sadržavati samo konstante (ključna riječ *const*)

Ukratko, sučelja definiraju zajedničku funkcionalnost za klase koje ih implementiraju.

6.1. Implementacija sučelja

Deklaracija sučelja započinje s ključnom riječi *interface* nakon koje slijedi ime sučelja.

```
interface upravljanjePodacima
{


}
```

Kod 6.1: Deklaracija sučelja upravljanjePodacima

Sučelje *upravljanjePodacima* sadržavat će deklaracije metoda za spremanje, dohvaćanje i brisanje podataka iz baze podataka.

```php
interface upravljanjePodacima
{
    public function spremi();
    public function dohvati();
    public function izbrisi();
}
```

Kod 6.2: Deklaracije metoda u sučelju

```php
class Clan extends Osoba implements upravljanjePodacima
{

    public function prikaziPorukuPozdrava()
    {
       echo $this->dohvatiImePrezime(), dobrodošli na stranicu foruma!";
    }

    public function spremi()
    {
       //sprema člana u bazu podataka
    }

    public function dohvati()
    {
       //dohvaća člana iz baze podataka
    }

    public function izbrisi()
    {
      //briše člana iz baze podataka
    }

}
```

Kod 6.3: Implementiranje sučelja upravljanjePodacima u klasi Clan

```
class Tema implements upravljanjePodacima
{
    public function prikaziNaslov()
    {
       //prikazuje naslov teme
    }
    public function spremi()
    {
       //sprema temu u bazu podataka
    }

    public function dohvati()
    {
       //dohvaća temu iz baze podataka
    }
    public function izbrisi()
    {
      //briše temu iz baze podataka
    }

}
```

Kod 6.4: Implementiranje sučelja upravljanjePodacima u klasi Tema

Klase *Clan* i *Tema* su dvije nepovezane klase koje dijele isto sučelje zbog kojeg obadvije klase moraju implementirati metode *spremi()*, *dohvati()*, *izbrisi()* (Kod 6.3 i kod 6.4). Način na koji će metode koje su definirane u sučelju biti implementirane ostavlja se svakoj pojedinoj klasi koja implementira to sučelje.

I apstraktne klase i sučelja bitne su komponente objektno orijentiranog programiranja koje poboljšavaju organizaciju koda i time omogućuju izbjegavanje različitih grešaka i konflikata. Koncept klasa koje imaju različitu funkcionalnost, ali dijele isti ‘nacrt’ (apstraktnu klasu ili sučelje) pripada jednom bitnom načelu OOP-a spomenutom u prijašnjim poglavljima, a to je polimorfizam.

7. Perzistencija objekata i baze podataka

Životni vijek objekta (eng. *object lifetime*) je vrijeme koje protekne između kreiranja objekta i njegovog uništavanja. Objekt čiji životni vijek počinje i završava se u okviru jednog procesa naziva se privremeni (tranzijentni) objekt (eng. *transient object*). Objekt koji ima sposobnost da nadživi proces koji ga je kreirao naziva se trajni (perzistentni) objekt (eng. *persistent object*).

Jedna od najpopularnijih tehnika za ostvarivanje perzistencije je pohranjivanje objekata u relacijske baze podataka. [11]

7.1. Primjena relacijskih baza podataka

Relacijske baze podataka predstavljaju skup organiziranih podataka spremljen u računalu na određen način. Sastoje se od skupa povezanih dvodimenzionalnih tablica odnosno relacija.

Podaci se spremaju u tablice koje se sastoje od redaka i stupaca, a za obradu podataka koristi se Strukturirani upitni jezik (eng. *Structered Query Language*) SQL, odnosno njegove različite inačice.

Sve relacijske baze podataka implementiraju četiri glavne funkcionalnosti, a to su stvaranje, iščitavanje, ažuriranje i brisanje podataka, odnosno CRUD (eng. *create, read, update, delete*) funkcionalnosti.

Sustavi za upravljanje relacijskim bazama podataka (eng. *relational database management systems*) i njihove popratne CRUD funkcionalnosti nisu vezani za bilo koji programski jezik ili program. To omogućuje podacima da nadžive tijek izvršavanja bilo kojeg programa, odnosno da budu perzistentni. [11]

7.2. MySQL

Jedan od najpopularnijih sustava za upravljanje relacijskim bazama podataka je MySQL. MySQL je otvorenog koda (eng. *open source*), pokreće se na poslužitelju, te podržava višekorisnički pristup bazama podataka.

Svaka MySQL baza može imati nekoliko korisnika koji joj mogu pristupiti, a svaki korisnik ima predefinirane mogućnosti za rad, odnosno različite razine ovlasti. Ovakav pristup znatno smanjuje mogućnost pojave grešaka. Još jedna od prednosti ovog sustava je to što postoje verzije za sve važnije operacijske sustave. [12]

Iako ima i podršku za veliki broj programskih jezika, osobito popularna kombinacija su MySQL sustav i programski jezik PHP. Jedna od najznačajnijih mogućnosti PHP-a je upravo podrška za različite baze podataka. Tome uvelike pridonosi PHP-ova objektno-orijentirana PDO (*PHP Data Objects*) ekstenzija.

7.3. PDO

Za stvaranje veze na odabranu bazu podataka trebamo inicijalizirati novi objekt klase PDO kojemu prenosimo argumente DSN, korisničko ime i lozinku, a možemo prenijeti i polje koje sadržava neke od brojnih dostupnih opcija.(kod 7.1)

```
$veza = new PDO( $dsn, $korisnickoIme, $lozinka);
```

Kod 7.1: Instanca klase PDO

DSN je akronim za Database Source Name, u doslovnom prijevodu Ime izvora baze podataka i sadrži niz znakova (eng. *string*) koji opisuju vezu.

```
$dsn = "mysql:host=localhost;dbname=mojabaza ";
```

Kod 7.2: Postavljanje DSN-a

U primjeru iznad, podaci koje prenosimo su MySQL domaćin (eng. *host)* i ime baze podataka – u ovom slučaju 'localhost' i 'mojabaza'. Pretpostavimo da već imamo tablicu 'student' spremljenu u bazi. Nakon uspostave veze možemo započeti s unošenjem i pretraživanjem podataka. SQL upit se direktno izvršava pomoću *query()* metode.

```
$rezultat = $veza->query( 'SELECT * FROM student');
```

Kod 7.3: Izvršavanje upita pomoću query metode

U varijablu $rezultat u primjeru iznad, spremili smo sve podatke iz tablice ‘student’. Postoji i drugi način izvršavanja upita koji se smatra sigurnijim i pouzdanijim od query() metode, a to je korištenjem prepare() i execute() metoda.

