Aplikacija za online učenje

Dragić, Anna-Maria

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

APLIKACIJA ZA ONLINE UČENJE

Diplomski rad

Anna-Maria Dragić

Osijek, 2019.

SADRŽAJ

1. UVOD

Aplikacija je namijenjena studentima kako bi im se olakšalo studiranje i učenje na način da putem aplikacije mogu razmjenjivati korisne sadržaje vezane za fakultet, kao što su skripte ili neki korisni linkovi i tekstovi. Student nakon registracije i prijave ima dvije opcije, jedna je objavljivanje i čitanje korisnog sadržaja, druga je postavljanje i odgovaranje na pitanja u obliku foruma, no prije obje radnje prvenstveno je potrebno odabrati koji je fakultet. Dakle, student može postavljati sadržaje, te ih uređivati i brisati ukoliko bude imao potrebu, dok na forumu student može postaviti neko pitanje na koje mu drugi studenti mogu odgovoriti i pomoći pri čemu isto ima mogućnost uređivanja i brisanja svojih pitanja. Rad je podjeljen u 5 poglavlja pri čemu je prvo uvodno poglavlje. U drugom poglavlju su navedene i objašnjene tehnologije, razvojna okruženja i baza podataka koji su korišteni pri relizaciji rada. Potom se prelazi na glavni dio rada koji je poglavlje u kojem se opisuje ralizacija rada podijeljena na dva dijela- dio vezan za objave i dio vezan za forum. Nakon toga, u četvrtom poglavlju su prikazane funkcionalnosti aplikacije i baza podataka. Te na kraju je rezimirajuće poglavlje, zaključak.

1.1. Zadatak diplomskog rada

Aplikacija je namjenjena studentima za razmjenu korisnih sadržaja, kao što su skripte korisni linkovi itd. Svaki student bi odabrao koji je fakultet, te bi na osnovu toga mogao vidjeti sadržaj koji se veže za njegov fakultet. Također može stavljati neke korisne sadržaj, te ukoliko bude htio može ih poslije i obrisati. Bio bi omogućen chat preko kojeg bi neki student mogao otvoriti neku temu ili postaviti neko pitanje, a drugi bi mu mogli pomoći i odgovoriti.

2. KORIŠTENE TEHNOLOGIJE

Prilikom izrade aplikacije korištene su razne tehnologije, kao što su HTML, CSS, Javascript, PHP, te njihova razvojna okruženja Laravel i Bootstrap. Dok je za spremanje podataka korištena MySQL baza podataka.

2.1. HTML

HTML se koristi za dizajn i razvoj web apilkacija. Skraćenica HTML dolazi od engleskog naziva Hyper Text Markup Language i služi za stvaranje hipertekstualnih dokumenata koji su neovisni o platformi. HTML dokument je tekstualna datoteka koja sadrži elemente koje preglednik koristi za prikaz teksta, multimedijskih objekata i hiperveze. Web dokument je tekstualni dokument i kao takav može se uređivat u bilo kojem tekst editoru.

HTML je označni jezik (engl. markup language) što znači da ima kodove koji određuju izgled i stil unutar tekstualne datoteke (npr. podebljan tekst, odlomci, veličina teksta itd). Markup jezik se potpuno se razlikuje od drugih programskih jezika[1]. Dakle, HTML se sastoji od puno elemenata. Elementi ukazuju pregledniku kako će prikazivati sadržaj HTML datoteke. Elementi se predstavljaju oznakama koje mogu biti oznake za naslove (engl. headings) (<h1>, <h2>, <h3> itd), oznaka za paragraf (<p>), za tablice (<table>), forme (<form>) itd.

Trenutna verzija HTML-a koja se koristi je HTML 5 koji je u upotrebi od 2014-te godine. Početak svakog HTML dokumenta mora biti sa oznakom <!DOCTYPE html> što označava da se radi o HTML5 dokumentu. Nakon toga, sav kod mora biti unutar <html> i </html> oznaka. Nadalje se dokument dijeli na dio koji se piše unutar zaglavlja s <head> oznakom i dio unutar tijela dokumenta koje se označava s <body> oznaka. Zaglavlje sadrži meta informacije o dokumentu, dok tijelo sadrži vidljiv sadržaj web stranice.

HTML oznake (tagovi) su imena elemenata smješetna između dviju kutnih zagrada. Oznake su obično u paru prilikom čega se prva oznaka zove startna ili početna oznaka i piše se <oznaka>, a druga je krajnja ili završna oznaka i piše se s kosom crtom: </oznaka>[2]. Neke oznake nemaju završni dio, te se nazivaju prazne oznake, takva oznaka je
. HTML oznake nisu osjetljive na velika i mala slova, dakle možemo napisati <h1> ili <H1>.

Komentari su bitan dio HTML dokumenta za programere jer im pomažu u lakšem snalaženju u kodu, u HTML komentari se umeću između oznaka <!-- i --> [1].

2.2. CSS

Skraćenica CSS dolazi od engleskog naziva Cascading Style Sheets. Korištenjem CSS-a moguća je primjena raznih stilova kako bi web stranica izgledala onako kako želimo. CSS stil elementima se može dodavati unutar <style> oznake. Kada želimo isti stil primjeniti na cijelu web stranicu, a ne samo na jednu stranicu, bolja opcija je pisanje odvojenog CSS dokmenta i uključivanje istog u <head> dio svake HTML datoteke u koju želimo. Primjer uključivanja CSS-s u HTML pomoću <link> oznake:

<link rel='stylesheet' type='text/css' href='styles.css'>

Svaka izjava u CSS pravilu počinje sa slektorom, koji je element na koji će pravilo biti primjenjeno, npr. selektor može biti body, p, h1, h2, h6 kojemu se dodjeljuje neko svojstvo kao boja fonta, veličina fonta itd.[3]. Osim selektora, CSS pravila se sastoje i od deklaracijskog bloka. Deklaracijski blok su vitičaste zagrade unutar kojih se nalaze deklaracije. Svaka deklaracija sastoji se od svojstva, dvotočke (:) i vrijednosti. Između svake dvije uzastopne deklaracije mora se nalaziti točka zarez (;).

Web developer može definirati stil za svaki HTML element i primjeniti ga na koliko god želi web stranica. ID selektor koristi ID atribut HTML elementa kako bi detektirao jedinstven element budući da bi ID elementa trebao biti jedinstven na svakoj stranici, što znači da se ID selektor koristi za odabir jednog elementa. Kako bi se naznačilo da želimo element s određenim ID-om piše se # oznaka ispred željenog ID-a, npr. *#myid { color: red }*. Klasni selektor, u drugu ruku, odabire elemente sa određenim „class“ atributom. Kako bi izabrali element sa određenom klasom pišemo točku (.) ispred naziva klase, npr. *.myclass { color: red }*.Grupni selektori se koriste da minimaliziraju kod ukoliko više elemenata ima ista svojstva, bitno je samo svaki element odvojiti zarezom, npr.: *h1, h2, p { color: red }*.

Komentari u CSS dokumentu se pišu između oznaka /*i */[4].

CSS ima mnoge prednosti, kao što je već navedeno s vanjskim CSS dokumentom moguće je odvojiti sadržaj stranice i njenu prezentaciju tj. izgled. Također, budući da je jedan CSS dokument moguće povezati s više HTML dokumenata, ukoliko je potrebno napraviti promjenu na HTML dokumentima postupak je dosta skraćen time što se promjena napravi samo u navedenom CSS dokumentu što pomaže u smanjenju vremena i truda utrošenog na održavanje web stranica[1].

2.3. JavaScript

JavaScript omogućava dinamičku funkcionalnost web stranica. Kada na web stranici postoji stvar (element) koja se pojavljuje kada mišem prelazite preko stavke u pregledniku ili ako vidite novi tekst, boje ili slike koje se u tom trenutku pojavljuju na stranici, ili ukoliko postoji mogućnost povlačenja objekta na novu lokaciju - sve te radnje se obavljaju pomoću JavaScripta. Točnije rečeno, JavaScript je skriptni jezik na strani klijenta koji se izvršava isključivo unutar web preglednika. Kako bi pozvali Javascript unutar HTML dokumenta postavlja se između oznaka <script> i <script>, npr.:

<script type="text/javascript" src="script.js"></script>

JavaScript dokument je moguće uključiti unutar tijela (engl. body) HTML elementa ili unutar zaglavlja (engl. head). Unutar zaglavlja se uključuje ukoliko želimo da se JavaScript skripta izvrši kada se učita stranica[3].

Leksička struktura programskog jezika je skup pravila koja određuju kako pisati kod koristeći se određenim jezikom. JavaScript se temelji na akcijskom modelu World Wide Weba. Elementi na web stranicama, kao što su gumb ili checkbox mogu pokrenuti akcije ili događaje. Kada se dogodi jedan od tih događaja izvršen je određeni dio JavaScript-a, obično JavaScript funkcija. Za razliku od HTML-a, Javascript razlikuje velika i mala slova, te na to treba obratiti pažnju prilikom pisanja koda. Nadalje, jedno od pravila pisanja JavaScript koda je to da nije obavezno pisati točku-zarez (;) ukoliko su tvrdnje u odvojenim linijama. Komentati se u Javascriptu pišu kao i u CSS-u; ukoliko želimo jedan red komentirati možemo to učiniti s dvije kose crte : //Ovo je komentar. Dok, ukoliko želimo veći dio koda komentirati onda komentar stavljamo između oznaka /* i */.

Varijable u JS se definiraju korištenjem ključne riječi *var* ispred naziva varijable. Imena varijabli moraju početi sa slovom ili donjom crtom a mogu sadržati brojeve. Jedna od najvećih razlika JavaScripta i drugih programskih jezika je ta da JavaScript nema eksplicitne tipove podataka. Dakle, nema načina da se specificira da određena varijabla predstavlja integer, string ili broj s pomičnim zrezom. Ista varijabla može biti različito interpretirana u različitim kontekstima[1].

2.4. PHP

PHP se može koristiti za dodavanje sadržaja u HTML datoteke. Dok HTML skripte obrađuje izravno web preglednik, PHP skripte izvršava web poslužitelj, a rezultirajući HTML se šalje pregledniku. PHP omogućava razvoj dinamičkih i interaktivnih web stranica.

Kao i kod svakog C-orijentiranog programskog jezika, svaka tvrdnja mora završavati sa točkom-zarezom (;). Dok se završna oznaka koristi da završi posljednji redak koda PHP bloka. Ukoliko se posljednji redak PHP koda završava točkom-zarezom, završna oznaka nije obvezna ako nakon tog koda nema više koda. No, ukoliko postoji kod nakon određenog bloka koda onda je završna oznaka obavezna.

Varijablama u PHP-u se može pristupiti preko dinamičkih naziva varijabli. Ime varijable može biti pohranjeno u drugoj varijabli, dopuštajući da joj se može dinamički pristupati[6].

PHP kod se piše između dvije oznake, to su početna oznaka <?php i završna ?>. Bitan simbol u PHP-u je $ koji se mora nalaziti ispred svih varijabli čime PHP parsiranje postaje brže. Imena varijabli moraju početi sa slovom ili donjom crtom i mogu sadržati brojeve, a ne smiju sadržavati razmake. PHP razlikuje velika i mala slova što znači da $Ime nije isto što i $ime, to su dakle dvije različite varijable. Varijable ne moraju biti deklarirane prije upotrebe jer PHP uvijek pretvara varijable u traženi tip prema njihovom kontekstu kada im se pristupa.

Konstante su slične varijablama, razlika je kako i sam naziv kaže ta što su one konstantne tj. jednom kada se definiraju njihova vrijednost se više ne može mijenjati. Npr. konstante se koriste za definiranje lokacije root poslužitelja:

define("ROOT_LOCATION", "/usr/local/www/");

Komentati se pišu na način ukoliko želimo jedan red komentirati možemo to učiniti s dvije kose crte : //Ovo je komentar. Dok, ukoliko želimo veći dio koda komentirati onda komentar stavljamo između oznaka /* i */[3].

2.5. Razvojna okruženja

2.5.1. Bootstrap

Bootstrap je najpoznatije besplatno HTML, CSS i Javascript razvojno okruženje za razvoj responzivnih web stranica. Uključuje HTML i CSS dizajne za forme, gumbe, tablice, navigaciju itd., kao i opcionalne JavaScript dodatke. Pojam responzivnog dizajna predstavlja dizajn web stranice koji omogućuje automatsko prilagođavanje stranice uređaju na kojem se prikazuje, od malih mobitela do velikih zaslona. Najnovija verzija je Bootstrap 4 koji uvodi poboljšanja u odnosu na prethodnu verziju u vidu novih komponenti, bržih stilova i veće responzivnosti.

Bootstrap je moguće integrirati tako što se u zaglavlje HTML dokumenta stavi link na bootstrap.min.css datoteku ili na način da se skine bootstrap dokument sa njihove stranice[5].

2.5.2. Laravel

Laravel je open-source PHP razvojno okruženje (framework). Koristi se za razvoj web aplikacija upotrebom MVC (model-view-controller) arhitekture. MVC omogućuje podjelu aplikacije na više dijelova, razdvaja poslovnu i prezentacijsku logiku. Model se zasniva na stvarnom svijetu i njegovim objektima. View je dio u kojem je definiran dio koda za prikaz korisniku. Controller je veza između modela i pogleda. Dakle, prvenstveno pri pokretanju aplikacije se poziva kontroler jer se njemu prvo predaje zahtjev, te on povezuje tražene modele i poglede.

Laravel ima brojne prednosti, neke od njih su to što omogućava sinkronizaciju koristeći migracije baze podataka i schema builder. Također, Laravel koristi moderne alate kao što je rutiranje i ORM, jednostavnu autentifikaciju i validaciju, omogućava jednostavno održavanje stranice itd.

Laravel je moguće koristiti na verzijama PHP-a 5.5.9 pa na dalje.

Laravel sadrži .env datoteku koja sadrži bitne informacije kao što su sql host, korisničko ime i lozinku.

Struktura Laravel direktorija sadrži nekoliko dijelova, a to su:

- App - glavni dio aplikacije
- Config – dio u kojem se nalaze konfigracijske datoteke
- Database – u ovom dijelu se nalaze migracije baze podataka
- Public – sadrži JavaScript, CSS i slike ili uploadane datoteke
- Resources – sadrži dio vidljiv korisniku tj. podatke za pogled
- Storage – sadrži datoteke koje stvara razvojno okruženje[7].

2.6. MySQL baza podataka

Baza podataka je strukturirana kolekcija zapisa ili podataka spremljenih u računalni sustav i organiziranih na takav način da se mogu lako pretraživati i da se može brzo preuzeti.

SQL skraćenica predstavlja Structured Query Language. Ovaj jezik je omogućuje jednostavne zahtjeve na bazu podataka kao na primjer dohvaćanje naslova knjige iz određene baze podataka gdje je točno specificiran autor:

SELECT title FROM publications WHERE author = 'Charles Dickens';

MySQL baza podataka sadrži jednu ili više tablica pri čemu svaka sadrži zapise ili retke. Unutar tih redatak nalaze se različiti stupci ili polja koja sadrže same podatke.

Tri su načina kojima se može komunicirati sa MySQL bazom, a to su:

- Korištenje komandne linije
- Web sučelje kao što je phpMyAdmin
- Pomoću programskog jezika kao što je PHP.

MySQL ne razlikuje velika i mala slova, što znači ukoliko napišemo CREATE ili create dogoditi će se isto- kreirati će se baza podataka. No, preporučuje se korištenje velikih slova za pisanje naredbi. Nadalje, na Linux-u i OS X-u treba voditi računa pri pisanju imena tablica jer razlikuju velika i mala slova, no na Windowsu to nije slučaj. No, preporučuje se pisanje imena tablica malim slovima.

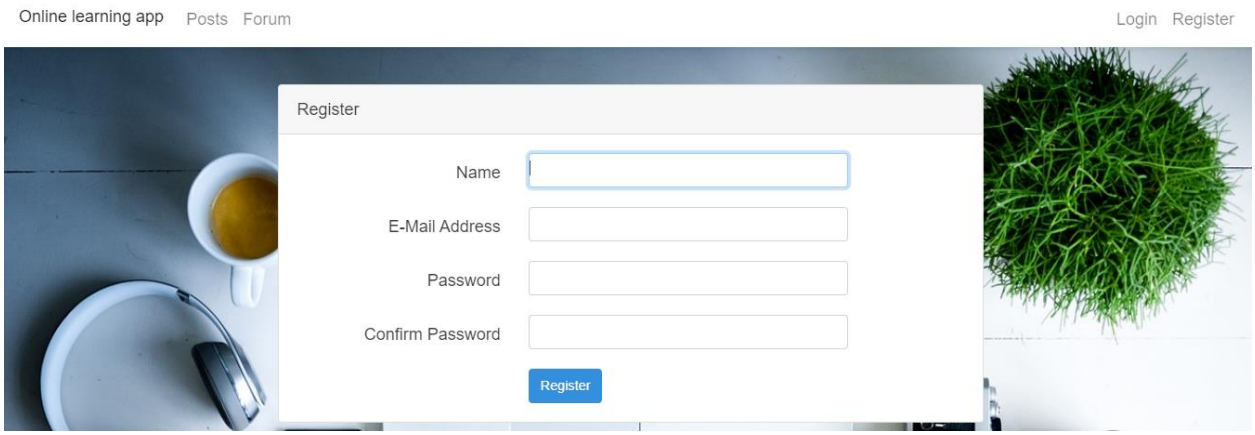
Postoje razne vrste podataka u MySQL bazi. Bitno je znati kojem podatku treba dati koju vrstu kako bi smo mogli spremati željeni sadržaj na ispravan način. Za spremanje stringa koriste se CHAR i VARCHAR vrste podataka. VARCHAR dolazi od engleskog naziva VARiable lenght CHARacter string, te se njime definira maksimalna dužina stringa koja se može spremiti u to polje. Dok CHAR koristimo kada želimo zadati točno koliko bitova podatak zauzima. Ukoliko spremimo podatak s manjim brojem znakova, ostatak se nadopunjava praznim mjestima. CHAR je efektivniji ukoliko je veličina podataka slična u svim zapisima, a VARCHAR ukoliko veličia podataka dosta varira. MySQL podržava i numeričke tipove podataka od jednog bajta (TINYINT) do brojeva s dvostrukom preciznošću s pomičnim zarezom (DOUBLE). Podržava i vrste podataka kojima se prikazuju datum i vrijeme, kao i tipove podataka koji želimo da se automatski povećavaju (koriste se npr. za ID korisnika u bazi, ID zadatka itd) tj. kada želimo da svaki redak u bazi bude jedinstven.

Indeksi se kreiraju kako bi postigli brže pretraživanje baze podataka, a mogu se kreirati pri kreiranju tablice ili naknadno. Postoje različiti tipovi indexa kao INDEX, PRIMARY KEY i FULL TEXT. Ne mora svaki podatak imati indeks. U tablicama se većinom kreira primarni ključ za koji se često uzima ID stupac jer je on jedinstven[3].

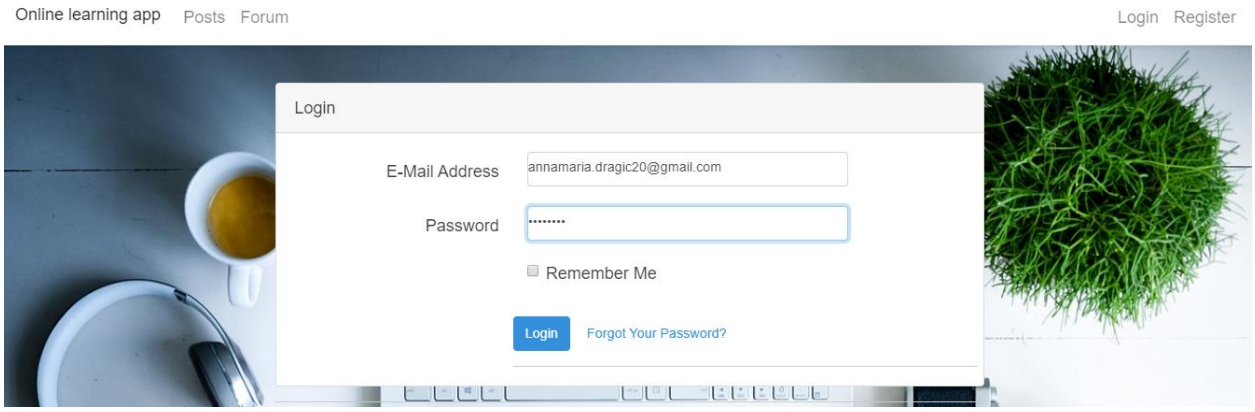
3. REALIZACIJA APLIKACIJE

Aplikacija se sastoji iz dva glavna dijela, to su dio na koji prijavljeni korisnici postavljaju korisne sadržaje u obliku Postova, te drugi dio gdje je omogućeno postavljanje pitanja i odgovaranje na pitanja postavljena od strane drugih korisnika u obliku Foruma.

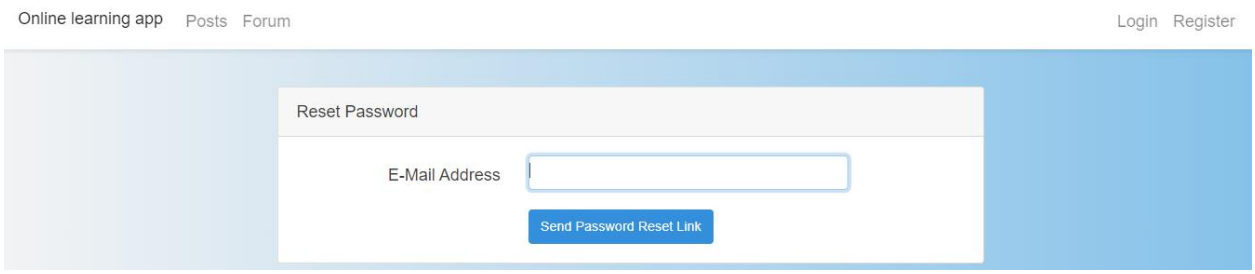
Prvenstveno da bi bilo koju od navedenih radnji mogao obavljati korisnik se mora prijaviti u aplikaciju (ili registrirati ukoliko već nije registriran). Naredba *php artisan make:auth* stvara sve dijelove MVC arhitekture za korisnika (User), stvara model User, potrebne kontrolere, dokumente koji definiraju izgled stranica za prijavu i registraciju, te potrebne migracije za bazu podataka. Na slikama 3.1. i 3.2. prikazani su izgledi stranica za registraciju i prijavu. Ukoliko je korisnik zaboravio lozinku pritiskom na „Forgot your password?“ link mu se pruža prilika da na svoju e-mail adresu dobije novu lozinku kako je prikazano na slici 3.3.



***Slika 3.1**.-Izgled stranice za registraciju*

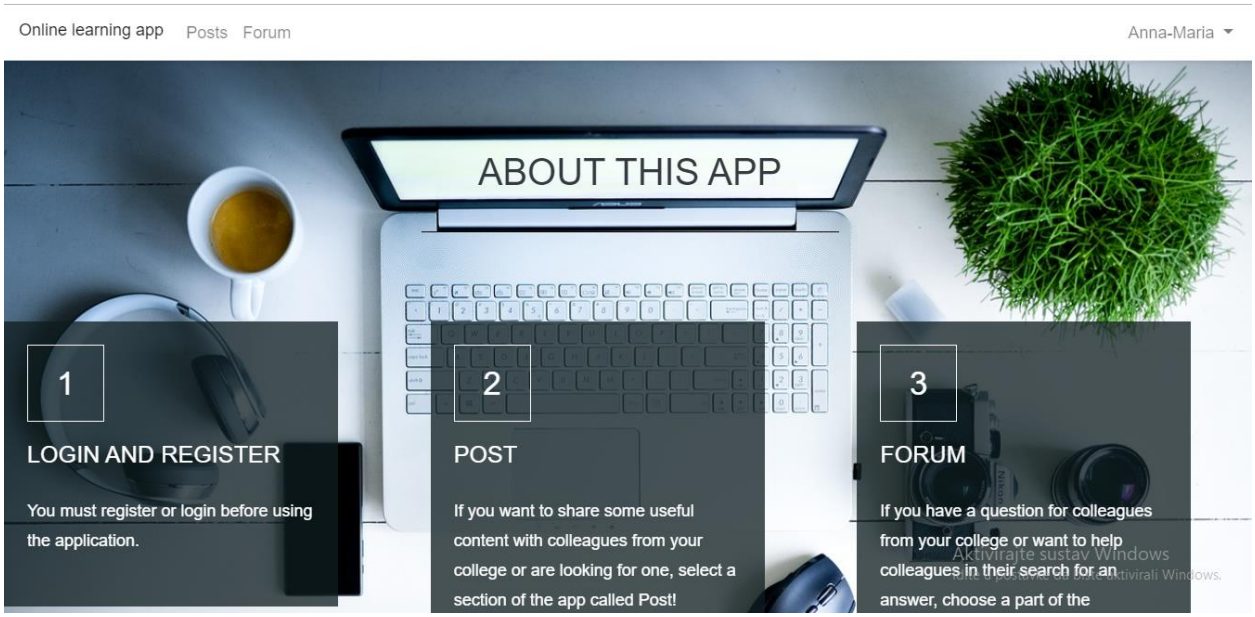


***Slika 3.2**.-Izgled stranice za prijavu*



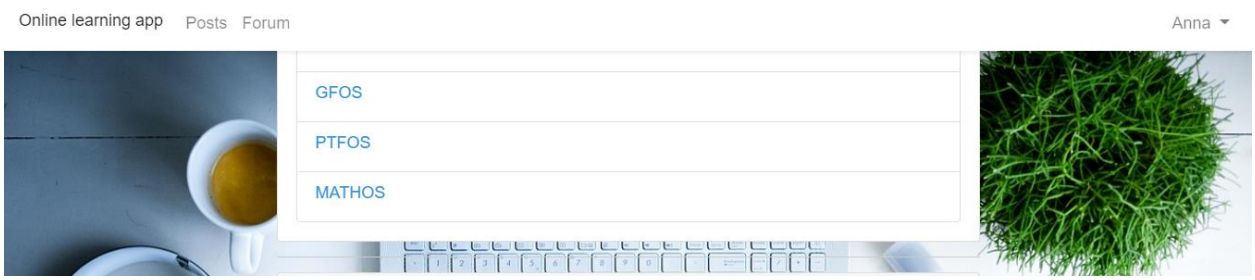
Slika 3.3. *- Mogućnost dobivanja nove lozinke za korisnike koji su ju zaboravili*

Nakon registracije ili prijave otvara se početna stranica koju je moguće vidjeti i prije same prijave u aplikaciju. Na početnoj stranici ispisane su informacije koje bi korisniku trebale olakšati korištenje aplikacije.



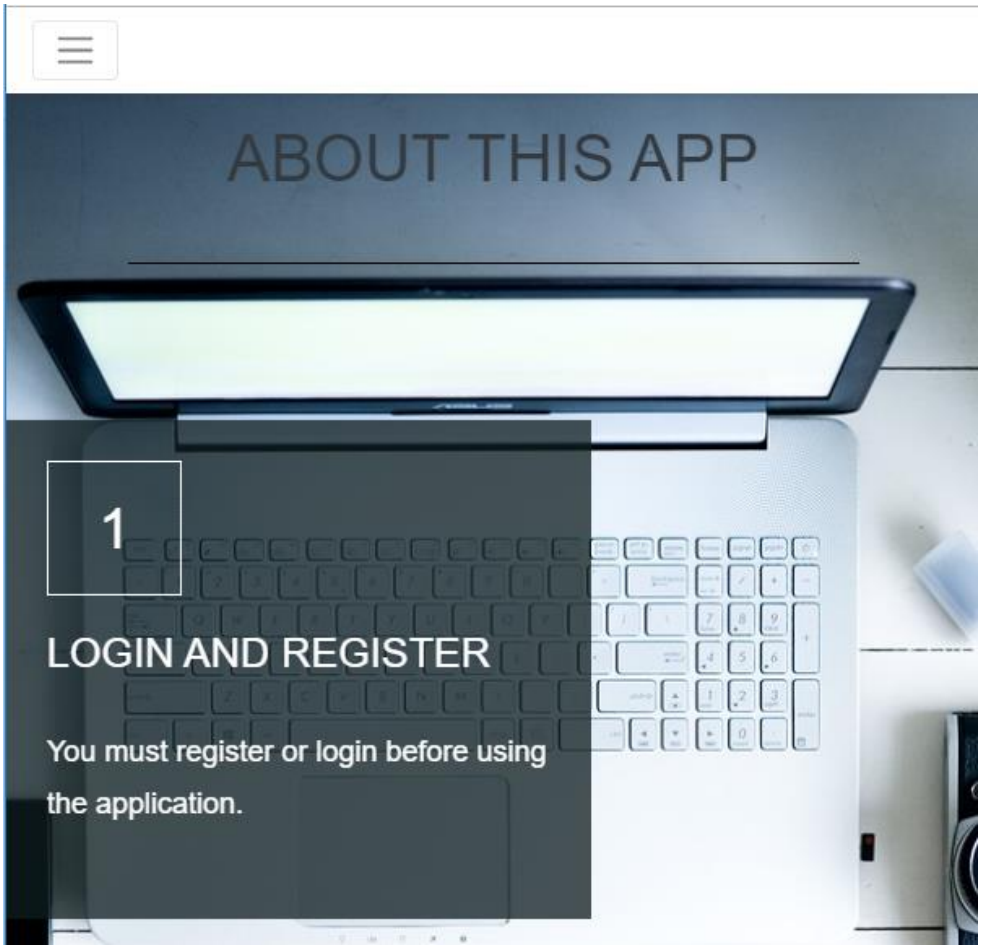
Slika 3.4. *-Početna stranica*

Na svakoj od stranica aplikacije nalazi se navigacijska traka koja je fiksna, dakle ostaje na vrhu stranice pri listanju sadržaja kao što se može vidjeti na slici 3.5. Navedeno svojstvo je realizirano dodavanjem „sticky-top“ klase unutar <nav> elementa.

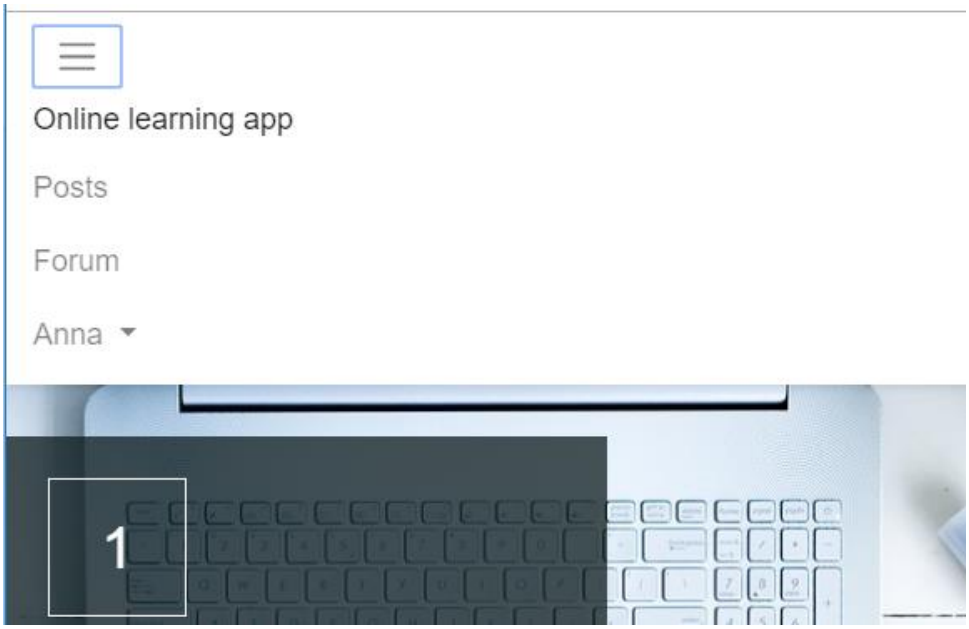


Slika 3.5. *-Navigacijska traka fiksirana na vrh stranice*

Također, bitno svojstvo aplikacije je responzivnost, prikaz se prilagođava uređaju na kojem se otvara stranica. Na slici 3.6. je prikazan izgled aplikacije pri manjem ekranu pri čemu možemo vidjeti da se elementi navigacijske trake nalaze u tzv. „hamburgeru“, te će se prikazati tek nakon klika na isti što je prikazano na slici 3.7.



Slika 3.6. *-Responzivnost aplikacije*



Slika 3.7. *-Responzivnost navigacijske trake*

Navigacijska traka je zajednička za sve stranice, te je iz toga razloga definirana u app.blade.php pogledu koji je pozvan na početku svih ostalih blade-ova. Tu je, dakle, definirano koje se stranice otvaraju klikom na pojedini dio navigacijske trake, te je definirano da se prijavljenom korisniku prikaže njegovo ime na desnoj strani trake s mogućnošću odjave, a korisnicima koji nisu prijavljeni da se prikažu gumbovi za prijavu i registraciju.

```
<!-- Left Side Of Navbar -->
<ul class="navbar-nav mr-auto">
 <a class="navbar-brand" href="{{ url('/') }}">Online learning app</a>
<ul class="navbar-nav mr-auto">
    <li class="nav-item ">
        <a class="nav-link" href="{{ url('/postthread') }}">Posts <span class="sr-only">(current)</span></a>
    </li>
    <li class="nav-item">
        <a class="nav-link" href="{{ url('/thread') }}">Forum</a>
    </li>
</ul>
</ul>
```

Slika 3.8. *-Lijeva strana navigacijske trake*

```
<ul class="navbar-nav ml-auto">
    <!-- Authentication Links -->
    @guest
        <li class="nav-item">
            <a class="nav-link" href="{{ route('login') }}">{{ __('Login') }}</a>
        </li>
        @if (Route::has('register'))
            <li class="nav-item">
                <a class="nav-link" href="{{ route('register') }}">{{ __('Register') }}</a>
            </li>
        @endif
    @else
        <li class="nav-item dropdown">
            <a id="navbarDropdown" class="nav-link dropdown-toggle" href="#" role="button"
            data-toggle="dropdown" aria-haspopup="true" aria-expanded="false" v-pre>
                {{ Auth::user()->name }} <span class="caret"></span>
            </a>

            <div class="dropdown-menu dropdown-menu-right" aria-labelledby="navbarDropdown">
                <a class="dropdown-item" href="{{ route('logout') }}"
                   onclick="event.preventDefault();
                                 document.getElementById('logout-form').submit();">
                    {{ __('Logout') }}
                </a>

                <form id="logout-form" action="{{ route('logout') }}" method="POST"
                style="display: none;">
                    @csrf
                </form>
            </div>
        </li>
    @endguest
```


Slika 3.9. *-Desna strana navigacijske trake*

U svrhu boljeg izgleda početnih stranica dodan je parallax efekt koji je omogućio da se sadržaj stranice pomiče brže od pozadinske slike pri listanju stranica. Kako bi parallax funkcionirao potrebno je dodati skripte za jquery, parallax i bootstrap u „body“ dio. Te je potrebno definirati da paralax efekt obuhvaća sadržaj stranice.

```
<div class="parallax-window" data-parallax="scroll" data-image-src="img/computer.jpg">

<main class="py-4">
    @yield('content')
</main>

</div>
```

Slika 3.10. *-Parallax efect*

Kako bi aplikacija i njeni elementi imali željeni izgled u <head> dio app.blade.php-a uključena je main.css skripta. Na narednim slikama prikazani su neki dijelovi te skripte, slika 3.11. prikazuje kako je uređen <body> element aplikacije pri čemu je dodana nešto jednostavnija pozadina koja će se primjenivati na stranicama na kojim nije primjenjen parallax, za pozadina je izvedena kao

gradijent. Na slici 3.12 prikazano je uređenje dijela welocme.blade.php stranice, točnije dijela koji prikazuje opis i način korištenja aplikacije unutar odvojenih kutijica (engl. box). Definirani su širina i dužina elementa, definirana je transparentnost tj. providnost pozadine unutar elementa, definirana je lokacija teksta unutar elementa, veličina naslova elementa postavljena je na velika slova. Na slici 3.13. prikazan je rezultat dobiven primjenom navedenog css dokumenta na nevedeni dio html stranice.

```css
body {
  font-family: "Open Sans", Arial, Helvetica, sans-serif;
  font-size: 18px;
  font-weight: 400;
  color: #3f3f3f;
  overflow-x: hidden;
  background: #85C1E9;
  background: -webkit-linear-gradient(to left, #85C1E9, #F2F3F4 );
  background: linear-gradient(to left, #85C1E9, #F2F3F4 );
  min-height: 100vh;

}
```