Da bismo izvršili neku SQL izjavu kod koje nam rezultat nije bitan ili samo želimo znati broj redaka pogođen izjavom, možemo koristiti *exec()* metodu. Varijabla $brojRedaka nam vraća broj redaka koje smo izbrisali. (kod 7.4)

```
$brojRedaka = $veza->exec( "DELETE FROM student WHERE ime = 'Ivan'");
```

Kod 7.4: Izvršavanje SQL izjave pomoću exec metode

I na kraju, da bi zatvorili vezu $veza koju smo stvorili na početku poglavlja, varijabli $veza samo dodijelimo vrijednost null.(kod 7.5)

```
$veza = null;
```

Kod 7.5: Zatvaranje PDO veze

8. Izrada PHP aplikacije za praćenje dolazaka studenata (praktični dio)

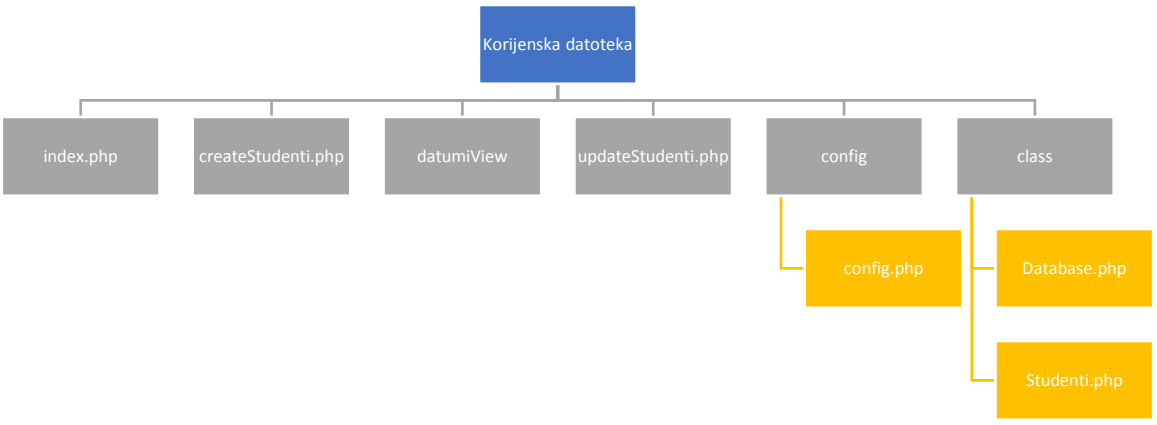
Za praktični dio završnoga rada izradila sam PHP aplikaciju za praćenje dolazaka studenata na vježbe ili predavanja poštujući objektno orijentirane principe obrađene u teoretskom dijelu rada. Također, aplikacija ostvaruje perzistenciju podataka pohranjivanjem istih u MySQL bazu podataka. Za komunikaciju s bazom podataka i izdavanje CRUD naredbi koristila sam PHP-ovu PDO ekstenziju. Koristila sam Bootstrap razvojni okvir za front-end dizajn aplikacije zbog jednostavnosti korištenja i brže izrade stranica.

Aplikacija se može pronaći na poveznici: http://evidencija-studenata.000webhostapp.com/

Razvoj aplikacije podijeljen je na pet osnovnih koraka:

- Izrada direktorija i strukture projekta
- Izrada baze podataka i pripadajućih tablica
- Spajanje na bazu podataka
- Dizajn stranica aplikacije
- CRUD funkcionalnosti

8.1. Izrada direktorija i strukture projekta



Slika 8.1: Struktura direktorija

U korijenskoj datoteci projekta nalaze se HTML stranice koje se prikazuju korisniku, te dva pod-direktorija od kojih jedan sadrži sve klase, a drugi config.php dokument. HTML (*HyperText Markup Language*) je jezik koji služi za izradu web stranica te ćemo pomoću njega i razvojnog okvira Bootstrap oblikovati sadržaj.

Dokument index.php predstavlja početnu HTML stranicu aplikacije koja će sadržavati tablicu za evidenciju dolazaka studenata za trenutni datum. U tablici jednom retku tablice će osim imena i prezimena studenta biti i dva radijska gumba (eng. *radio button*) za označavanje prisutnosti studenta, te jedan gumb za brisanje studenta iz tablice, odnosno baze.

Ispod tablice će se nalaziti gumb za spremanje označenih prisutnosti studenata, te jedan gumb koji će korisnika odvesti na stranicu createStudenti.php. Na stranici createStudenti.php će se nalaziti forma za unos novog studenta i gumb za povratak na početnu stranicu. Iznad tablice na početnoj stranici će se nalaziti gumb koji će odvesti korisnika na stranicu datumiView.php. Na stranici datumiView.php će se nalaziti tablica sa svim spremljenim datumima, odnosno evidencijama dolazaka studenata na određene datume i gumb za vraćanje na početnu stranicu. Red tablice će sadržavati spremljeni datum i kraj njega, gumb koji će odvesti korisnika na stranicu updateStudenti.php. Na stranici updateStudenti.php korisnik će moći ažurirati dolaske studenata za određeni datum. Stranica će sadržavati i gumb za spremanje ažuriranih dolazaka, te gumb za povratak na stranicu datumiView.php.

U pod-direktoriju config se nalazi dokument config.php koji će sadržavati podatke za spajanje na bazu podataka.

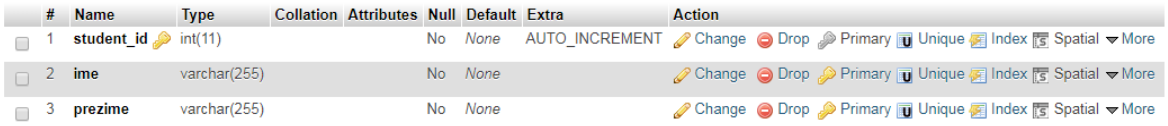
U pod-direktoriju class će se nalaziti dvije klase. U dokumentu Database.php će biti klasa *Database* koja će upravljati spajanjem na bazu podataka, a u dokumentu Studenti.php klasa Studenti koja će sadržavati CRUD funkcionalnosti, odnosno omogućavati stvaranje novih studenata, spremanje evidencije dolazaka, iščitavanje spremljenih studenata i evidencija i, te brisanje istih.