Slika 3.11. *- Dio main.css datoteke - svojstva body elementa*

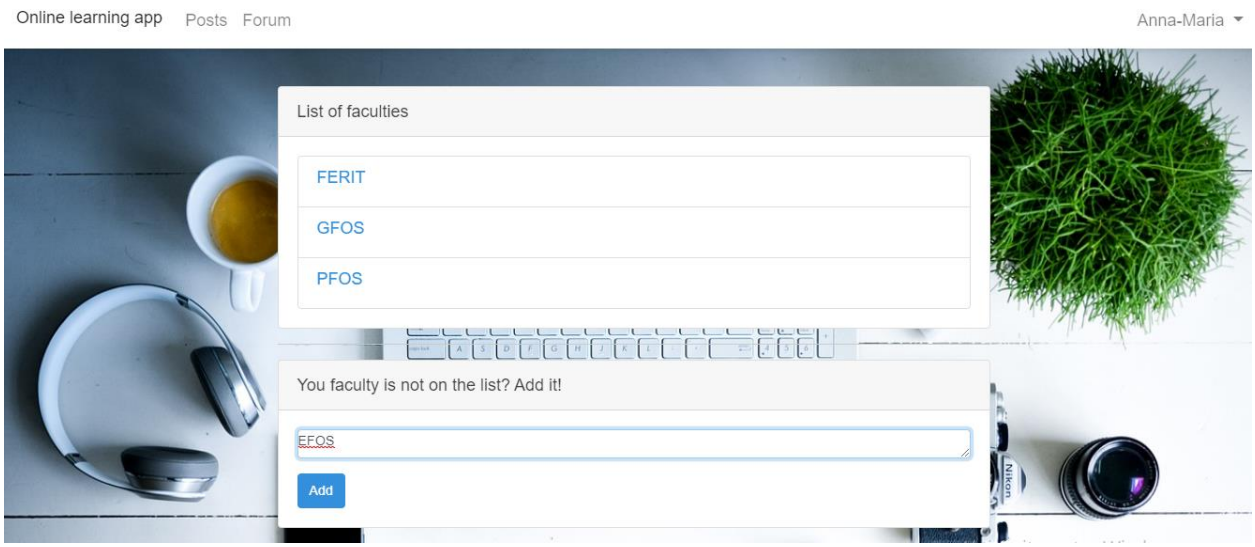
```css
.tm-about-box {
  padding: 25px;
  max-width: 360px;
  margin-bottom: 30px;
}
.tm-bg-black-transparent {
  background-color: rgba(15, 29, 32, 0.76);
  color: white;
}

.tm-about-number-container {
  width: 83px;
  height: 83px;
  font-size: 2.2rem;
  border: 1px solid white;
  display: flex;
  align-items: center;
  justify-content: center;
  margin-bottom: 20px;
}

.tm-about-name {
  margin-bottom: 30px;
  text-transform: uppercase;
}

.tm-about-description {
  margin-bottom: 30px;
  line-height: 1.8;
}
```

Slika 3.12. *- Dio main.css datoteke - about box*

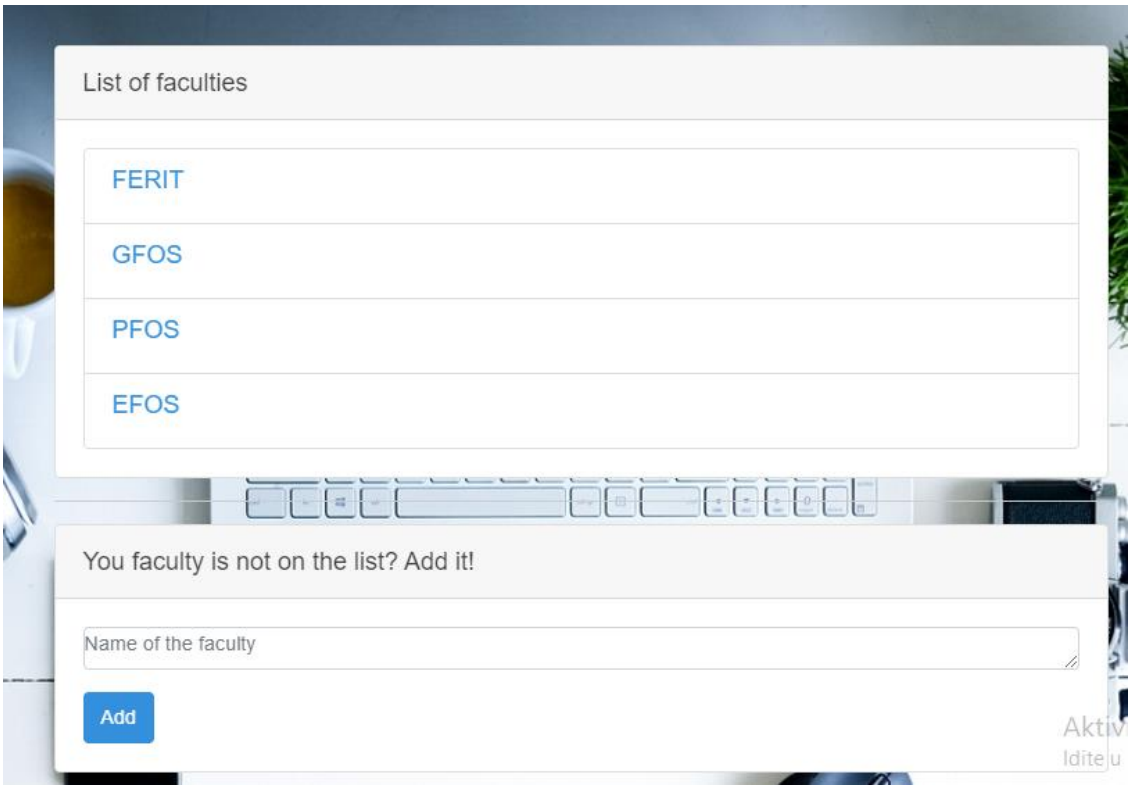
Slika 3.13. *- Element uređen pomoću css-a*

3.1. Objave – Posts

Ukoliko korisnik na navigacijskoj traci odabere Post dio aplikacije prvenstveno mu se otvara izbornik s listom ponuđenih fakulteta, također postoji mogućnost dodavanja fakulteta ukoliko njegov fakultet nije na listi. Navedeno je prikazano na slikama 3.14. i 3.15.



Slika 3.14. *-Lista ponuđenih fakulteta*



Slika 3.15. *-Dodan novi fakultet na listu*

Svaki fakultet predstavlja jedan Thread, kako bi se thread-ovi mogli spremati u bazu i prikazivati na stranici potrebno je definirati PostThread model, kontroler pod nazivom PostThreadContoller, definirati rute, migracije u bazu, te u pogledu postthread.blade.php definirati potrebne akcije.

Prvenstveno je napravljena baza MySQL baza podataka pod nazivom „diplomski“, te je u .env datoteci postavljen taj naziv baze podataka. Potom su pomoću naredbe *php artisan make:model PostThread –m –c –v* napravljeni prethodno spomenuti model, kontroler, pogled i migracija. U migraciji za bazu podataka su definirane kolone koje želimo imati u tablici i spremati u bazu podataka kako je prikazano na slici 3.16. Dakle, svaki PostThread predstavlja određeni fakultet, kolona unutar tablice koja definira ime tog fakulteta je nazvana „faculty“.

```php
class CreatePostThreadsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('post_threads', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('faculty');
            $table->timestamps();
            $table->unsignedBigInteger('user_id');
        });
    }
```

Slika 3.16. *-Kreiranje tablice u bazi podataka*

U modelu su definirana polja koja se moraju popuniti, te je definirano da je određeni PostThread napravljen od strane određenog korisnika. Između PostThread-a i određenog Post-a definirana je polimorfna (polymorphic) veza. Takva veza dozvoljava jednom modelu da pripada više od jednom tipu modela koristeći jednu riječ kao asocijaciju (u ovom slučaju ta riječ je „postable“). PostThread model je prikazan na slici 3.17.

```php
class PostThread extends Model
{

    protected $fillable = [
        'faculty',
    ];


    //
    public function user(){
        return $this->belongsTo(User::class);
    }

    public function posts(){
        return $this->morphMany(Post::class, 'postable')->latest();
    }
```

Slika 3.17. *-PostThread model*

Dakle, PostThread može imati više vrsta Postova, dok je jedan Post (objava) vezana samo za jedan PostThread tj. za jedan fakultet što je definirano u Post modelu kao na slici 3.18.

```
public function postable(){
    return $this->morphTo;
}
```

Slika 3.18. *-Dio Post modela*

U kontroleru su definirane funkcije *store* za spremanje thread-ova u bazu, *index* za prikaz tih thread-ova (prikaz liste fakulteta), te *show* za prikaz objava (Post-ova) koji pripadaju određenom thread-u. Funkcija za spremanje sadržaja u bazu se sastoji od validacije (maximalna dužina umetnutog teksta može biti 1500 znakova), te od metode za kreiranje thread-a koji pretstavlja određeni fakultet i spremanje datoteke. Dohvaća se vrijednost koju smo upisali u pogledu kao 'faculty' tj. kao ime fakulteta, te se sprema u „faculty“ polje koje je kreirano u migraciji za bazu podataka.

```
public function store(Request $request)
{
    //validate
    $this->validate($request, [
        'faculty' => 'required|max:1500',
    ]);

    //store
    $request->user()->postthreads()->create([
        'faculty' => $request->faculty,

    ]);

    return back();
}
```

Slika 3.19. *-Store funkcija u PostThreadController-u*

Funkcija za prikaz objava pojedinog fakulteta nalazi se na slici 3.20. Unutar nje se prema $id-u nalazi određeni thread, te se pridodaje objektu koji će se koristiti u pogledu za dohvaćanje tog thread-a.

```php
public function show($id)
{
    $postthread =PostThread::find($id);

    return view('postthreadsingle',['postthread'=> $postthread]);
}
```

Slika 3.20. *-Show funkcija u PostThreadController-u*

Te na kraju što se tiče thread-ova za postove, kako bi sve funkcioniralo potrebno je definirati rute kao na slici 3.21. Definirane su 3 rute koje će biti pozvane pritiskom na različite gumbove. *Postthread* ruta koristi PostController i *index* funkciju unutar kontrolera te služi za prikaz svih fakulteta. Sljedeća služi da spremanje fakulteta kao PostThread-ova . Te zadnja služi za prikaz svih objava koje pripadaju određenom thread-u tj. fakultetu, ona će npr. biti pozvana kada korisnik klikne na neki od fakulteta koji je ponuđen na listi. Dok će ruta za spremanje biti pozvana ukoliko korisnik unese naziv nekog novog fakulteta i klikne da se spremi, dok će prva ruta pod nazivom postthread biti pozvana pri samom učitavanju stranice tj. klikom na Post na navigacijskoj traci.

```php
//post thread
Route::get('/postthread', 'PostThreadController@index')->name('postthread');
Route::post('/postthread', 'PostThreadController@store');
Route::get('/postthreadsingle/{postthread_id}', 'PostThreadController@show')->name('postsingle');
```

Slika 3.21. *-Rute za PostThread*

Izgled i akcije koje se dešavaju klikom na gumbove definirani su u postthread.blade.php pogledu. U pogledu je bitno uključiti csrf_field() za zaštitu, on verificira da je autentificirani korisnik taj korisnik koji pokušava napraviti neki zahtjev u aplikaciji, u ovom slučaju da je korisnik koji pokušava dodati novi fakultet na listu autentificiran. Bitno je da se ime <textarea> polja polja slaže s nazivom koji je korišten u kontroleru i modelu. Izgled forme definiran je kao na slici 3.22. Dok je na slici 3.23. prikazan kod za prikaz liste fakulteta gdje je vidljivo da se pritiskom na određeni fakultet poziva ruta *postsingle* koja je prethodno spomenuta.

```
<form action="{{url('postthread')}}" method="post" >
{{ csrf_field() }}
    <div class="form-group">
        <textarea class="form-control" name="faculty" id="faculty-content" rows="1"
        placeholder="Name of the faculty"></textarea>

    </div>
    <button type="submit" class="btn btn-primary">Add</button>
    <input type="hidden" value="{{ Session::token() }}" name="token">
</form>
```

Slika 3.22. *-Forma za dodavanje novih fakulteta*

```
@foreach($postthreads as $postthread)

<a href="{{route('postsingle',$postthread->id)}}" class="list-group-item" >
    <h5 class="list-group-item-heading">{{$postthread->faculty}} </h4>
</a>

@endforeach
```

Slika 3.23. *-Prikaz liste fakulteta*

Kako bi se objave mogle spremati u bazu, prikazivati na stranici i uređivati potrebno je definirati Post model, kontroler pod nazivom PostController, politika ili PostPolicy, definirati rute, migracije u bazu, te u pogledu postthreadsingle.blade.php definirati potrebne akcije.

Na identičan način i za objave je napravljena migracija za bazu podataka, no u ovoj tablici su dva dodatna parametra, tj. dvije dodatne kolone, to su *postable_id* i *postable_type*. *Postable_id* predstavlja *id* od PostThread-a kojem pripada objava, dok *postable_type* predstavlja ime modela kojem objava pripada.

```
Schema::create('posts', function (Blueprint $table) {
    $table->bigIncrements('id');
    $table->timestamps();
    $table->text('body');
    $table->string('file');
    $table->integer('postable_id');
    $table->string('postable_type');
    $table->integer('user_id');
});
```

Slika 3.24. *- Kreiranje baze podataka*

U modelu su definirana polja koja se moraju popuniti, te je definirano da određena objava (engl. post) pripada određenom korisniku (engl. user), također kako je već ranije napomenutno definirana je *postable* funkcija koja omogućuje da se Post model poveže s više modela tj. da Post model može pripadati više od jednom modelu. Post model je prikazan na slici 3.25.

```php
class Post extends Model
{
    //
    protected $fillable = [
        'body',
        'file',

    ];

    protected $casts = [

        'user_id' => 'int',
    ];

    public function user(){
        return $this->belongsTo(User::class);
    }
    public function postable(){
        return $this->morphTo;
    }
```

Slika 3.25. *- Post model*

Objava pripada samo jednom korisniku, ali korisnik može imati više objava što je potrebno definirati u User modelu na način na koji je prikazano na slici 3.26.

```php
public function posts(){
    return $this->hasMany(Post::class);
}
```

Slika 3.26. *- Dio User model-a*

U kontroleru su definirane funkcije *addPostThread* za spremanje objava koje pripadaju određenom thread-u u bazu, *downloadfunction* za preuzimanje datoteka koje su drugi uploadali u svojim objavama, *destroy* za brisanje vlastitih objava, te *edit* i *update* za uređivanje i spremanje uređenog sadržaja u bazu. Funkcija za spremanje sadržaja u bazu se sastoji od validacije (maximalna dužina umetnutog teksta može biti 1500 znakova), te od metode za kreiranje dijelova

objave i spremanje datoteke. Dohvaća se vrijednost koju smo upisali u pogledu kao 'body' tj. kao sadržaj objave, te se sprema u body polje koje je kreirano u migraciji za bazu podataka. Provjerava da li zahtjev sadrži datoteku, ako da onda dohvaća njeno originalno ime i pod tim imenom ju sprema u public/files folder.

```
public function addThreadPost(Request $request, PostThread $postthread)
{

    $this->validate($request, [
        'body' => 'required|max:1500',
    ]);
   $post = new Post();
    $post->body = $request->body;
    $post->file = $request->file->getClientOriginalName();
    $post->user_id = auth()->user()->id;
    $postthread->posts()->save($post);

    //store file
    if( $request->has('file')) {
      $filename=  $request->file('file')->getClientOriginalName();
       $request->file->storeAs('public/files', $filename);
    }

    return back();
}
```

Slika 3.27. *-addThreadPost funkcija u PostController-u*

Unutar funkcije za brisanje objava prvo se provjerava da li je korisnik autoriziran tj. da li ima pravo na brisanje objave, te ukoliko ima navedena objava se briše kako je prikazano na slici 3.28.

```
public function destroy(Request $request, Post $post)
{
    $this->authorize('destroy', $post);

    $post->delete();

    return back();
}
```

Slika 3.28. *- Destroy funkcija u PostController-u*

Za uređivanje objava potrebno je definirati dvije funkcije, to su *edit* za uređivanje i *update* za spremanje uređenog sadržaja. *Edit* funkcija vrši autorizaciju provjerom *checkowner* funkcije koja

je definirana u PostPolicy tj. provjerava se može li korisnik izvršiti željenu radnju, te otvara novi pogled za uređivanje objave, te nakon završetka nas vraća na početni pogled.

```php
public function edit(Post $post)
{
    $this->authorize('checkowner', $post);
    return view('edit', [
        'post' => $post,
    ]);
}

public function update(Request $request, Post $post)
{
    $this->authorize('checkowner', $post);
    $post->update($request->all());
    return redirect('/postthread');
}
```

Slika 3.29. *- Edit i Update funkcije PostController-a*

Kako bi ispravno radila funkcija za pruzimanje datoteke u PostControlleru je prvenstveno je potrebno provjeriti da li datoteka postoji u mapi, te ukoliko postoji pokreće se preuzimanje.

```php
public function downloadfunction($filename= '')
{

    // Check if file exists in app/storage/file folder
    $file_path = storage_path() . "storage/app/public/files" . $filename;
    $headers = array(
        'Content-Type: csv',
        'Content-Disposition: attachment; filename='.$filename,
    );
    if ( file_exists( $file_path ) ) {
        // Send Download
        return \Response::download( $file_path, $filename, $headers );
    } else {
        // Error
        exit( 'Requested file does not exist on our server!' );
    }
}
```

Slika 3.30. *-Funkcija za preuzimanje datoteke*

Kako bi radnje brisanja i uređivanja objava bile moguće potrebna je PostPolicy i potrebno ju je dodati u provider *AuthServiceProvider*. Politika (policy) nam pomaže u zaštiti od neovlaštenog pristupa. Dakle, omogućuju da samo korisnici koji su stvorili objavu istu mogu uređivati i brisati.

```php
public function destroy(User $user, Post $post)
{
    return $user->id === $post->user_id;
}
public function checkowner(User $user, Post $post)
{
    return $user->id === $post->user_id;
}
```

Slika 3.31. *– PostPolicy*

Te na kraju, kako bi sve funkcioniralo potrebno je definirati rute kao na slici 3.32. Definirano je 5 ruta koje će biti pozvane pritiskom na različite gumbove. Ruta koja se koristi za spremanje objave koja pripada točno određenom fakultetu je nazvana *threadpost.store*. Sljedeća ruta služi za brisanje objava i koristi također PostController i funkciju destroy. Te sljedeće dvije služe za uređivanje i update objava te koriste PostController i funkcije edit i update. Dok zadnja pod nazivom *downloadfile* služi za preuzimanje datoteka.

```php
//posts
Route::post('post/create/{postthread}','PostController@addThreadPost')->name('threadpost.store');
Route::delete('/post/{post}', 'PostController@destroy');
Route::get('/edit/{post}', 'PostController@edit');
Route::patch('/post/{post}', 'PostController@update');
//download file from post
Route::get('downloadfile','PostController@downloadfunction')->name('downloadfile');
```

Slika 3.32. *–Rute za objave*

Izgled i akcije koje se dešavaju klikom na gumbove definirani su u postthreadsingle.blade.php pogledu. U pogledu je bitno uključiti csrf_field() za zaštitu, on verificira da je autentificirani korisnik taj korisnik koji pokušava napraviti neki zahtjev u aplikaciji. Te je kao akciju u formi potrebno postaviti *{{route('threadpost.store, $postthread->id')}}* pri čemu se poziva ruta *threadpost.store* za spremanje objava točno određenog thread-a, dok *enctype="multiport/form-data"* služi za upload datoteka u formi. Bitno je da se ime <textarea> polja i <input> polja slažu s nazivima koje smo koristili u kontroleru i modelu. Izgled forme definiran je kao na slici 3.33.

```
<form action="{{route('threadpost.store',$postthread->id)}}" method="post"
    enctype="multipart/form-data">
{{ csrf_field() }}
    <div class="form-group">
        <textarea class="form-control" name="body" id="post-content" rows="5" placeholder="Your post">
        </textarea>
        <input type="file" name="file" id="file" >
    </div>

    <button type="submit" class="btn btn-primary">Create post</button>
    <input type="hidden" value="{{ Session::token() }}" name="token">
</form>
```

Slika 3.33. *- Forma za kreiranje objava*

U pogledu su ispod forme za kreiranje objava prikazane objave drugih korisnika, na način da je prikazan sadržaj, link na dokument s imenom dokumenta, te ime korisnika i datum kada je postavio objavi.

```
<p> {{ $post->body }} </p>
<a href="{{route('downloadfile',$post->file)}}"  download="{{$post->file  }}">
{{$post->file}}
</a>
<div class="info">
    Posted by {{ $post->user->name }} on {{ $post->created_at }}
</div>
```

Slika 3.34. *- Objave drugih korisnika*

Ukoliko se radi o korisniku koji je vlasnik objave prikažu mu se i gumbovi za brisanje i uređivanje objave. Klikom na Edit gumb preusmjerava se na drugi pogled unutar kojeg mu se otvara samo forma unutar koje može unijeti željene izmjene. Da bi bilo moguće brisanje potrebno je osim *csrf_field()* uključiti i *{{ method_field('DELETE') }}*.

```
@if(Auth::user() == $post->user)
 <!-- Post Edit Button -->
<td>
      <form style="display: inline-block;" action="{{url('edit/' . $post->id)}}" method="GET">
        {{ csrf_field() }}
            <button type="submit"  class="btn btn-danger">
                <i class="fa fa-btn fa-edit"></i>Edit
            </button>
    </form>
</td>
  <!-- Post Delete Button -->
<td>
    <form action="{{url('post/' . $post->id)}}" method="POST" style="display: inline-block;">
        {{ csrf_field() }}
        {{ method_field('DELETE') }}

            <button type="submit" id="delete-post-{{ $post->id }}" class="btn btn-danger">
                <i class="fa fa-btn fa-trash"></i>Delete
            </button>
    </form>
</td>
@endif
```

Slika 3.35. *-Gumovi za brisanje i uređivanje objave*

3.2. Forum

Klikom na navigacijskoj traci na „Forum“ otvara se pogled na ovaj dio aplikacije. Kao i kod Post dijela aplikacije prvenstveno je potrebno da korisnik izabere koji je fakultet, i ovdje postoji opcija dodavanja fakulteta na popis ukoliko već nije na istom. Kao i kod thread-a namjenjenog za objave, i ovdje su pomoću naredbe *php artisan make:model Thread –m –c –v* definirani model pod nazivom Thread, kontroler pod nazivom ThreadController, migracija za bazu podataka i odgovarajući pogled unutar kojeg će biti definirane određene akcije.

Migracija za bazu podataka izgleda isto kao i za PostThread pri čemu ovaj put svaki Thread predstavlja određeni fakultet, a kolona „faculty“ ime tog fakulteta.

Budući da je ovaj dio aplikacije namjenjen za postavljanje pitanja i postavljanje odgovora na ta pitanja, to je realizirano na način da je omogućeno dodavanje komentara (pitanja) na određenom thread-u koji predstavlja određeni fakultet, te dodavanje komentara (odgovora) na ta pitanja. Da bi se to ostvarilo u modelu je između Thread-a i određenog Comment-a definirana polimorfna (polymorphic) veza koja dozvoljava jednom modelu da pripada više od jednom tipu modela (model Comment pripada modelu Thread i modelu Comment) koristeći jednu riječ kao asocijaciju (u ovom slučaju ta riječ je „comentable“). Thread model je prikazan na slici 3.36.

```php
class Thread extends Model
{
    use CommentableTrait;

    protected $fillable = [
        'faculty',
    ];


    //
    public function user(){
        return $this->belongsTo(User::class);
    }

}
```

Slika 3.36. *-Thread model*

Kao što vidimo na prethodnoj slici unutar Thread kontrolera koristi se CommentableTrait. Trait omogućava upotrebu istog koda više puta, a budući da je i Thread i Comment povezan s Comment modelom polimorfnom vezom jedan naprema više; dio koda koji to definira je ispisan u CommentableTrait te korišten i u Thread i u Comment modelu. Također, još jedna funkcija koja je definirana u trait-u je funkcija za dodavanje komentara (pitanja i odgovora) koja se također poziva u CommentContoller-u u funkcijama za dodavanje pitanja i odgovora . Trait je prikazan na slici 3.37.

```php
trait CommentableTrait
{

    public function comments(){
        return $this->morphMany(Comment::class, 'commentable')->latest();
    }

    public function addComment($body)
    {
        $comment=new Comment();
        $comment->body=$body;
        $comment->user_id=auth()->user()->id;

        $this->comments()->save($comment);

        return $comment;
    }

}
```

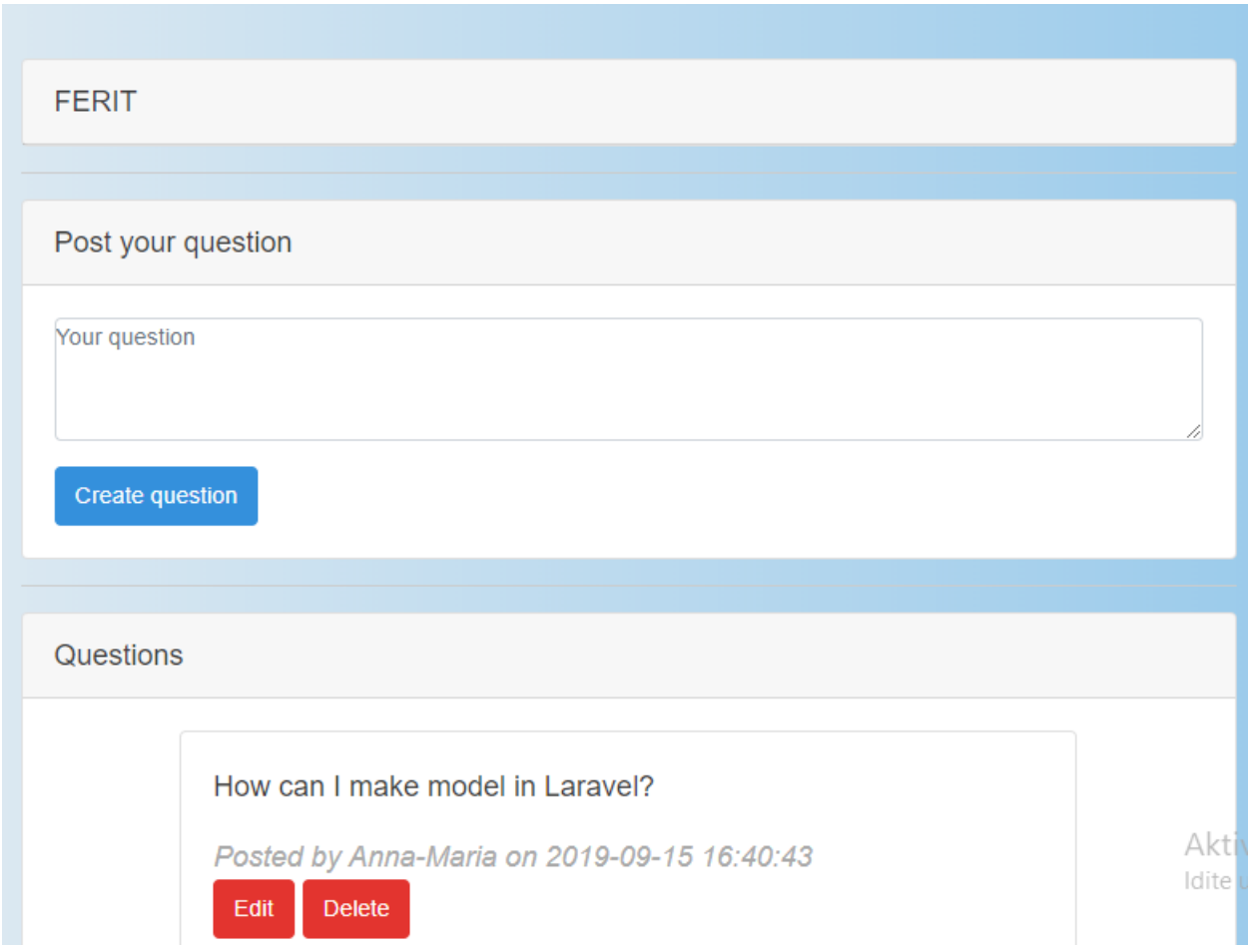
Slika 3.37. *–CommentableTrait*

Dakle, Thread i Comment mogu imati više vrsta komentara , dok je jedan komentar vezan za samo jedan Thread tj. za jedan fakultet ukoliko se radi o pitanju i za samo jedan Commentar (pitanje) ako se radi o odgovoru.

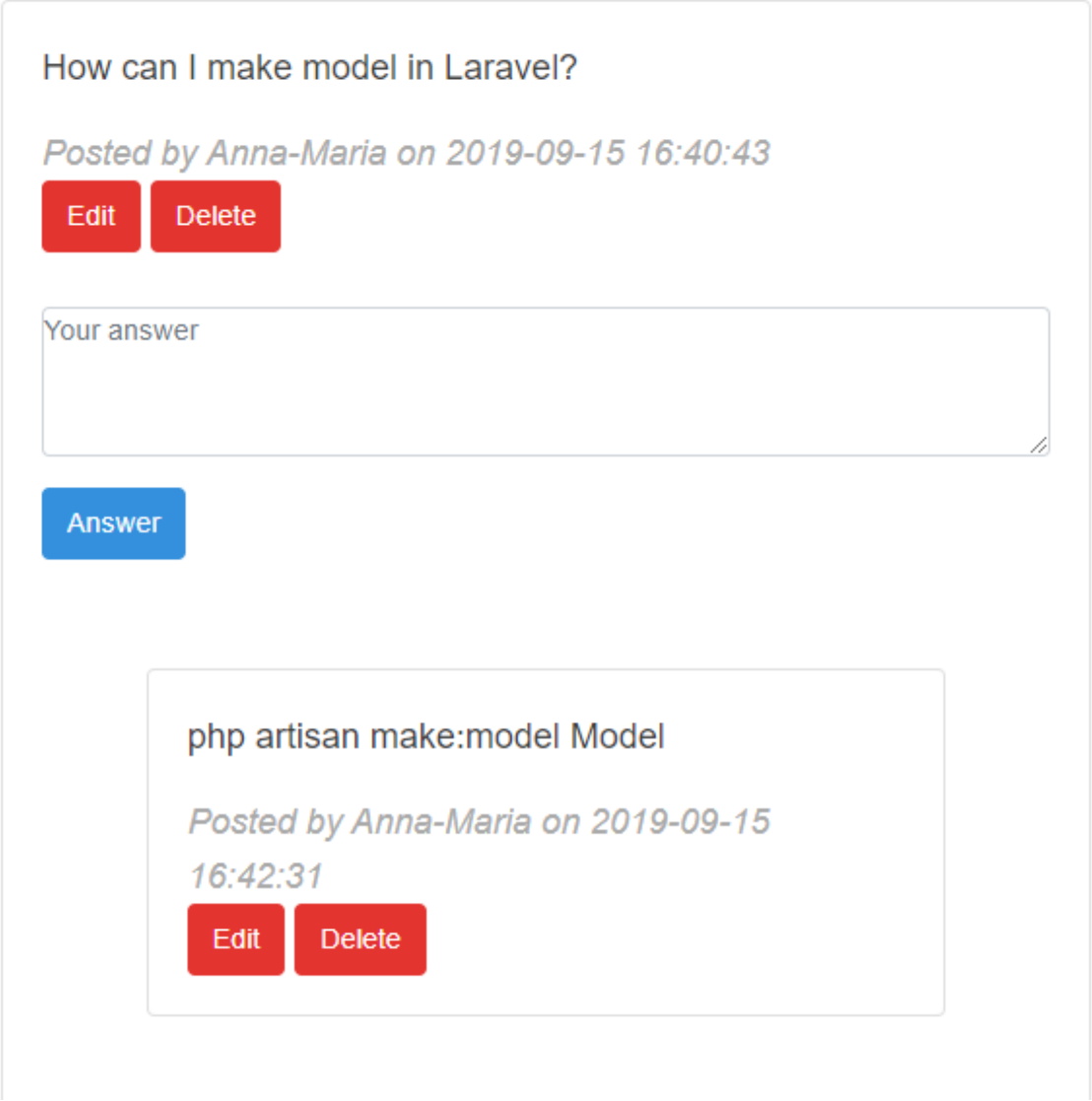
```
public function commentable(){
    return $this->morphTo;
}
```

Slika 3.38. *-Dio Comment modela*

Kontroleri, rute i pogled su identični kao kod PostThread modela. Postoje funkcije za spremanje i za prikaz pitanja i odgovora koji pripadaju određenom fakultetu koje su izvedene na isti način kao i kod PostThread-a. Klikom na neki od ponuđenih fakulteta poziva se funkcija *show* za prikaz pitanja i odgovora koji pripadaju tom fakultetu što izgleda kao na sljedećim slikama.



Slika 3.39. *- Forum-pitanje*



Slika 3.40. *-Forum-pitanje i odgovor*

Dakle, na vrhu stranice se prikazuje ime fakulteta koji je odabran, potom forma za postavljanje pitanja, te na kraju prikaz pitanja i forma za odgovor na isto uz popis ostalih odgovora. Kako bi dobili ovaj prikaz i ovu fukcionalnost kao na prethodnoj slici kao i kod objava (Post) potrebno je definirati model Comment, kontroler CommentController, politiku CommentPolicy, definirati rute, migracije u bazu, te u pogledu threadsingle.blade.php definirati potrebne akcije.

U Comment modelu se definira da jedan kometar pripada jednom korisniku, te se poziva Trait kako bi se definirala jedan prema više polimorfna veza (jedan kometar prema više komentara, pojednostavljeno jednom komentaru koji predstavlja pitanje pripada više kometara koji predstavljaju odgovor, dok jedan kometar koji predstavlja odgovor može pripadati samo jednom kometaru koji predstavlja pitanje) i naravno definira se funcija commentable koja omogućuje da

se navedene radnje ispune tj. da jedan model pripada više od jednom tipu modela, u ovom slučaju da jedan model pripada i samom sebi. Comment model je prikazan na slici 3.41.

```php
class Comment extends Model
{
    use CommentableTrait;

    protected $fillable = [
        'body',
    ];
    public function commentable(){
        return $this->morphTo;
    }
    public function user(){
        return $this->belongsTo(User::class);
    }

}
```

Slika 3.41. *-Comment model*

U CommentController-u su definirane funkcije *addThreadComment* koja služi za dodavanje pitanja unutar jednog thread-a i *addReplyComment* koja služi za dodavanje odgovora na postavljena pitanja, koje unutar svog tijela poziva već pomenutu *addComment* funkciju definiranu u CommentableTrait-u, te funkcije za brisanje i uređivanje pitanja (*destroy*, *edit*, *update*) i brisanje i uređivanje odgovora (*destroyreply*, *editreply*, *updatereply*)koje su izvedene na isti način kao kod PostContoller-a. Na slici 3.42. prikazane su funkcije za dodavanje pitanja i odgovora unutar kojih se obavlja prvenstveno validacija, potom se poziva funkcija definirana u Trait-u, te se definira da se nakon tih radnji vrati nazad na prethodni pogled.

```php
public function addThreadComment(Request $request, Thread $thread)
{

    $this->validate($request, [
        'body' => 'required|max:1500',
    ]);

   $thread->addComment($request->body);
    return back();
}
public function addReplyComment(Request $request, Comment $comment)
{

    $this->validate($request, [
        'body' => 'required|max:1500',
    ]);

   $comment->addComment($request->body);
    return back();
}
```

Slika 3.42. *-Dio CommentController-a*

U migraciji su definirane kolone *comentable_id* i *commentable_type* kao na slici 3.43. pri čemu *comentable_type* predstavlja kojem modelu pripada komentar (pri čemu njegova vrijednost u bazi podataka može biti App\Thread ili App\Comment), a *comentable_id* predstavlja *id* tog threada-a ili pitanja. Na slici 3.44. prikazano je navedeno u bazi podataka.

```php
Schema::create('comments', function (Blueprint $table) {
    $table->bigIncrements('id');
    $table->text('body');
    $table->integer('user_id');
    $table->integer('commentable_id');
    $table->string('commentable_type');
    $table->timestamps();
});
```

Slika 3.43. *-Migracija za comments tablicu*

id	body	user_id	commentable_id	commentable_type	created_at	updated_at
48	How can I make model in Laravel?	1	1	App\Thread	2019-09-15 16:40:43	2019-09-15 16:40:43
49	php artisan make:model Model	1	48	App\Comment	2019-09-15 16:42:31	2019-09-15 16:42:31
50	What is MVC?	1	1	App\Thread	2019-09-17 10:01:55	2019-09-17 10:01:55
51	What is polymorphic relation?	1	1	App\Thread	2019-09-17 10:02:44	2019-09-17 10:02:44
52	It stands for Model-View-Controller	1	50	App\Comment	2019-09-17 10:03:10	2019-09-17 10:03:10
53	Model–View–Controller (usually known as MVC) is a ...	1	50	App\Comment	2019-09-17 10:04:26	2019-09-17 10:04:26
54	A polymorphic relationship allows the target model...	1	51	App\Comment	2019-09-17 10:05:02	2019-09-17 10:05:02

Slika 3.44. *-Prikaz funkcionalnosti polimorfne veze u bazi podataka*

Sljedeći korak su rute, kao i kod objava, tu su rute za dodavanje pitanja i odgovora, te za njihovo uređivanje, update i brisanje. Navedeno je prikazano na slici 3.45.

```
//comment
Route::resource('comment','CommentController');
Route::post('comment/create/{thread}','CommentController@addThreadComment')->name('threadcomment.store');
Route::get('/editcomment/{comment}', 'CommentController@edit');
Route::patch('/comment/{comment}', 'CommentController@update');
Route::delete('/comment/{comment}', 'CommentController@destroy');
//reply
Route::post('reply/create/{comment}','CommentController@addReplyComment')->name('replycomment.store');
Route::get('/editreply/{comment}', 'CommentController@editreply');
Route::patch('/reply/{comment}', 'CommentController@updatereply');
Route::delete('/reply/{comment}', 'CommentController@destroyreply');
```