8.2. Izrada baze podataka i pripadajućih tablica

Baza podataka pod imenom *pa3* je stvorena u MySQL jeziku koristeći phpMyAdmin sučelje. phpMyAdmin je besplatno sučelje za upravljanje bazama podataka napisano u PHP-u. Omogućuje kreiranje baza podataka, izradu tablica, unos i izmijene podataka ovisno o razini

pristupa korisnika baze. Administrator baze ima sva pripadajuća prava i može mijenjati prava drugih korisnika.

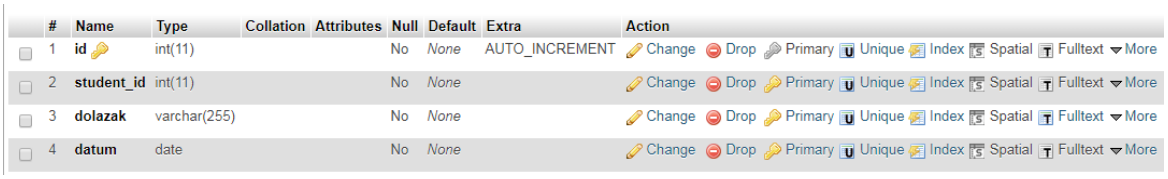
Sljedeći korak je stvaranje dvije nove tablice unutar baze pa3, *studenti* i *dolasci*, služeći se phpMyAdmin sučeljem.



#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	student_id	int(11)			No	None	AUTO_INCREMENT	Change Drop Primary Unique Index Spatial More
2	ime	varchar(255)			No	None		Change Drop Primary Unique Index Spatial More
3	prezime	varchar(255)			No	None		Change Drop Primary Unique Index Spatial More

Slika 8.2: Tablica studenti

Tablica *studenti* treba imati tri stupca – *student_id*, *ime* i *prezime*. Stupac *student_id* je brojčanog tipa INT, označen je kao primarni ključ i vrijednosti su postavljene na AUTO_INCREMENT što znači da se vrijednost tih polja dodjeljuje automatski. Ostali stupci – *ime* i *prezime* – će sadržavati ime i prezime pojedinog studenta i oni su tekstualnog tipa VARCHAR s duljinom znakova u zagradi.(slika 8.2)



#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	id	int(11)			No	None	AUTO_INCREMENT	Change Drop Primary Unique Index Spatial Fulltext More
2	student_id	int(11)			No	None		Change Drop Primary Unique Index Spatial Fulltext More
3	dolazak	varchar(255)			No	None		Change Drop Primary Unique Index Spatial Fulltext More
4	datum	date			No	None		Change Drop Primary Unique Index Spatial Fulltext More

Slika 8.3: Tablica dolasci

Tablica *dolasci* treba imati četiri stupca – *id*, *student_id*, *dolazak* i *datum*. Stupac *id* predstavlja cjelobrojan primarni ključ tablice *dolasci.* Stupac *student_id* bit će popunjen istim vrijednostima kao i istoimeni stupac u tablici studenti, stoga mora biti tipa INT. On će biti poveznica između tih tablica. Stupac dolazak je tekstualnog tipa VARCHAR i sadržavat će podatke o dolasku ili odsutnosti studenta.

Nakon stvaranja tablica *studenti* i *dolasci*, potrebno je preko opcije relacijski pregled (eng. relation view) postaviti stupac *student_id* u tablici *dolasci* kao strani ključ iz tablice *studenti* u kojoj je on primarni ključ. Tako je između dvije tablice uspostavljena relacija i one moraju sadržavati iste vrijednosti u svakom polju stupca *student_id*.

8.3. Spajanje na bazu podataka

Prije bilo kakvog rada s bazom u PHP-u, moramo se spojiti se na poslužitelj baze podataka i odabrati bazu za rad. Za početak, potrebno je definirati konfiguraciju za spajanje. Konfiguraciju definiramo unutar config.php dokumenta i mora sadržavati ime domaćina, korisničko ime, lozinku i ime baze na koju se spajamo. Te vrijednosti definiramo kao konstante pomoću PHP-ove *define()* ugrađene funkcije. Vrijednost konstanti se ne može mijenjati ili brisati tijekom izvršavanja programa.

```
define('DB_HOST', 'localhost');
define('DB_USER', 'root');
define('DB_PASS', '');
define('DB_NAME', 'pa3');
```

Kod 8.1: Konfiguracija za spajanje na bazu podataka

U priloženom kodu može se vidjeti kako izgleda gotova konfiguracija. Konstanti DB_HOST dodijeljeno je ime domaćina na kojeg se spajamo, u ovom slučaju *localhost*. Vrijednost konstante DB_USER predstavlja korisničko ime, dok vrijednost za DB_PASS, odnosno lozinke za prijavu na bazu podataka nije postavljena i ostaje prazna. Na kraju, vrijednost konstante DB_NAME je ime baze podataka na koju se spajamo.

Nakon što je konfiguracija gotova, potrebno je napisati klasu čiji će zadatak biti provjera da li postoji već ostvarena veza na bazu podataka, i ako je odgovor negativan, spajanje na *pa3* bazu. Klasa će se zvati *Database* i nalazit će se u dokumentu Database.php unutar foldera *class*.

Za spajanje na bazu i daljnje vršenje raznih CRUD akcija koristimo objektno-orijentiranu PDO ekstenziju. Kako bi se spojili na bazu instanci PDO klase ćemo prenositi konfiguraciju koju smo definirali u config.php dokumentu. Stoga je kao prvi korak potrebno uključiti dokument config.php unutar dokumenta Database.php. Klasa će se sastojati od funkcije *_connectToDB()*, konstruktora i atributa s podacima za pokretanje veze na bazu.

```
class Database {

    private $dbHost = DB_HOST;
    private $dbUser = DB_USER;
    private $dbPass = DB_PASS;
    private $dbName = DB_NAME;
    public $pdo;

    public function __construct() {
        $this->_connectToDB();
    }

    private function _connectToDB() {
        if (!isset($this->pdo)) {
            try {
                $this->pdo = new PDO("mysql:host=".$this->dbHost.
";dbname=".$this->dbName, $this->dbUser, $this->dbPass);
                $this->pdo->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
                $this->pdo->exec("SET CHARACTER SET utf8");
            } catch (PDOException $e) {
                die("Failed to connect with Database!");
            }
        }
    }
}
```