Slika 3.45. *–Rute za pitanja i odgovore*

U pogledu threadsingle.blade.php definirani su dio gdje se prikaže ime thread-a na koji je korisnik kliknuo tj. ime fakulteta koji je korisnik izabrao, definiran izgled dijela za unos pitanja. Potom su prikazana sva pitanja pri čemu je unutar svakog omogućeno dodavanje odgovora i prikaz pripadajućih odgovora. Pri čemu klikom na određene gumove se pozivaju pripadajuće rute i obavlja određena radnja kao npr. dodavanje odgovora pri čemu se poziva threadcomment.store ruta koja ukazuje na korištenje *addThreadComment* funkcije unutar kontrolera. Na sljedećim slikama prikazan je dio u blade-u za prikaz komentara i za prikaz odgovora.

```
@foreach($thread->comments as $comment)
<!-- display each question in separate card body -->
<div class="row justify-content-center ">
<div class="card w-75 ">
<div class="card-body">

<article class="post">
        <p> {{ $comment->body }} </p>
        <div class="info">
            Posted by {{ $comment->user->name }} on {{ $comment->created_at }}
        </div>
```

Slika 3.46. *-Prikaz komentara –pitanja*

```
<!-- Answer-->
<form action="{{route('replycomment.store', $comment->id)}}" method="post" >
    {{ csrf_field() }}
    <div class="form-group">
        <textarea class="form-control" name="body" id="answer-content" rows="3"
        placeholder="Your answer"></textarea>

    </div>

    <button type="submit" class="btn btn-primary">Answer</button>
    <input type="hidden" value="{{ Session::token() }}" name="token">
</form>
```

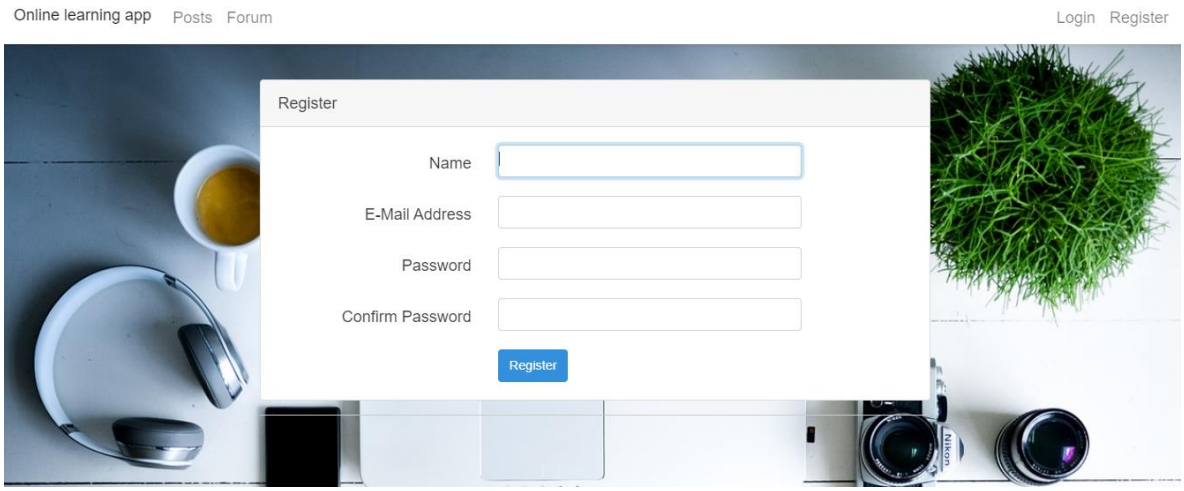
Slika 3.47. *-Dodavanje odgovora*

```
@foreach($comment->comments as $reply)

<!-- card inside card inside card -->
<div class="row justify-content-center">
<div class="card w-75">
<div class="card-body">

    <article class="post reply-list">
        <p> {{ $reply->body }} </p>
        <div class="info">
            Posted by {{ $reply->user->name }} on {{ $reply->created_at }}
        </div>
```

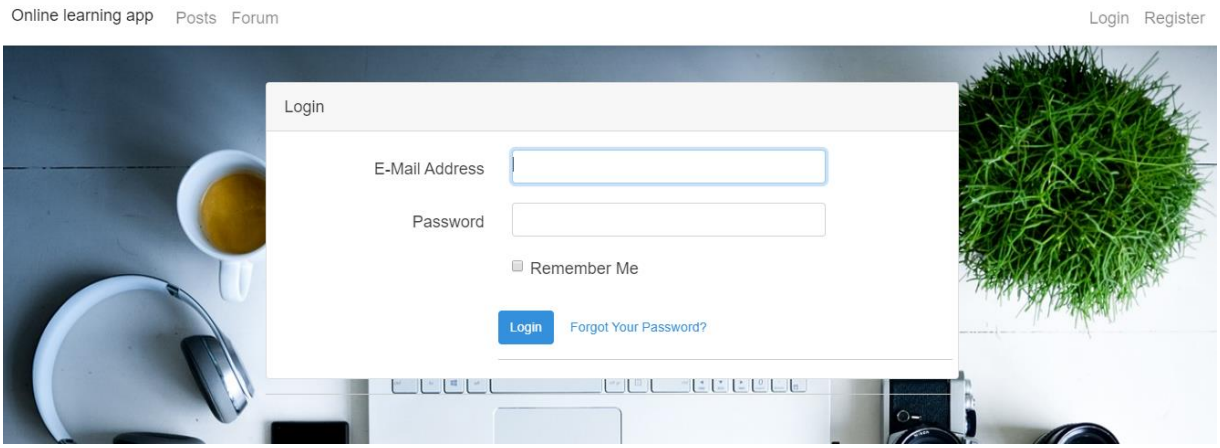
Slika 3.48. *-Prikaz odgovora*

4. TESTIRANJE I PRIKAZ FUNKCIONALNOSTI APLIKACIJE

Kako je prethodno već navedeno prvi korak pri korištenju aplikacije je prijava ili registracija kao što je prikazano na slikama 4.1. i 4.2.



Slika 4.1. *- Registracija korisnika*

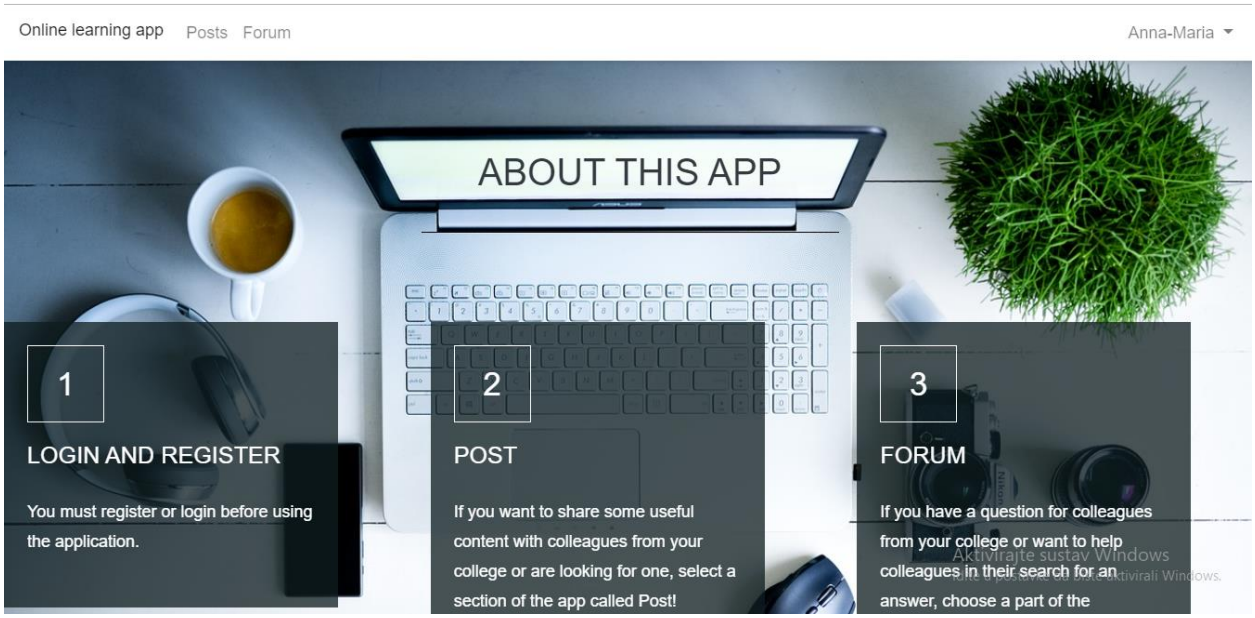


Slika 4.2. *-Prijava korisnika*

Također, prijavljeni korisnik se može odjaviti. Te može klikom na navigacijskoj traci ići iz dijela za objave u forum i obratno kako je prikazano na slici 4.3.

Slika 4.3. *- Odjava korisnika*

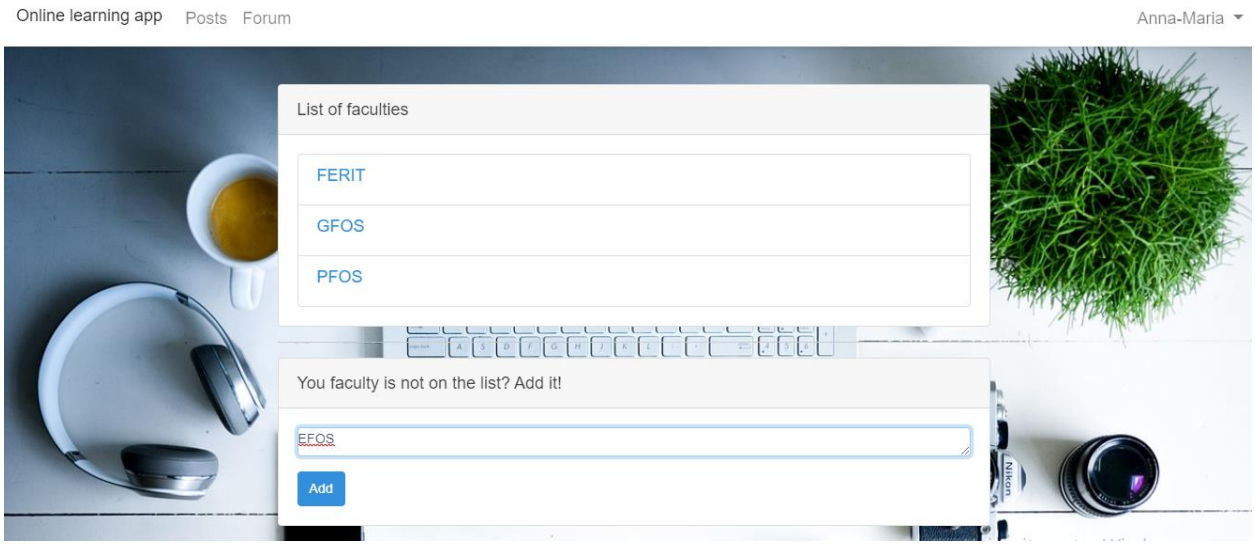
Prvenstveno nakon prijave ili registracije se otvara stranica dobrodošlice koju mogu vidjeti i korisnici koji nisu prijavljeni kao na sljedećoj slici.



Slika 4.4. *-Početna stranica*

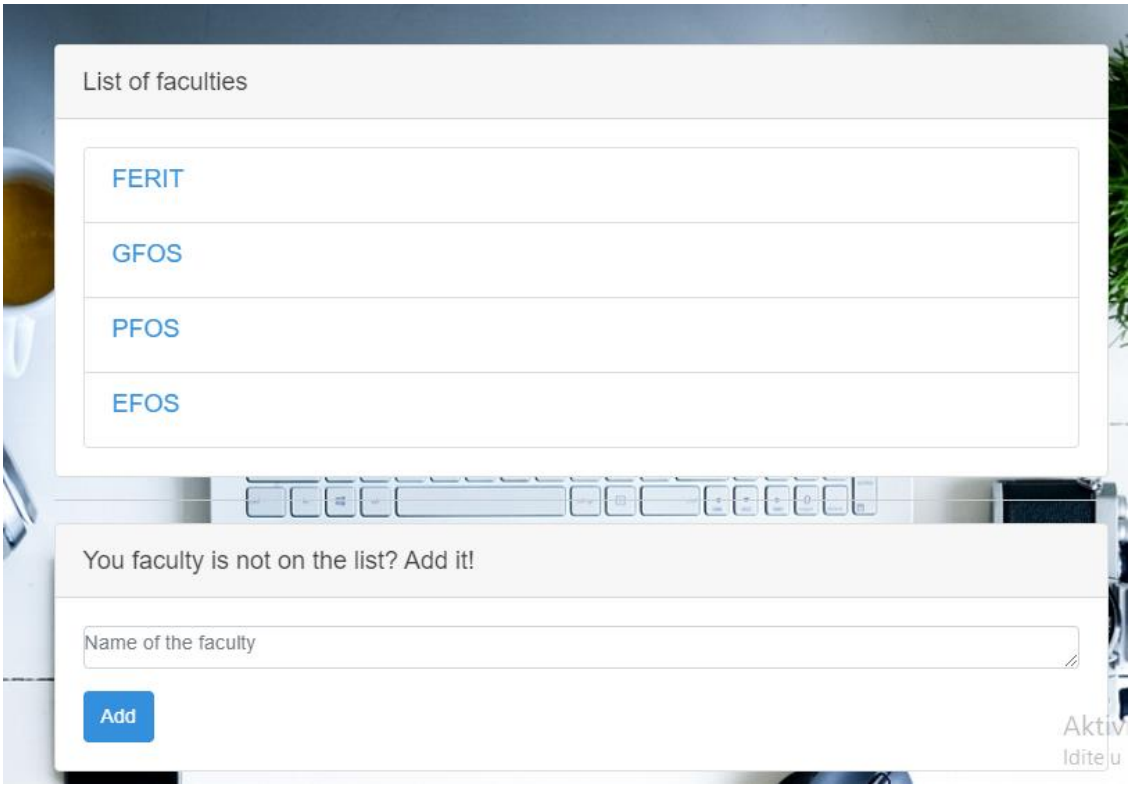
4.1. Prikaz funkcionalnosti objava

Prilikom klika na Post na navigacijskoj traci prvenstveno se otvara stranica s popisom fakltеta i mogućnošću dodavanja novog fakulteta ukoliko korisnikom fakultet nije na popisu što je prikazano na slici 4.5.



Slika 4.5. *- Post-popis fakulteta*

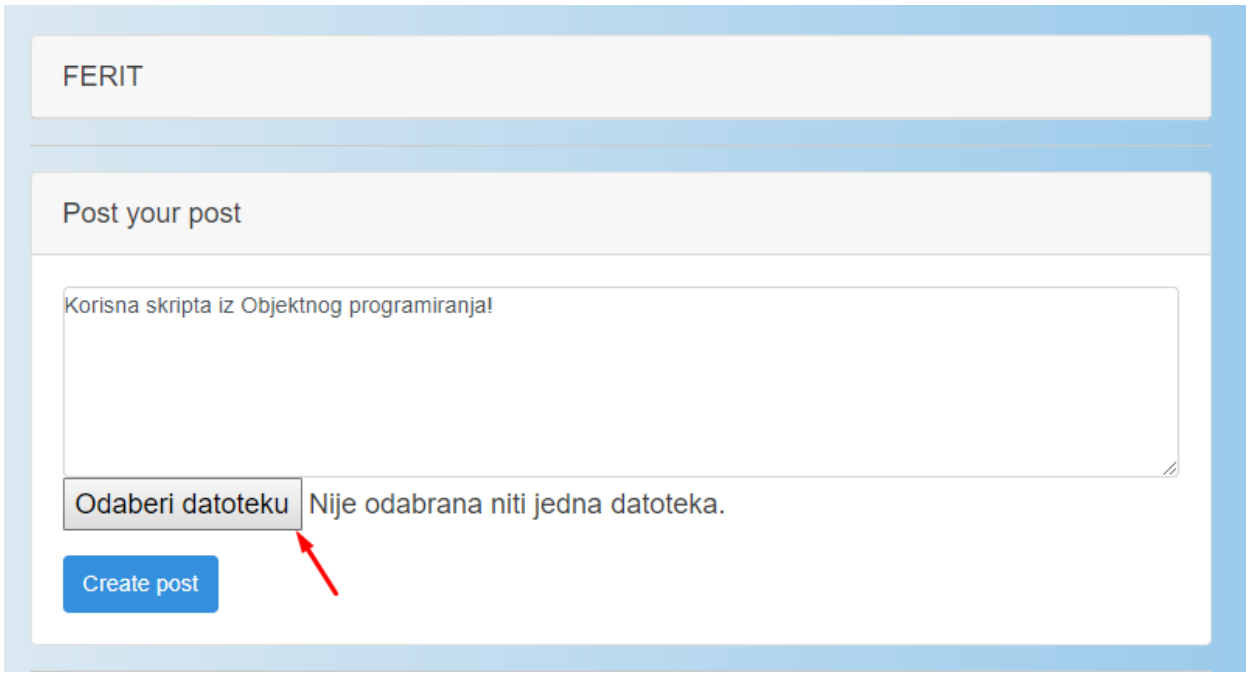
Klikom na Add gumb fakultet pod nazivom EFOS se dodaje na popis te se može odabrati i postavljati pitanje unutar njegovog thread-a.



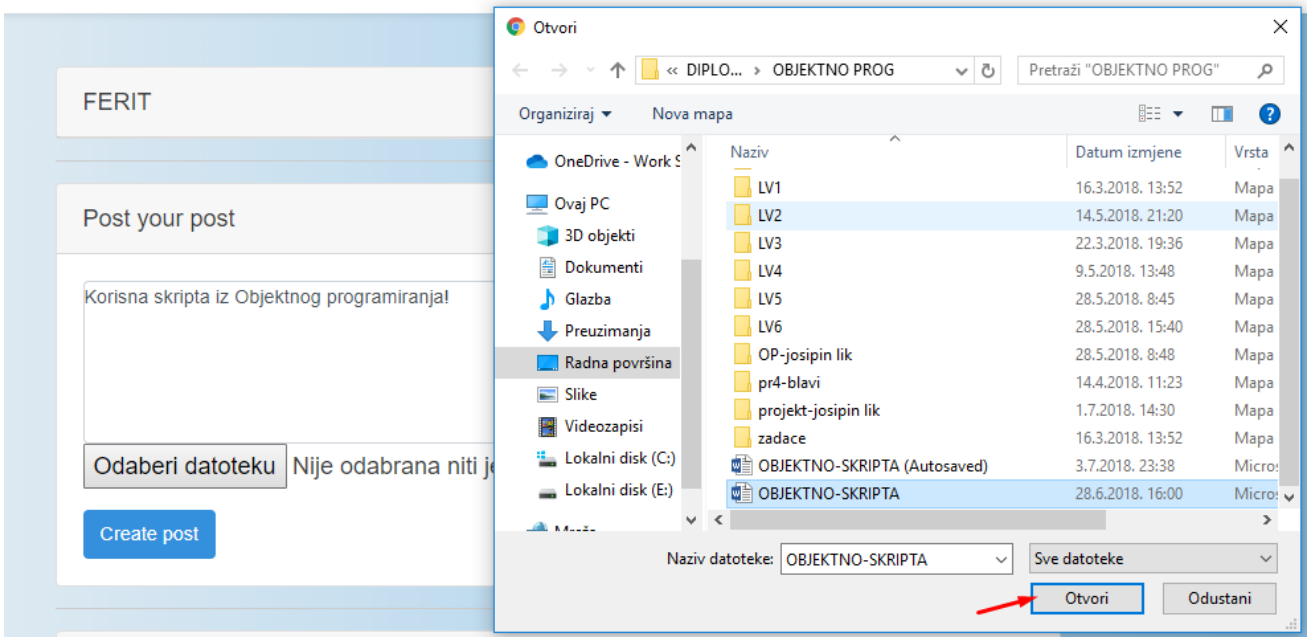
Slika 4.6. *-Fakultet dodan na popis*

Klikom na neki od ponuđenih fakulteta otvara se stranica gdje je prikazano ime tog fakulteta, moguće kreiranje objave uz upload datoteke. Te se na kraju te stranice prikažu objave od svih korisnika koji pripadaju tom fakultetu s mogućnošću preuzimanja datoteka.

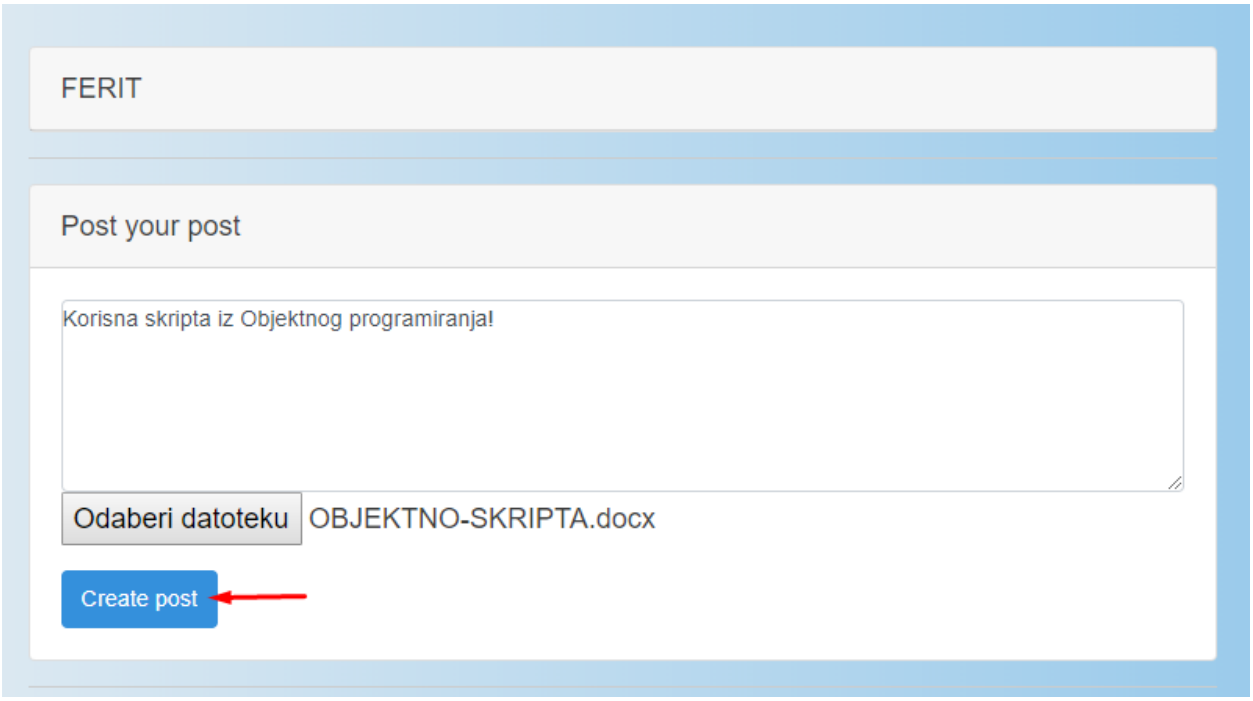
Na sljedećim slikama prikazan je način kreiranja objave i uploada datoteke. Kako bi učitali datoteku ,naravno, prvenstveno je klikom na „Odaberi datoteku“ potrebno otvoriti prozor za odabir datoteke s korisnikovog računala. Nakon što je datoteka odabrana i sadržaj objave napisan klikom na „Crete post“ objava se sprema u bazu podataka i prikazuje na istoj stranici gdje je i kreirana. Odabrana datoteka sprema se u *storage/app/public/files* direktorij unutar projekta.



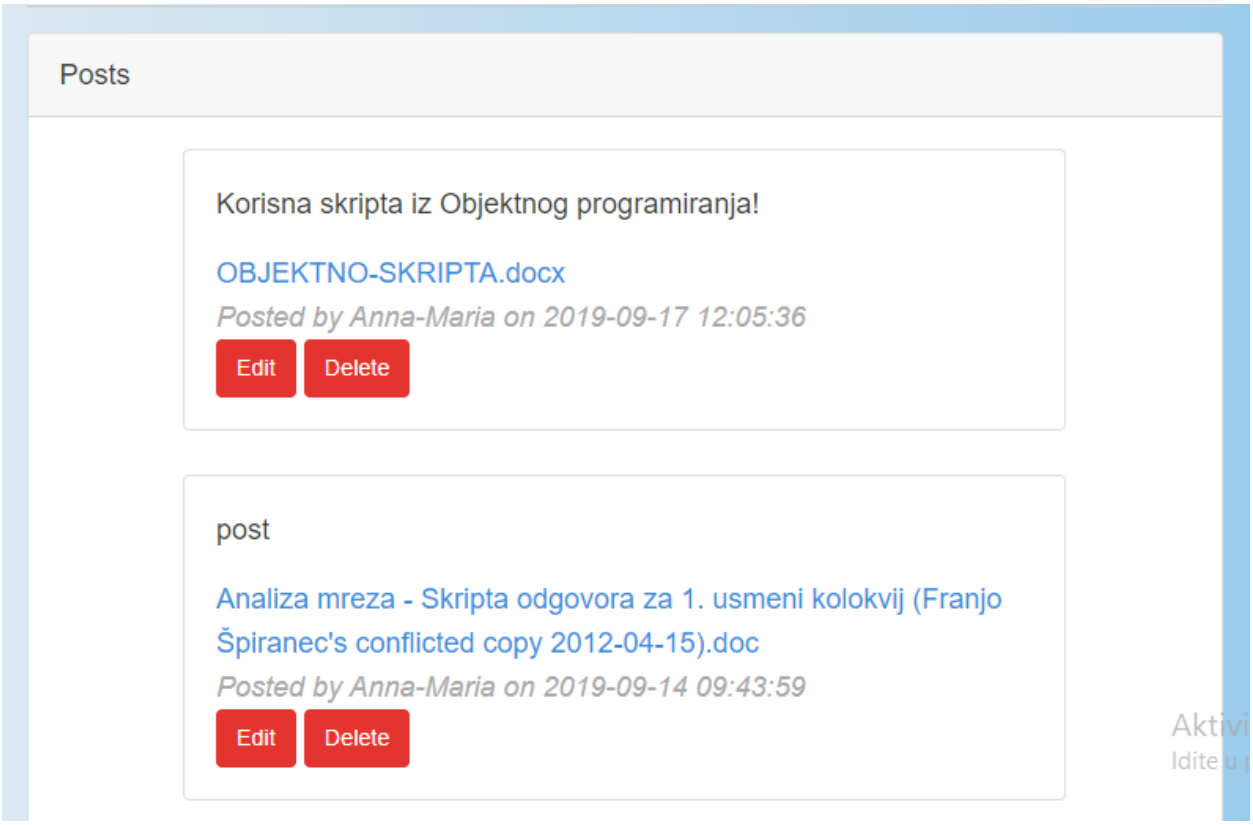
Slika 4.7. *- Odabir datoteke*



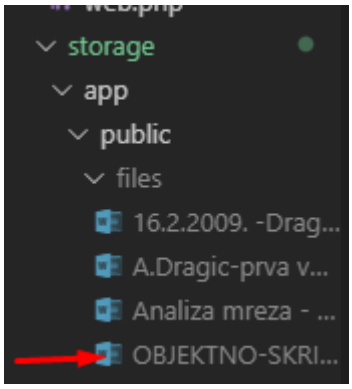
Slika 4.8. *- Odabir datoteke*



***Slika 4.9**. - Kreiranje objave*



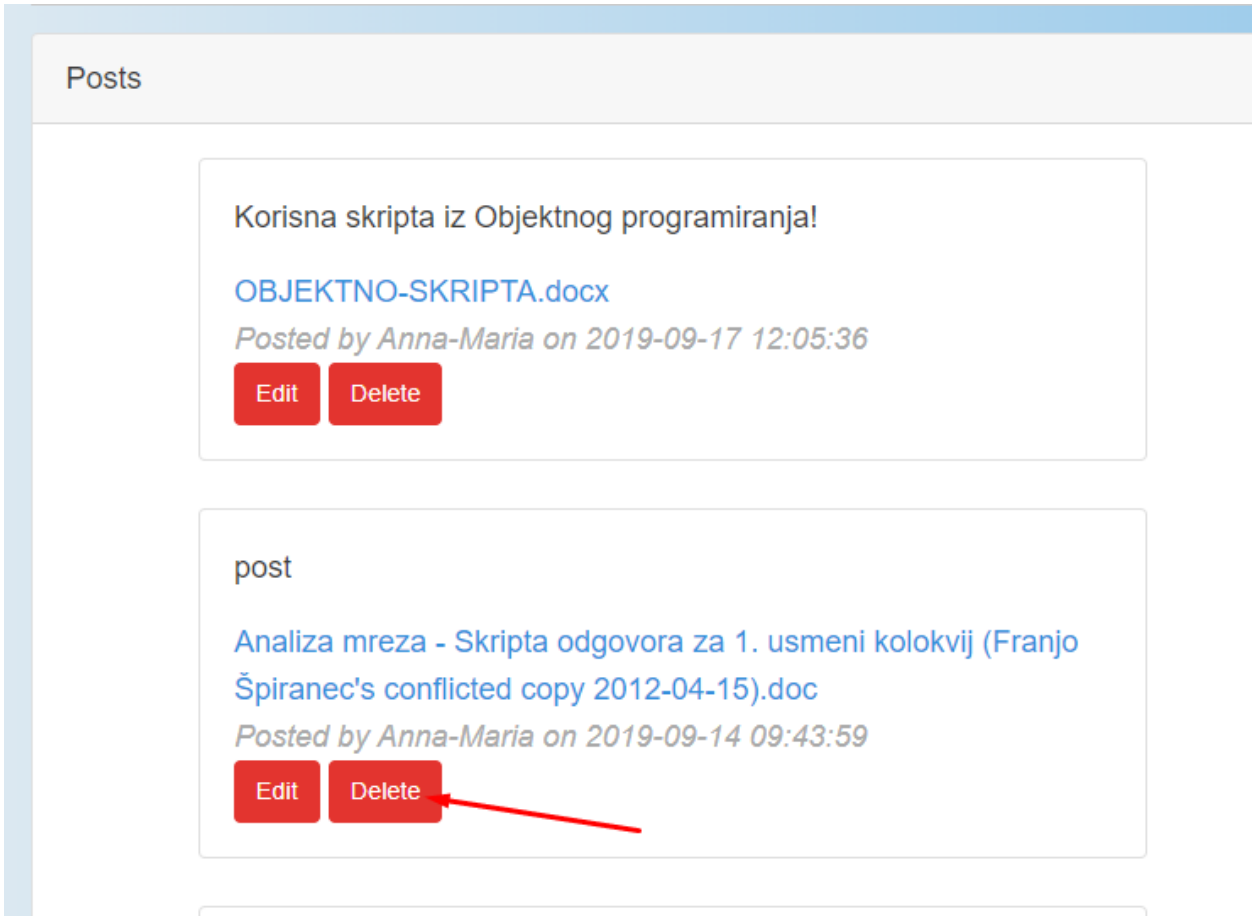
***Slika 4.10.** - Prikaz kreirane objave*



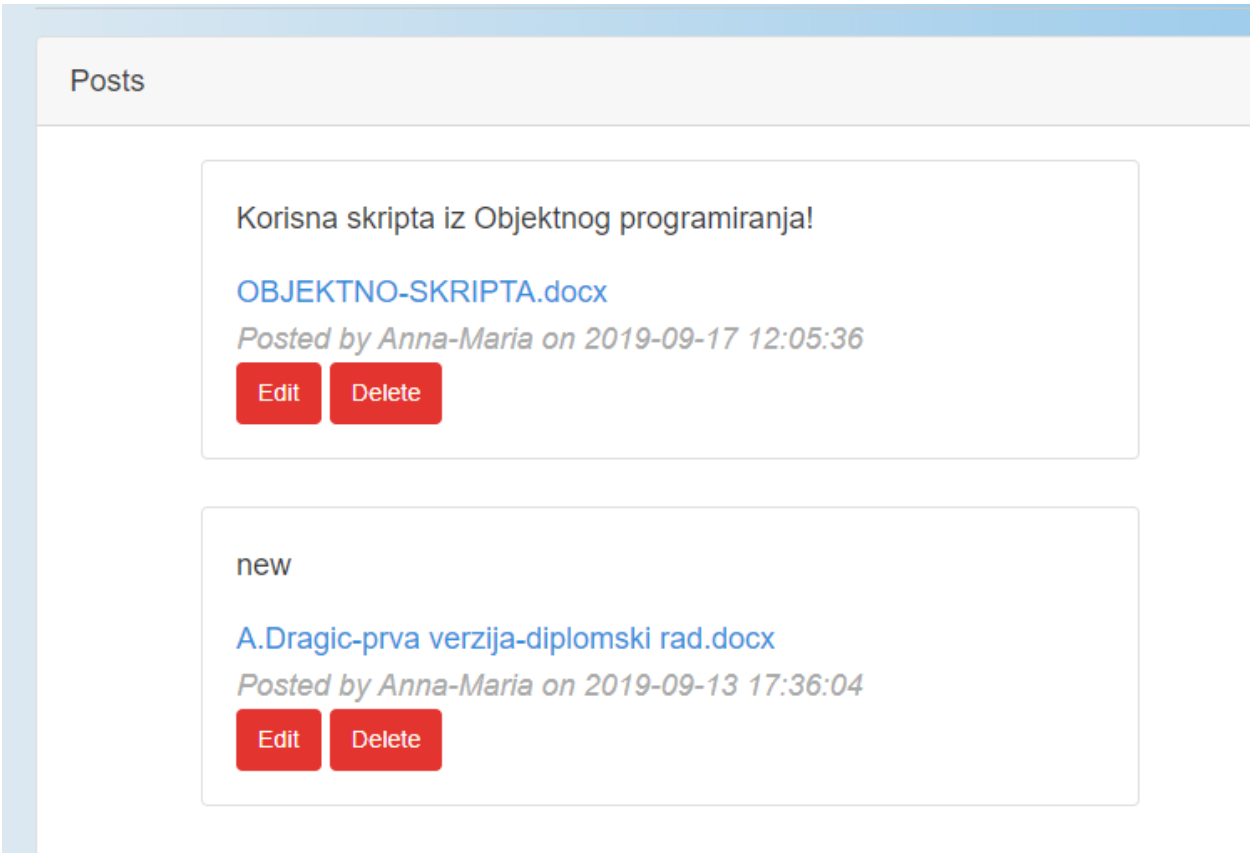
Slika 4.11. *- Datoteka spremljena*

4.1.1. Uređivanje i brisanje objava

Omogućeno je uređivanje i brisanje vlastitih objava klikom na gumbe Edit ili Delete. Klikom na Delete se jednostavno željena objava izbriše i sa stranice a i iz baze.

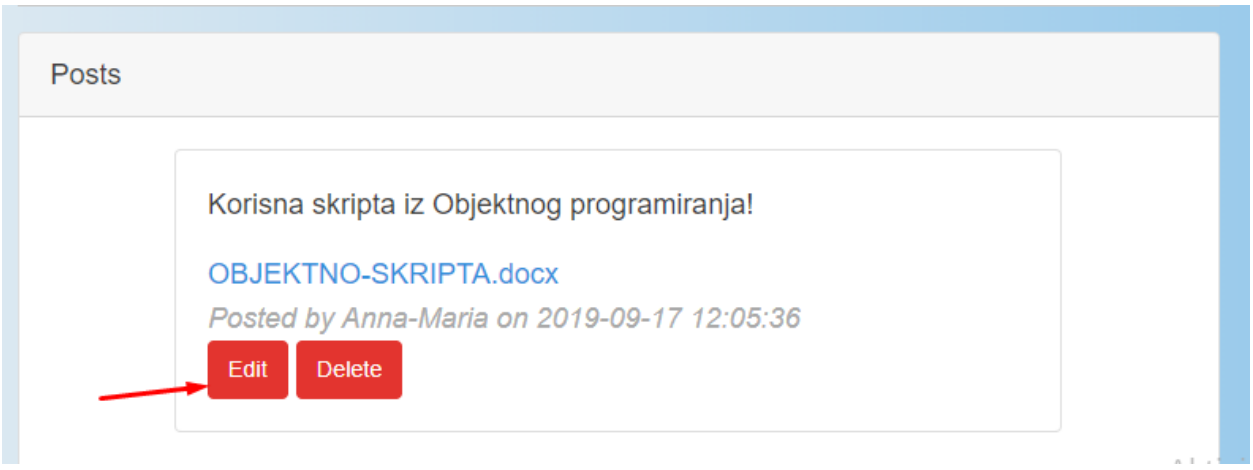


Slika 4.12. *- Brisanje objave*

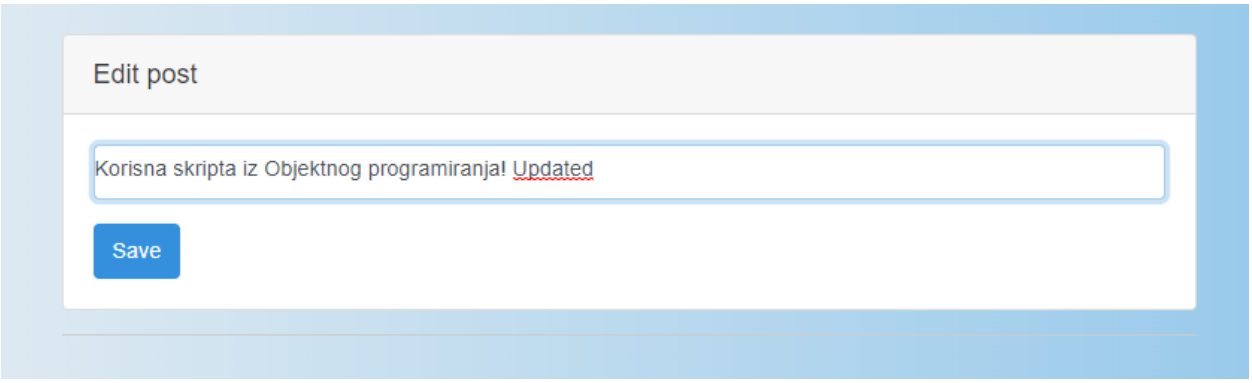


***Slika 4.13**. - Objava je obrisana*

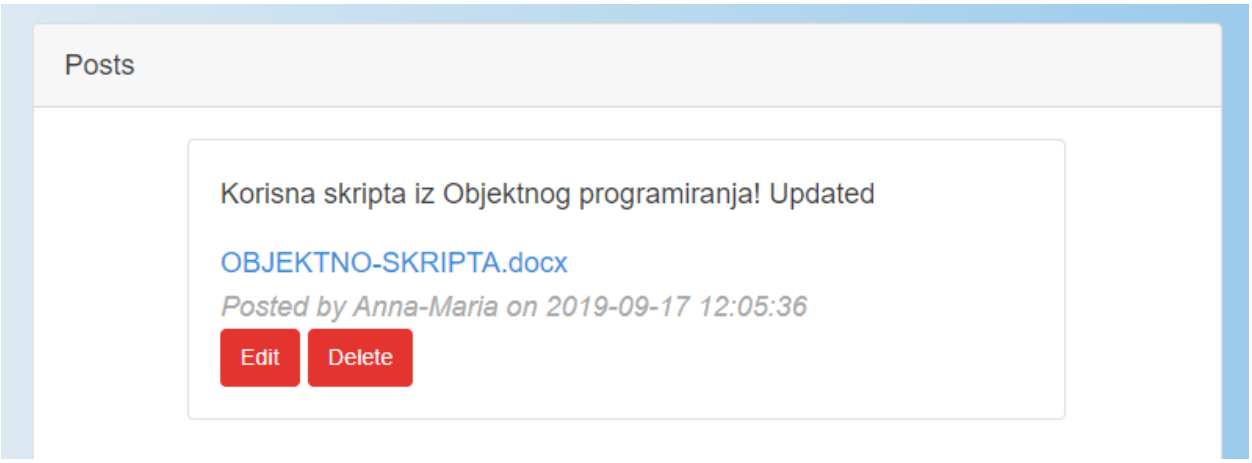
Klikom na gum Edit otvara se nova stranica ili novi pogled gdje korisnik uređuje tekst te ga sprema u bazu i prikazuje na prethodnoj stranici. Navedene radnje su prikazane na sljedećim slikama.