Kod 8.2: Klasa Database

Kao što se može vidjeti u kodu iznad, atributi $dbHost, $dbUser, $dbPass i $dbName spremaju podatke iz config.php dokumenta. Njihovo pravo pristupa je postavljeno na privatno jer će se koristiti samo unutar klase i to u *_connectToDB* privatnoj funkciji. Atribut $pdo služi za spremanje uspostavljene PDO veze i koristit će se i izvan klase, pa mu je zbog toga pravo pristupa javno. Funkcija *_connectToDB()* provjerava da li je atribut $pdo postavljen na neku vrijednost i u slučaju da nije stvara PDO instancu i sprema ju u taj atribut. PDO metodi *setAttribute()* prenosimo atribute ATTR_ERRMODE i ERRMODE_EXCEPTION koji omogućavaju upravljanje greškama pomoću PDO iznimki (eng. *exceptions*). Funkcija *_connectToDB()* koristi *try-catch* blok. U *try* bloku se izvršava kod za spajanje na bazu, a u slučaju greške u spajanju, PDO iznimka se hvata u *catch* bloku koji se onda izvršava. *Catch* blok ispisuje poruku o greški i prekida izvođenje programa pomoću ugrađene PHP *die()* funkcije.

PDO *exec()* metodu koristimo za postavljanje sustava kodiranja znakova na UTF-8. Funkcija *_connectToDatabase()* se izvršava u konstruktoru prilikom stvaranja svake nove instance klase *Database*.

Sljedeći korak u izradi aplikacije je dizajn stranica koje će se prikazivati korisniku.

8.4. Dizajn stranica aplikacije

Kao što je spomenuto u prošlim poglavljima, za oblikovanje sadržaja aplikacije koristimo HTML i razvojni okvir Bootstrap. Izradu stranice započinjemo pisanjem jednostavnog HTML kostura te uključivanjem Bootstrap funkcionalnosti u zaglavlje (eng. *head*) HTML dokumenta. Koristit ćemo razne Bootstrap klase za stilove kako bi uredili prikaz HTML elemenata. Za dizajn gumbi koristit ćemo besplatnu vektorsku galeriju ikona *Font Awesome*, tako da je i tu poveznicu potrebno dodati u zaglavlje. Nakon što smo dodali sve navedeno HTML struktura početne stranice index.php trebala bi izgledati ovako:

```
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
        <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.mi
n.css" integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
        <script src="https://kit.fontawesome.com/d186bf1433.js">
        </script>
    </head>
    <body>
    </body>
</html>
```

Kod 8.3: HTML struktura stranice

Zatim u HTML tijelo (eng. *body*) možemo početi dodavati elemente opisane u poglavlju 8.1. Izrada direktorija i strukture projekta.

Početna stranica će uvijek sadržavati trenutni datum i njega možemo ispisati direktno u HTML pomoću sljedećeg koda:

```
<?php
  echo date('Y-m-d');
?>
```

Kod 8.4: Ispisivanje trenutnog datuma

Date() je PHP-ova ugrađena funkcija, a unutar zagrada se nalazi format datuma kojeg želimo. U našem slučaju taj se format podudara s formatom kojeg imamo u bazi u tablici *dolasci*. Iznad datuma mora se nalaziti gumb za pregled stranice datumiView.php.

Centralna komponenta stranice je forma i tablica koja se nalazi unutar nje. Preko formine POST metode prenosit ćemo unijete dolaske nakon klika na gumb „Spremi dolaske“. U kasnijem poglavlju implementirati ćemo njihovo spremanje u bazu. Klikom na gumb „Izbriši“ će se preko GET metode prenositi identifikator retka koji mora biti izbrisan, te će biti implementirano njegovo brisanje iz baze. Također, učitavanje redaka studenata u tablicu bit će dinamično.

```
<form action="" method="Post">
                <table class="table table-bordered">
                    <thead class="text-center">
                        <th>#</th>
                        <th>IME</th>
                        <th>PREZIME</th>
                        <th>PRISUTAN/NA</th>
                        <th>IZBRIŠI</th>
                    </thead>
                    <tr>
                        <td> Petra</td>
                        <td> Gal </td>
                        <td>
                            <label class="radio-inline">
                                <input type="radio" name="dolazak"
value="tu" required>+
                            </label>
                            <label class="radio-inline">
                                <input type="radio" name="dolazak"
value="odsutan">-
                            </label>
                        </td>
                        <td>
                            <a href="index.php" class="btn btn-
danger                  btn-sm"><i class="fa fa-trash" aria-
hidden="true"></i></a>
                        </td>
                    </tr>
                </table>
                <div class="form-group">
                    <input type="submit" class="btn btn-outline-primary
btn-sm m-0 waves-effect" name="submit" value="Spremi dolaske">
                    <a class="btn btn-success  float-right"  data-
toggle="tooltip" title="Dodaj novog studenta" href="createStudenti.php">
                     <i class="fas fa-plus"></i></a>
                </div>
   </form>
```

Kod 8.5: Forma za unos dolazaka studenata u obliku tablice

Kao što se može vidjeti iz priloženog koda, trenutno su svi podaci na stranici hard kodirani, no to nam to omogućuje da vidimo kompletni izgled stranice u pregledniku.

Slika 8.4: Izgled početne stranice

Klik na zeleni gumb sa Font Awesome plus ikonom nas vodi na stranicu createStudenti.php. Stranica mora sadržavati formu za unos imena i prezimena studenta koja će se klikom na gumb „Spremi“ slati preko POST metode i spremati u bazu podataka.

```
<form style="padding-top:10px;" action="" method="Post">
                        <div class="form-group">
                            <label for="ime">Ime</label>
                            <input type="text" class="form-control"
name="ime" id="ime">
                        </div>
                        <div class="form-group">
                            <label for="prezime">Prezime
                            </label>
                            <input type="text" class="form-control"
name="prezime" id="prezime">
                        </div>
                        <div class="form-group">
                            <input type="submit" class="btn btn-outline-
primary btn-sm m-0 waves-effect" name="noviStudent" value="Spremi">
                        </div>
                    </form>
```

Kod 8.6: Forma za unos novog studenta

Zasad funkcionalnost za spremanje nije implementirana. Rezultat pokretanja stranice u pregledniku može se vidjeti na sljedećoj slici:

Slika 8.5: Izgled forme za unos novog studenta

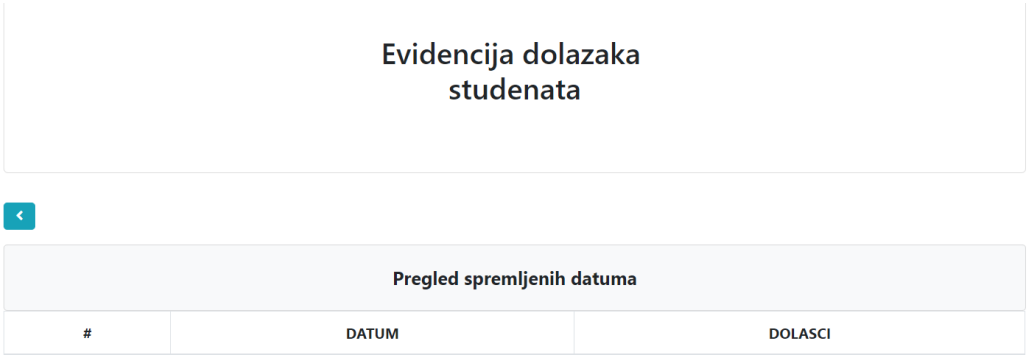
Klik na Font Awesome ikonu strelice vraća korisnika na početnu stranicu.

Stranica datumiView.php također mora sadržati tablicu. Njezina svrha bit će iščitavanje spremljenih datuma iz baze podataka. Trenutni sadržaj tablice može se vidjeti u kodu ispod.

```
<table class="table table-bordered">
    <thead class="text-center">
        <tr>
            <th>#</th>
            <th>DATUM</th>
            <th>DOLASCI</th>
        </tr>
    </thead>
    <tbody class="text-center">
    </tbody>
</table>
```

Kod 8.7: HTML struktura tablice u kojoj će se ispisivati spremljeni datumi

Sastoji se od tri stupca – redni broj, datum i dolasci. Stupac dolasci će u svakom retku kraj datuma imati gumb za izmjenu spremljenih dolazaka. Nakon klika na gumb korisnik će biti odveden na stranicu updateStudenti.php, a pomoću GET metode u URL-u će se prenositi datum na čiji se gumb kliknulo. Stranica bez podataka iščitanih iz baze izgleda ovako:



Slika 8.6: Izgled stranice sa spremljenim datumima

Stranica updateStudenti.php bit će po izgledu slična početnoj stranici. Neće biti mogućnosti dodavanja novog studenta i brisanja studenta jer su one rezervirane samo za početnu stranicu. Iščitavat će zabilježena imena i prezimena studenata te njihove dolaske iz baze i omogućavati izmjenu dolazaka koristeći forminu POST metodu.

8.5. CRUD funkcionalnosti

Iako smo mogli ostvariti vezu s bazom u prošlom poglavlju, nismo mogli manipulirati s njenim podacima preko PHP-a, te je aplikacija, iako više-manje vizualno gotova, bila statična i bez bilo kakvih CRUD funkcionalnosti.

Kao prvi korak, potrebno je stvoriti klasu Studenti unutar dokumenta Studenti.php. Klasa Studenti će u konstruktoru spremati instancu klase Database u svoj privatni atribut, pa je odmah pri vrhu dokumenta potrebno uključiti dokument Database.php. Klasa Studenti će se sastojati od javnih funkcija - *createStudent()*, *readStudents()*, *deleteStudent()*, *addAttendance()*, *updateAttendance()*, *getDBDates()*, *getAllData()* - i niza pomoćnih privatnih funkcija umetnutih radi povećanja čitljivosti koda.

```php
<?php
require_once 'Database.php';

class Studenti {

    private $db;
```

```
    public function __construct() {
        $this->db = new Database();
    }

    public function createStudent() {
    }

    public function readStudents() {
    }

    public function deleteStudent() {
    }

    public function addAttendance() {
    }

    public function updateAttendance() {
    }

    public function getDBDates() {
    }

    public function getAllData() {
    }

}
```

Kod 8.8: Struktura klase Studenti u dokumentu Studenti.php

U kodu iznad konstruktoru klase je dodana instanca klase Database spremljena u privatni atribut $db preko kojeg možemo koristiti razne PDO mogućnosti. Dodane su i zasad prazne funkcije koje ćemo koristiti za ostvarivanje CRUD funkcionalnosti.

U nadolazećim potpoglavljima popunit ćemo prazne funkcije i pretvoriti statični dizajn stranica u dinamični.

8.5.1. Stvaranje novog studenta

Baza podataka zasad je prazna i da bi mogli manipulirati podacima u njoj, prvo ih moramo unijeti. To će biti zadatak *createStudent()* funkcije koja će primati argumente $ime i $prezime prenesene preko POST metode sa stranice createStudenti.php. Funkcija će vraćati poruku u slučaju uspješno izvršene akcije. Na stranici createStudenti.php treba biti umetnut sljedeći kod:

```php
<?php
if (isset($_POST['noviStudent'])) {
    $studenti = new Studenti();
    $ime = filter_var($_POST['ime'], FILTER_SANITIZE_STRING);
$prezime = filter_var($_POST['prezime'], FILTER_SANITIZE_STRING);
    $addStudent = $studenti->createStudent($ime, $prezime);
}
?>
```

Kod 8.9: Provjera i spremanje POST varijable za stvaranje novog studenta

Prvo se provjerava da li u POST varijabli postoji vrijednost pod imenom „noviStudent“. To je ime gumba za spremanje podataka u createStudent.php formi. Ako smo kliknuli gumb, spremili smo podatke u POST varijablu. Nakon toga, stvara se nova instanca klase Studenti. Podatke tipa *string* koje smo dobili preko POST metode filtriramo pomoću *filter_var()* PHP funkcije kako bi bile sigurne za unos u bazu. Kao zadnji korak, funkciji *createStudent()* šaljemo očišćene podatke spremljene u varijablama.

```php
public function createStudent($ime, $prezime) {

        if (empty($ime) || empty($prezime)) {
            return '<div class="alert alert-danger"><strong> Polja moraju
biti popunjena!</strong></div>';
        }

        $query = "INSERT INTO studenti (ime, prezime) VALUES (:ime,
:prezime)";

        $result = $this->db->pdo->prepare($query);
        $result->execute([
            'ime' => $ime,
            'prezime' => $prezime,
        ]);

        $rowsInserted = $result->rowCount();

        if ($rowsInserted == 1) {
            $student_id = $this->db->pdo->lastInsertId();

            $query = "INSERT INTO dolasci (student_id, datum) SELECT
:student_id, CURDATE() WHERE EXISTS(SELECT datum FROM dolasci WHERE
datum=CURDATE())";

            $result = $this->db->pdo->prepare($query);
            $result->execute([
                 'student_id' => $student_id,
            ]);
```

```
            return '<div class="alert alert-success"><strong>Student
uspješno spremljen!</strong></div>';
            }
}
```