***Slika 4.14.** - Uređivanje objave*



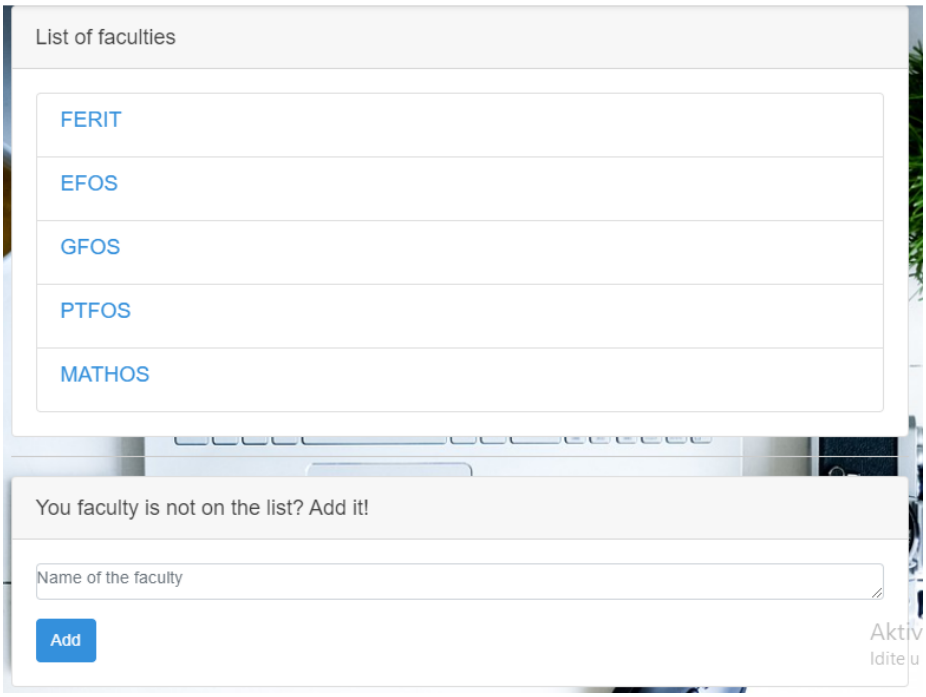
Slika 4.15. *- Uređivanje teksta objave*



Slika 4.16*. - Objava uređena*

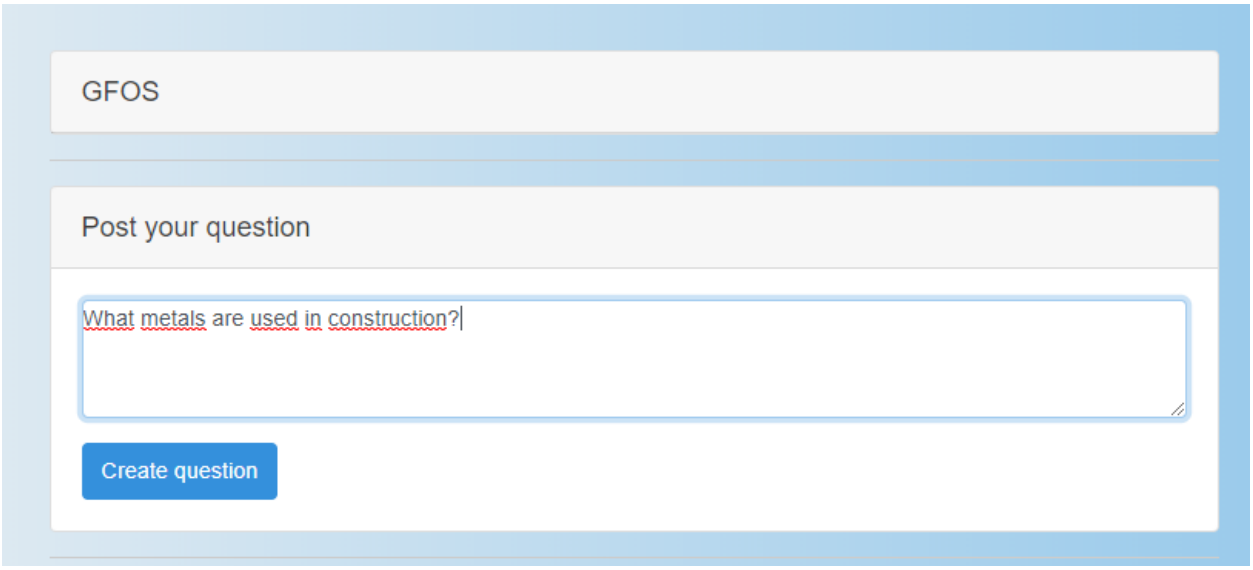
4.2. Prikaz funkcionalnosti foruma

Klikom na navigacijskoj traci na „Forum“ prebacujemo se na drugi dio aplikacije koji izgleda jednako kao i ukoliko kliknemo na Post, dakle otvara se stranica s listom fakulteta i mogućnošću dodavanja novog fakulteta na popis kao na 4.17.

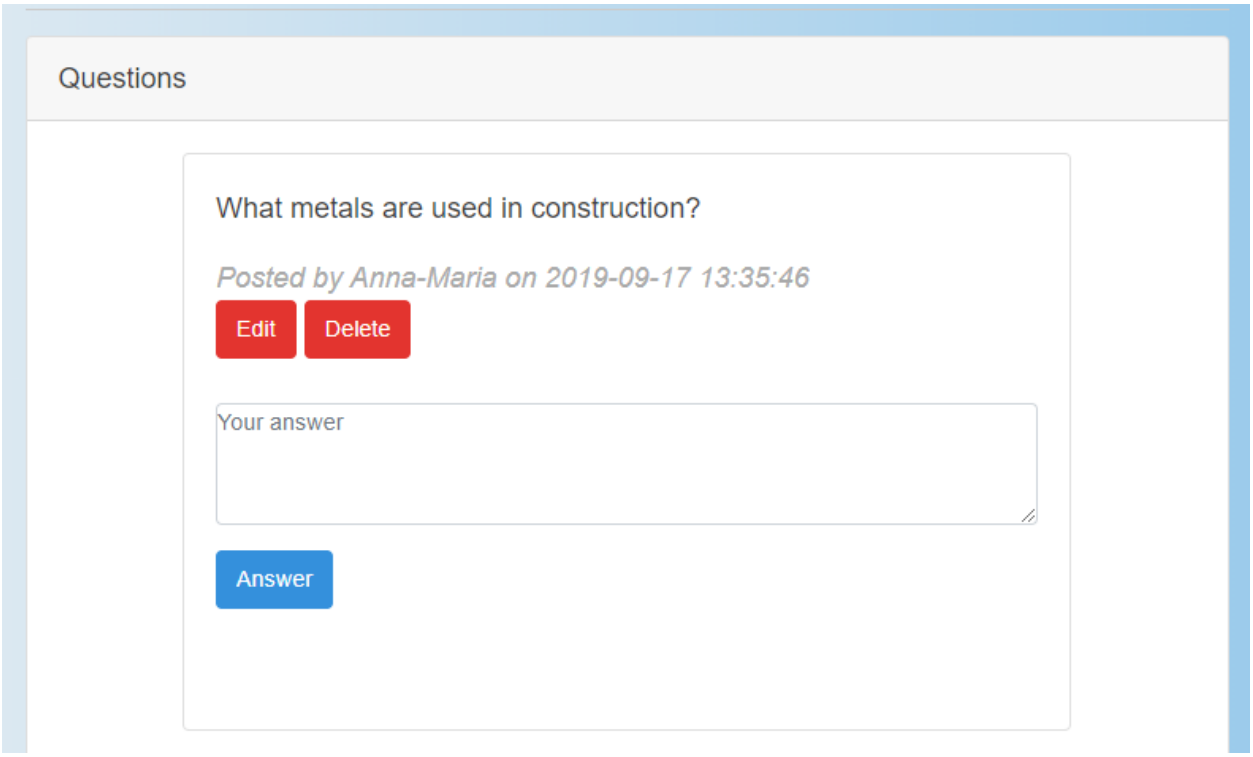


***Slika 4.17**. - Popis fakulteta*

Klikom na neki od ponuđenih fakulteta otvara se stranica koja predstavlja forum. Stranica sadrži ime odabranog fakulteta, sadrži dio za unos pitanja, te sadrži dio s postavljenim pitanjima. Unutar svakog pitanja su prikazani svi odgovori i postoji opcija dodavanja odgovora. Navedeno je prikazano na sljedećim slikama.

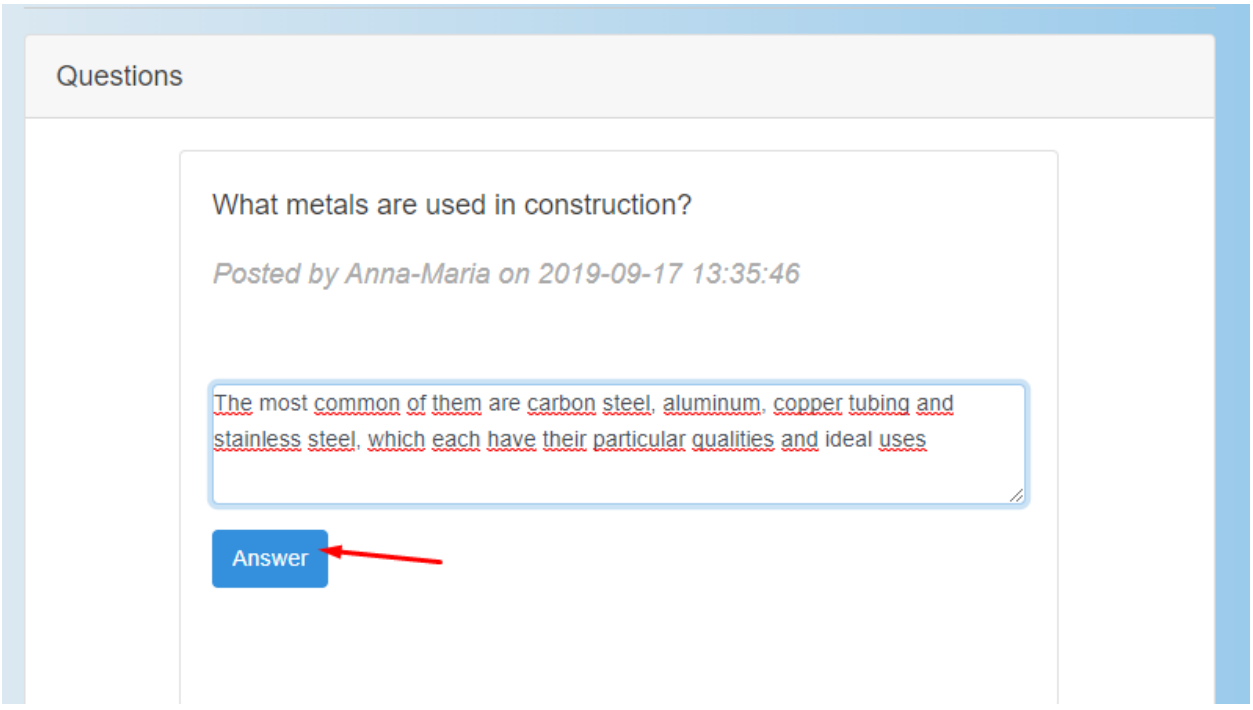


***Slika 4.18.** - Postavljanje pitanja na forumu*

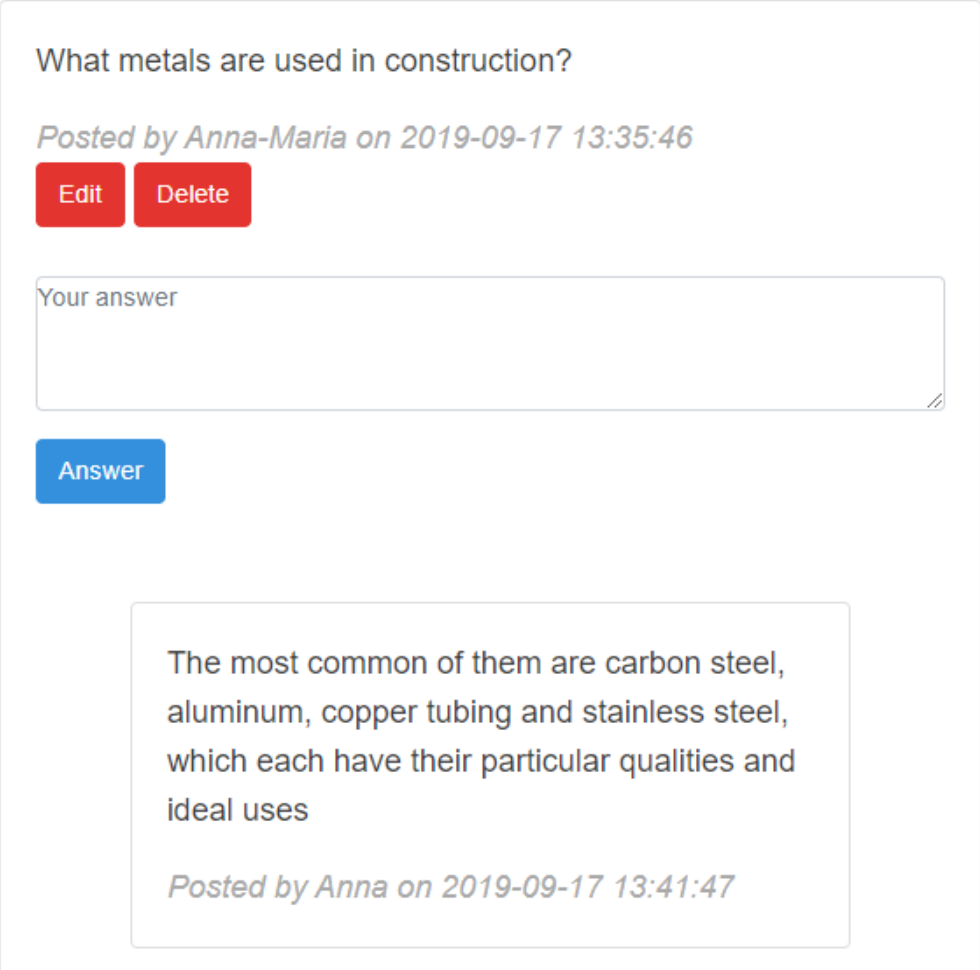


Slika 4.19. *- Prikaz pitanja na forumu*

Odgovaranje na pitanja omogućeno je i vlasniku pitanja i ostalim sudionicima foruma. Na sljedećim slikama prikazano je odgovaranje na pitanje od strane drugog korisnika.