Kod 8.10: Funkcija za stvaranje novog studenta

U slučaju da polja nisu bila popunjena, odnosno da podaci $ime i $prezime budu prazni, funkcija odmah vraća upozorenje da polja moraju biti ispunjena. Ako to nije slučaj, SQL upit za umetanje imena i prezimena u tablicu Studenti se sprema u varijablu $query. Podaci se umeću pomoću *prepare()*, *execute()* PDO metoda, a u varijablu $rowsInserted se sprema broj redaka koji je umetnut. Zatim se provjerava da li je redak doista i umetnut.

U drugom dijelu *createStudent()* metode iskorištava se činjenica da tablice studenti i dolasci dijele zajednički stupac – student_id. U tablici studenti student_id je primarni ključ, pa tu vrijednost dohvaćamo pomoću PDO metode *lastInsertId()*. Sada možemo spremiti vrijednosti i u tablicu dolasci. Spremamo uvijek trenutni datum, jer ako se sjetimo, formi se može pristupiti samo sa stranice s trenutnim datumom, te spremamo dobiveni student_id. Funkcija vraća poruku koja se sprema u $addStudent varijablu na stranici createStudent.php. Poruku možemo na stranici ispisati pomoću PHP-ovog *echo* konstrukta.

Kao zadnji korak, mijenjamo dijelove koda u *createStudent()* privatnim funkcijama radi preglednosti i čitljivosti koda.

```
public function createStudent($ime, $prezime) {

        if (empty($ime) || empty($prezime)) {
            return '<div class="alert alert-danger">
<strong>Polja moraju biti popunjena!</strong></div>';
        }

        $rowsInserted = $this->_insertIntoStudenti($ime, $prezime);

        if ($rowsInserted == 1) {
            $student_id = $this->db->pdo->lastInsertId();
            $this->_insertIntoDolasci($student_id);

            return '<div class="alert alert-success">
<strong>Student uspješno spremljen!</strong></div>';
        }
    }
```

Kod 8.11: Uređena funkcija za stvaranje novog studenta

8.6. Iščitavanje podataka iz baze i brisanje studenta

Nakon što su uneseni, svi studenti se moraju vidjeti na početnoj stranici aplikacije. Podatke moramo iščitati iz baze. Za to će biti zadužena *readStudents()* funkcija. U slučaju da smo unijeli krivi podatak, studenta ćemo moći trajno izbrisati iz baze pomoću *deleteStudent()* funkcije.

```
public function readStudents() {

    $query = 'SELECT * FROM studenti';
    $results = $this->db->pdo->query($query);

    if( $results) {
        return $results;
    }
}
```

Kod 8.12: Funkcija za ispis svih spremljenih studenata

Funkcija readStudents() je prilično jednostavna. Sastoji se od upita za sve podatke iz tablice studenti koji se dobivaju pomoću PDO *query()* metode. Ta metoda vraća objekt koji sadrži redove tablice u slučaju da je upit izvršen.

```
$studenti = new Studenti();
$results = $studenti->readStudents();
```

Kod 8.13: Instanciranje klase i izvršavanje funkcije za ispis studenata

Te rezultate spremamo na stranici index.php u varijablu $results. Sada možemo ispisati sve studente koje smo spremili u bazu. Hard kodirane podatke unutar tijela tablice potrebno je zamijeniti sljedećim kodom:

```php
<?php
$i = 0;
foreach ($results as $result)
{
?>
<tr>
    <td><?php echo ++$i; ?></td>
    <td><?php echo $result['ime']; ?></td>
    <td><?php echo $result['prezime']; ?></td>
    <td>
        <label class="radio-inline">
            <input type="radio" name="dolazak[<?php echo
$result['student_id']; ?>]" value="tu" required>+
        </label>
        <label class="radio-inline">
            <input type="radio" name="dolazak[<?php echo
$result['student_id']; ?>]" value="odsutan">-
        </label>
    </td>
    <td><a href="index.php?id=<?php echo $result['student_id']; ?>"
class="btn btn-danger btn-sm"><i class="fa fa-trash" aria-
hidden="true"></i></a>
    </td>
</tr>
<?php
}
?>
```

Kod 8.14: Dinamično ispisivanje studenata iz baze podataka

Kroz objekt dobiven *query()* metodom možemo iterirati uz pomoć PHP *foreach* petlje. Retke tablice ispisujemo direktno u HTML-u. Svakom podatku iz retka tablice možemo pristupiti imenom njegovog stupca u bazi. Na stranici se prvo se ispisuje brojač svakog retka, zatim ime i prezime studenta iz tablice studenti, nakon toga dva radijska gumba i u zadnjem stupcu retka ikona kantice za smeće, odnosno gumb za brisanje. Za svaki redak tablice, radijski gumbi moraju imati isto ime. To smo osigurali tako što smo kraj imena odlazak ispisali $result['student_id'], čime dobivamo ime odlazak plus jedinstveni identifikator studenta u tom retku.

Što se tiče gumba za brisanje, u URL-u kod klika prenosimo identifikator studenta kojeg želimo izbrisati. Na stranicu je potrebno dodati kod koji provjerava da li postoji $_GET['id'] varijabla. Nakon toga možemo funkciji deleteStudent poslati tu vrijednost. Za kraj, potrebno je osvježiti (eng. *refresh*) stranicu kako bi rezultati postali vidljivi. Kod bi trebao izgledati ovako:

```php
if (isset($_GET['id'])) {
    $id = filter_var($_GET['id'], FILTER_VALIDATE_INT);

    $studenti->deleteStudent($id);
    header('Location:index.php');
}
```

Kod 8.15: Brisanje studenta preko GET identifikatora

Funkcija *delete()* klase studenti prima student_id i sastoji se od dva dijela, odnosno dvije podfunkcije. U prvom dijelu se briše student iz tablice studenti i vraća se broj izbrisanih redaka. Ako je redak izbrisan, u drugom dijelu brišu se podaci o tom studentu iz tablice dolasci.

```php
public function deleteStudent($student_id) {

    $rowsDeleted = $this->_deleteFromStudenti($student_id);

    if ($rowsDeleted == 1) {
    $this->_deleteFromDolasci($student_id);
    }
}
```

Kod 8.16: Funkcija za brisanje studenta

8.7. Spremanje dolazaka

Unesene dolaske šaljemo preko POST metode i prvo na početnoj stranici moramo provjeriti da li postoji POST varijabla pod tim imenom.

```php
if (isset($_POST['dolazak'])) {
    $dolasci = $_POST['dolazak'];
    $addAttendance = $studenti->addAttendance($dt, $dolasci);
}
```

Kod 8.17: Provjera i spremanje dolazaka preko POST metode

Nakon provjere, dolaske zajedno sa trenutnim datumom šaljemo *addAttendance()* metodi.

```php
public function addAttendance($datumDanas, $dolasci) {

        $dbDates = $this->getDBDates();

        foreach ($dbDates as $date) {
            if ($date['datum'] == $datumDanas) {
                return '<div class="alert alert-danger">
<strong>Dolasci za ovaj datum su već spremljeni!</strong>
</div>';
            }
```

```
        }

        $query = 'INSERT INTO dolasci( student_id, dolazak, datum)
VALUES( :student_id, :dolazak, CURDATE())';
        $result = $this->db->pdo->prepare($query);

        foreach ($dolasci as $student_id => $dolazak) {

            try {
                $assocArray = array('student_id' => $student_id,
'dolazak' => $dolazak);

                if ($dolazak == 'tu') {
                    $result->execute($assocArray);
                } elseif ($dolazak == 'odsutan') {
                    $result->execute($assocArray);
                }
            } catch (PDOException $e) {
                throw $e;
            }
        }

        return '<div class="alert alert-success"><strong>
Dolasci zabilježeni!</strong></div>';
    }
```

Kod 8.18: Funkcija za spremanje dolazaka u bazu podataka

Metoda *addAttendance()* prikazana u kodu gore, prvo sprema postojeće datume iz baze dobivene pozivom getDBDates() funkcije u varijablu $dbDates. Zatim se provjerava da li datum koji pokušavamo spremiti već postoji u bazi. Ako postoji, metoda *addAttendance()* vraća upozorenje. U slučaju da tvrdnja nije istinita, u varijablu $query se sprema upit za pohranjivanje podataka u tablicu dolasci. Upit se priprema za bazu pomoću *prepare()* metode. Nakon toga, dolaske iteriramo kroz petlju gdje vrijednost ključa dobivamo iz name vrijednosti gumba, a vrijednost dolaska iz *value* vrijednosti gumba. U slučaju da je vrijednost dolaska 'tu', ta se vrijednost sprema u bazu, a ako je vrijednost 'odsutan', onda se student sprema kao 'odsutan'. Na kraju se vraća poruka o uspjehu. Poruku ispisujemo u HTML kodu dokumenta, na način na koji je to izvedeno i u prijašnjim primjerima.

8.8. Pregled svih spremljenih datuma

Pregled svih spremljenih datuma ispisivati će se na stranici datumiView.php s mogućnošću izmjena dolazaka za pojedini datum prebacivanjem na stranicu updateStudenti.php. Datume

dohvaćamo pomoću funkcije korištene i u prošlom poglavlju, a to je *getDBDates()*. Ona dohvaća svaki datum spremljen u bazi, bez ponavljanja istih datuma.

```
public function getDBDates() {

    $query = 'SELECT DISTINCT datum FROM dolasci';

    $results = $this->db->pdo->query($query);

    return $results;
}
```

Kod 8.19: Funkcija za dohvaćanje datuma iz baze

Da bi ispisali datume, opet upotrebljavamo *foreach* petljom u tijelu tablice. U zadnjom stupcu se nalazi gumb pomoću kojeg u URL-u stranici updateStudenti.php prenosimo datum kojeg smo odabrali.

```
<?php
$i = 0;
foreach ($results as $result) {
?>
<tr>
    <td><?php echo ++$i; ?></td>
    <td><?php echo $result['datum']; ?></td>

    <td><a class="btn btn-primary btn-sm"
href="updateStudenti.php?dt=<?php echo $result['datum']; ?>"><i
class="fas fa-edit"></i></a></td>
</tr>
<?php
}
?>
```

Kod 8.20: Dinamično ispisivanje datuma

8.9.Izmjena dolazaka

```
$dt = $_GET['dt'];
$studenti = new Studenti();
$results = $studenti->getAllData($dt);
```

Kod 8.21: Spremanje datuma iz GET varijable i izvršavanje getAllData funkcije

Na stranici updateStudenti.php spremamo datum koji smo dobili GET metodom u varijablu $dt, te ga šaljemo funkciji *getAllData()*.

```
public function getAllData($datum) {

        $query = "SELECT studenti.*, dolasci.* FROM studenti INNER JOIN
dolasci ON studenti.student_id=dolasci.student_id WHERE datum=:datum";

        $result = $this->db->pdo->prepare($query);
        $result->execute([
            'datum' => $datum
        ]);
        return $result->fetchAll(PDO::FETCH_ASSOC);
   }
```

Kod 8.22: Funkcija za dohvaćanje svih spremljenih podataka po određenom datumu

Unutar funkcije prvo spremamo upit koji spaja tablice studenti i dolasci po podudarajućim student_id poljima u svakom retku, ali samo uz datum koji smo funkciji poslali.

```
public function updateAttendance($datum, $dolasci) {

        $query = "UPDATE dolasci SET dolazak=:dolazak WHERE
student_id=:student_id AND datum=:datum";
        $result = $this->db->pdo->prepare($query);

        foreach ($dolasci as $student_id => $dolazak) {
            try {
                $assocArray = array('dolazak' => $dolazak, 'student_id'
=> $student_id, 'datum' => $datum);

                if ($dolazak == 'tu') {
                    $result->execute($assocArray);
                } elseif ($dolazak == 'odsutan') {
                    $result->execute($assocArray);
                }

            } catch (PDOException $e) {
                throw $e;
            }
        }
    }
```

Kod 8.23: Funkcija za izmjenu dolazaka

Rezultate ispisujemo uz pomoć foreach petlje unutar HTML koda stranice kao i na početnoj stranici, uz jednu bitnu razliku – provjeravamo vrijednost polja dolazak kojeg smo dobili iz baze i označavamo gumb koji se podudara s tom vrijednošću.

```
<input type="radio" name="dolazak[<?php echo $result['student_id']; ?>]"
value="tu" <?php
       if ($result['dolazak'] == 'tu') {
           echo "checked";
       }
       ?> required><b>+</b>
<input type="radio" name="dolazak[<?php echo $result['student_id']; ?>]"
value="odsutan" <?php
       if ($result['dolazak'] == 'odsutan') {
           echo "checked";
       }
       ?>><b>-</b>
```

Kod 8.24: Isječak za ispisivanje spremljenih dolazaka iz baze

Za izmjenu dolazaka koristimo POST metodu. Na stranici provjeravamo da li su dolasci spremljeni u tu varijablu. U slučaju da dolasci postoje šaljemo datum dobiven GET metodom i izmijenjene dolaske funkciji *updateAttendance()*. Na kraju je potrebno osvježiti stranicu da bi se vidjeli izmijenjeni dolasci. Funkcija *updateAttendance()* je slična funkciji *addAttendance()*. Razlika je u SQL upitu koji započinje s ključnom riječi UPDATE i koji izmjenjuje dolazak u retku s identifikatorom studenta i datumom koji smo mu poslali. Također, nema potrebe za provjerom da li je datum već unesen.

Aplikaciju se može testirati sljedeći poveznicu koja se nalazi u prvom poglavlju praktičnog dijela. S obzirom da je pisana samo PHP programskim jezikom, prilično je spora i nije optimizirana za javnu upotrebu.

9. Zaključak

PHP je popularan skriptni jezik otvorenog koda (eng. open source) koji se izvršava na poslužitelju. Njegova jednostavnost ali i napredne mogućnosti, poput podrške za objektno orijentirano programiranje i rada s različitim bazama podataka, čine ga savršenim i za iskusne programere i za početnike.

OOP pomaže programerima u pisanju čišćeg koda kojeg je lakše mijenjati i ponovno koristiti. To je osobito važno u radu na složenim programskim sustavima. OOP počiva na izradi klasa i primjeraka (instanci) tih klasa koje se zovu objekti. Jedan program se sastoji od skupa objekata koji međusobnom komunikacijom rješavaju neki problem. Neki od bitnih principa u OOP-u su učahurivanje, nasljeđivanje polimorfizam. Programer bi trebao osigurati da osjetljivi podaci budu zaštićeni. Zbog toga se oni enkapsuliraju. Enkapsulacija se vrši kroz kontrolu pristupa. Nasljeđivanje predstavlja stvaranje novih klasa i objekata koristeći se postojećim klasama. Ono omogućava iskorištavanje postojećeg koda i proširivanje njegove funkcionalnosti. Polimorfizam se odnosi na sposobnost varijable, funkcije ili objekta da preuzme više oblika. To znači da možemo stvoriti više metoda s istim imenom koje će se ponašati različito i u skladu s tipom objekata s kojima se koriste.

Životni vijek objekta (eng. object lifetime) je vrijeme koje protekne između kreiranja objekta i njegovog uništavanja. Objekt koji ima sposobnost da nadživi proces koji ga je kreirao naziva se trajni (perzistentni) objekt (eng. persistent object). Jedna od najpopularnijih tehnika za ostvarivanje perzistencije je pohranjivanje objekata u relacijske baze podataka. Relacijske baze podataka predstavljaju skup organiziranih podataka spremljen u računalu na određen način. Sastoje se od skupa povezanih dvodimenzionalnih tablica odnosno relacija, a za obradu podataka koristi se Strukturirani upitni jezik – SQL. Sve relacijske baze podataka implementiraju funkcionalnosti za stvaranje, iščitavanje, ažuriranje i brisanje podataka. Jedan od najpopularnijih sustava za upravljanje relacijskim bazama podataka je MySQL.

Za praktični dio završnoga rada izradila sam PHP aplikaciju za praćenje dolazaka studenata na vježbe ili predavanja poštujući osnovne objektno orijentirane principe koji su obrađeni u teoretskom dijelu rada. Također, aplikacija ostvaruje perzistenciju podataka pohranjivanjem istih u MySQL bazu podataka. Za komunikaciju s bazom podataka i izdavanje CRUD naredbi aplikacija koristi PHP-ovu objektno orijentiranu PDO ekstenziju.

U Varaždinu, listopad 2019.

_____

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, PETRA GAL *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/~~diplomskog~~ (*obrisati nepotrebno*) rada pod naslovom OBJEKTNO ORIJENTIRANO PROGRAMIRANJE U PHP-U *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)
Petra Gal
(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, PETRA GAL *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/~~diplomskog~~ (*obrisati nepotrebno*) rada pod naslovom OBJEKTNO ORIJENTIRANO PROGRAMIRANJE U PHP-U *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)
Petra Gal
(vlastoručni potpis)

10. Bibliografija

[1] http://www.netakademija.hr/sto-je-php/, dostupno 28.07.2019.

[2] K. Žurbić, Uvod u objektno orijentirano programiranje, diplomski rad, informatički klub FUTURA, Dubrovnik, 2014.

[3] C. Collins, PHP This! A Beginners Guide to Learning Object Oriented PHP, First Edition, eBookIt, 2013.

[4] http://php.net/manual/en/language.oop5.basic.php, dostupno 21.03.2019.

[5] M. Doyle, Beginning PHP 5.3, Wrox, 2009.

[6] N. Begović, Primjena OOP principa u PHP programskom jeziku, završni rad, Sveučilište Josipa Jurja Strossmayera u Osijeku, Osijek, 2015.

[7] https://almirvuk.blogspot.com/2016/04/oop-nasljeivanje-objasnjeno.html, dostupno 27.03.2019.

[8] https://www.fer.unizg.hr/_download/repository/4_Nasljedjivanje_Polimorfizam.pdf. dostupno 29.03.2019.

[9] https://www.oreilly.com/library/view/web-database-applications/0596005431/ch04.html, dostupno 29.03.2019.

[10] M. Zandstra, PHP Objects, Patterns, and Practice, Fifth Edition, Apress, 2016.

[11] https://ngrgic.wordpress.com/2009/12/25/perzistencija-objekata/, dostupno 05.07. 2019.

[12] http://php.com.hr/66, dostupno 05.07.2019.

[13] D. Powers, PHP Object-Oriented Solutions, Apress, 2008.

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

Popis slika

Popis primjera kodova

Popis tablica