Slika 4.20. *- Odgovaranje na pitanje na forumu*

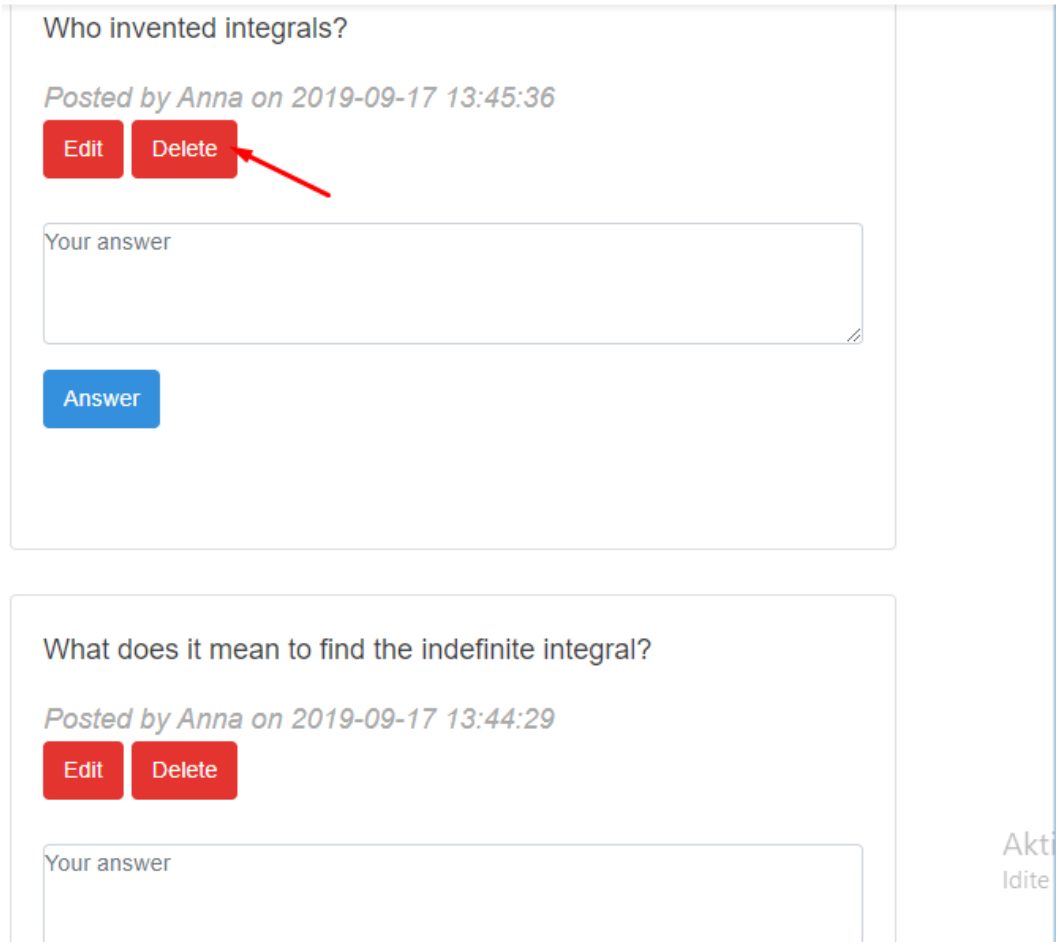


Slika 4.21. *- Prikaz odgovora na pitanje*

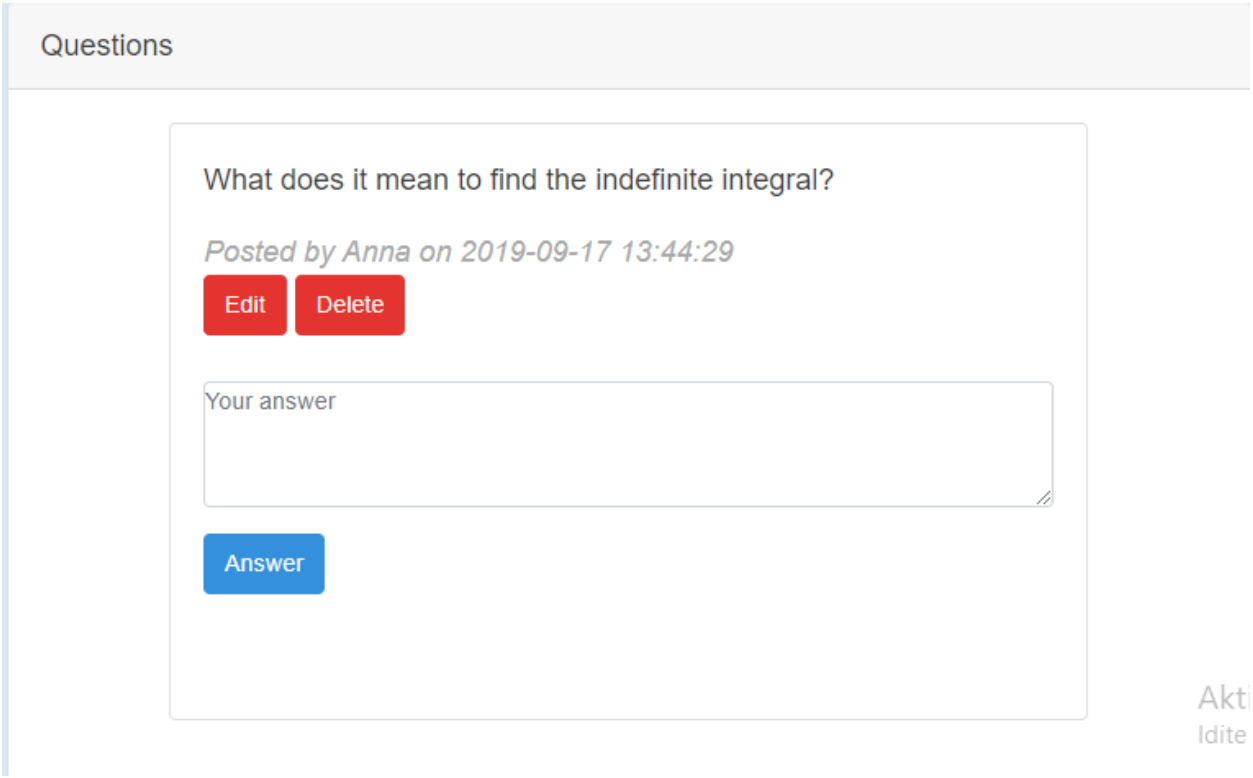
Kao što je vidljivo, drugom korisniku nisu vidljivi gumbovi za uređivanje i brisanje pitanja jer za to nema dozvolu.

4.2.1. Brisanje i uređivanje pitanja i odgovora na forumu

Ispod svakog pitanja vlasniku pitanja su prikazana dva gumba pomoću kojih može, na isti način kao i kod objava, obrisati ili urediti pitanje na forumu. Dakle, klikom na Delete gumb briše se pitanje s foruma kao što je prikazano na sljedećim slikama.

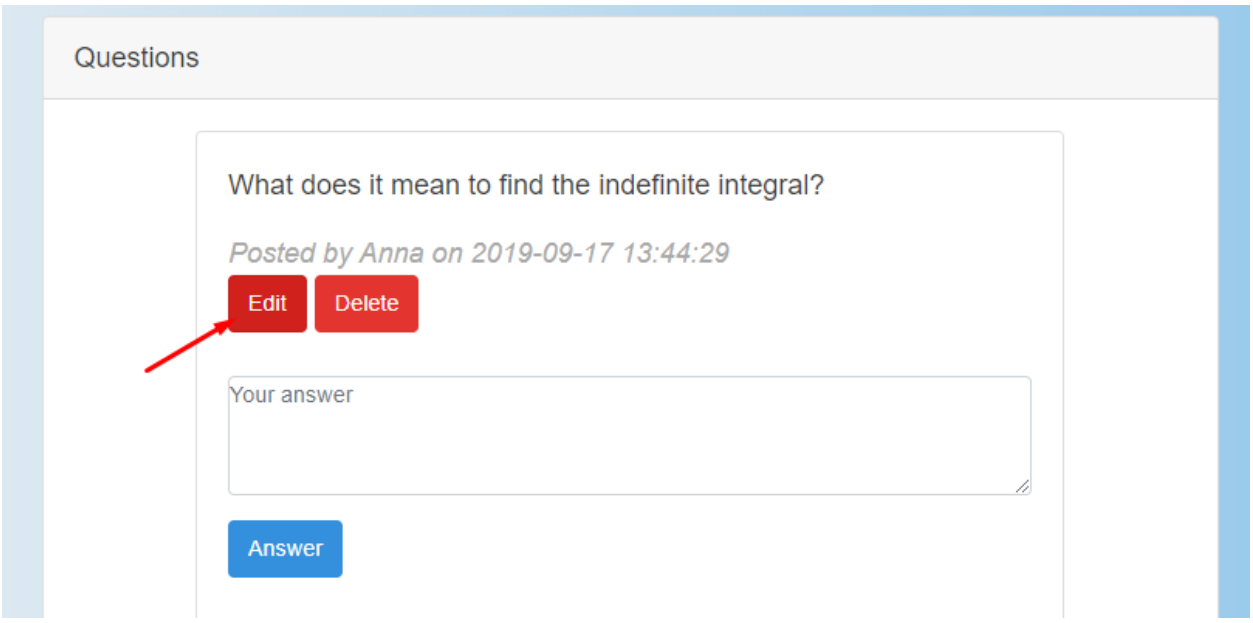


Slika 4.22. *- Brisanje pitanja s foruma*

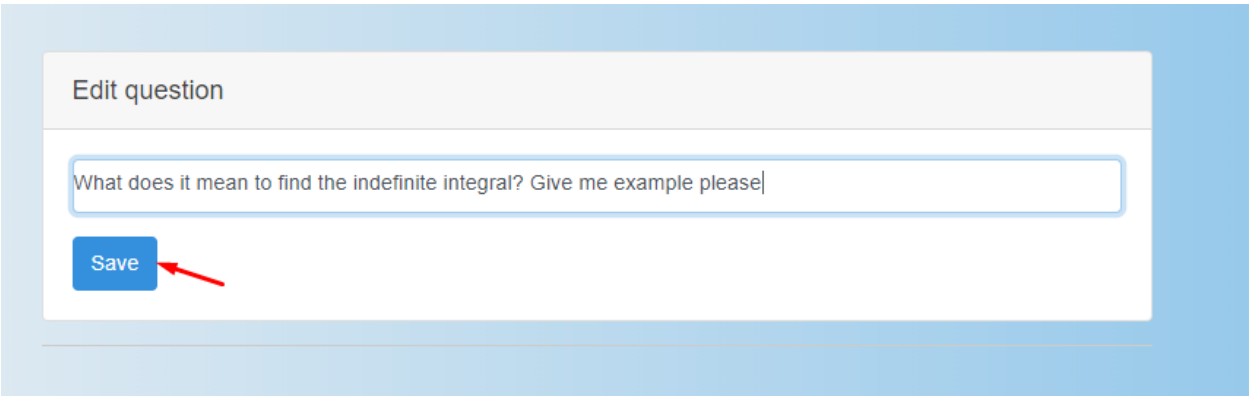


Slika 4.23. *- Pitanje je obrisano*

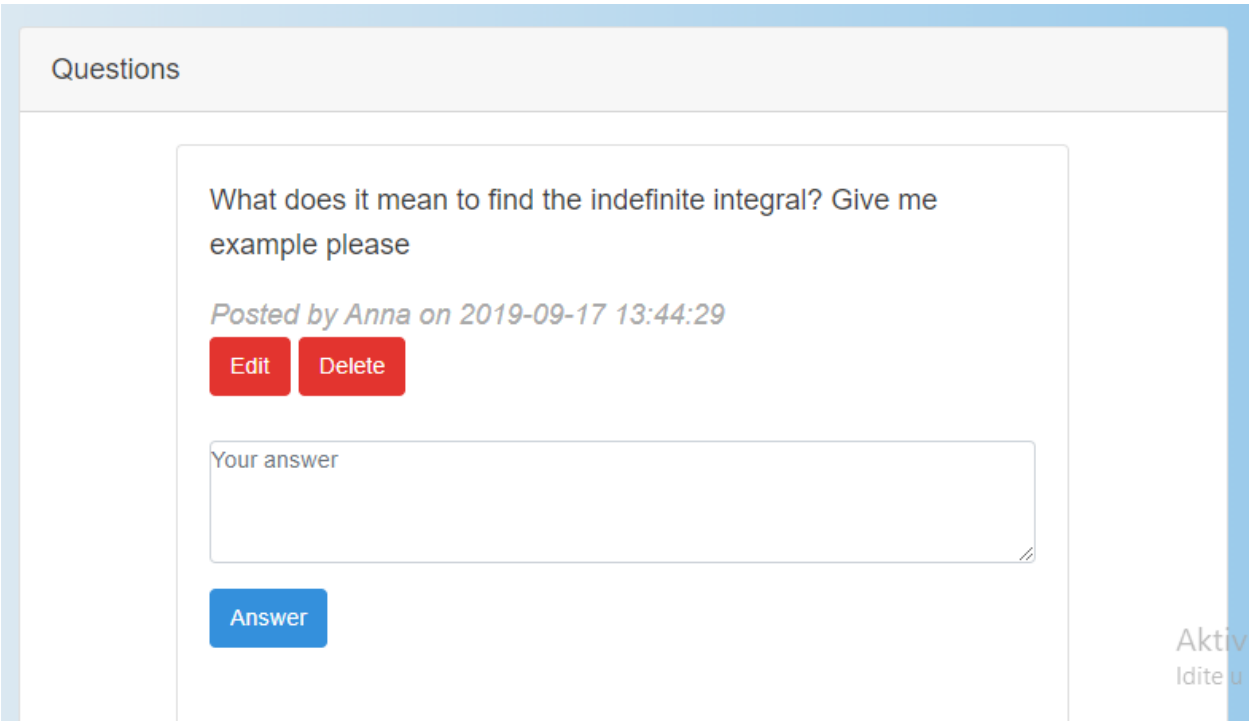
Klikom na gumb Edit otvara se novi pogled u kojem korisnik uređuje sadržaj svoga pitanja, te klikom na Save gumb u novom pogledu sprema uređeno pitanje koje se u tom obliku prikazuje na forumu i u bazi.



Slika 4.24. *– Otvaranje stranice za uređivanje pitanja*



Slika 4.25. *– Stranica za uređivanje teksta pitanja*



Slika 4.26. *- Uređen tekst pitanja*

Na isti način funkcionira i uređivanje i brisanje odgovora, klikom na Edit otvara se prozor za uređivanje odgovora, potom klikom na Save se sprema uređeni odgovor, a brisanje odgovora se obavlja klikom na Delete gumb. Naravno, ove radnje su dopuštene korisniku koji je vlasnik odgovora.

4.3. Baza podataka

Kako je već napomenuto korištena je MySQL baza podataka preko phpmyadmin web sučelja. Migracijama su stvorene tablice u bazi, tablica migracija je prikazana na slici 4.27.

	id	migration	batch
Edit Copy Delete	1	2014_10_12_000000_create_users_table	1
Edit Copy Delete	2	2014_10_12_100000_create_password_resets_table	1
Edit Copy Delete	6	2019_08_31_194247_create_threads_table	1
Edit Copy Delete	11	2019_09_07_183211_create_comments_table	3
Edit Copy Delete	12	2019_06_08_195858_create_posts_table	4
Edit Copy Delete	13	2019_09_11_120702_create_post_threads_table	4

***Slika 4.27**. – Tablica migracija u bazi podataka*

Tablica za spremanje thread-ova za objave je spremljena pod nazivom „post_threads“, a za spremanje thread-ova za komentare pod nazivom „threads“. Obje izgledaju isto, izgled je prikazan na sljedećoj slici.

	id	faculty	created_at	updated_at	user_id
Edit Copy Delete	1	FERIT	2019-09-11 13:10:25	2019-09-11 13:10:25	1
Edit Copy Delete	2	GFOS	2019-09-11 14:02:45	2019-09-11 14:02:45	1
Edit Copy Delete	4	PFOS	2019-09-11 18:17:54	2019-09-11 18:17:54	1
Edit Copy Delete	5	EFOS	2019-09-14 16:11:53	2019-09-14 16:11:53	1

***Slika 4.28.** -Tablica za spremanje fakulteta pod nazivom threads*

Tablica za spremanje objava je spremljena pod nazivom „posts“. Prethodno prikazane objave možemo vidjeti spremljene u toj tablici na slici 4.29.

id	created_at	updated_at	body	file	postable_id	postable_type	user_id
51	2019-09-13 17:36:04	2019-09-13 17:36:04	new	A.Dragic-prva verzija-diplomski rad.docx	1	App\PostThread	1
55	2019-09-17 12:05:36	2019-09-17 13:25:01	Korisna skripta iz Objektnog programiranja! Update...	OBJEKTNO-SKRIPTA.docx	1	App\PostThread	1
56	2019-09-17 14:00:46	2019-09-17 14:00:46	seminar iz menadzmenta	Menadžerska etika i održivi razvoj i društvena ...	5	App\PostThread	2
57	2019-09-17 14:02:02	2019-09-17 14:02:02	slika zaslona mobilne aplikacije	mobile-screen.png	5	App\PostThread	2
58	2019-09-17 14:02:58	2019-09-17 14:02:58	Signali skripta!	SIS-temeljna-za-2-kolokvij.docx	1	App\PostThread	2
59	2019-09-17 14:04:15	2019-09-17 14:04:15	Robot usisivac - seminar	Robot-usisivac_seminar_konacno.pdf	1	App\PostThread	2

Slika 4.29. *- Tablica "posts" u bazi podataka*

Dok je tablica za spremanje pitanja i odgovora spremljena pod nazivom „comments“, te njenu strukturu možemo vidjeti na slici 4.30.

id	body	user_id	commentable_id	commentable_type	created_at	updated_at
57	The most common of them are carbon steel, aluminum...	2	56	App\Comment	2019-09-17 13:41:47	2019-09-17 13:41:47
58	What does it mean to find the indefinite integral?...	2	5	App\Thread	2019-09-17 13:44:29	2019-09-17 13:48:42
60	Indefinite integral is more of a general form of i...	2	58	App\Comment	2019-09-17 14:06:18	2019-09-17 14:06:18

Slika 4.30. *-Tablica "comments" u bazi podataka*

5. ZAKLJUČAK

Budući da je internet postao dio svakodnevice svakog čovjeka, pa tako i studenata, realizirana je aplikacija koja ima za cilj olakšavanje studiranja na način da se kroz aplikaciju dijele skripte, korisni linkovi, koristan sadržaj i općenito da studenti svoja znanja prenose na druge studente. Aplikacija je realizirana uz pomoć neophodnih tehnologija kao što su HTML, CSS, JavaScript i PHP, te njihovih razvojnih okruženja Bootstrap i Laravel. Dok je za spremanje podataka korištena MySQL baza podataka preko myphpadmin web sučelja.

Dakle, aplikacija je koncipirana u dva dijela. Prvi dio je dio gdje studenti postavljaju koristan sadržaj s ciljem da pomognu ostalim kolegama. U tom dijelu također sam student može vidjeti objave drugih studenata te naći nešto korisno i za sebe. Drugi dio je koncipiran ako forum gdje postoji mogućnost postavljanja pitanja i odgovaranja na postavljena pitanja što je također jako korisno za studente ukoliko pri svom učenju i radu imaju problem da si smanje utrošak vremena i potraže pomoć od drugih kolega koji možda znaju odgovor.

Možemo zaključiti da je uz pomoć poznatih tehnologija kreirana vrlo praktična i korisna aplikacija kao pomoć pri studiranju.

LITERATURA

[1] Infosys, https://nanduri2kalyan.files.wordpress.com/2012/12/html_dhtml_javascript-infosys-material.pdf , (18.6.2019)

[2] w3schools , https://www.w3schools.com/html/default.asp, (20.6.2019)

[3] R., Nixon, **Learning PHP, MySQL, JavaScript, CSS & HTML5, 3rd Edition**, O'Reilly Media Inc., 1005 Gravenstein Highway North, Sebastopol, June 2014

[4] w3school , https://www.w3schools.com/css/css_syntax.asp , (21.6.2019)

[5] w3school, https://www.w3schools.com/bootstrap4/bootstrap_get_started.asp , (23.6.2019)

[6] StackOverflow Documentation, **PHP Notes for Professional** , Creative Commons BY-SA , June 2018

[7] Napredno web programiranje, predavanje 3, **Laravel**

SAŽETAK

Zadatak diplomskog rada je bio realizirati aplikaciju pod nazivom Aplikacija za online učenje koja je namjenjena studentima kako bi im olakšala studiranje lakim pronalaskom korisnog sadržaja i brzim dobivanjem odgovora na pitanja u vezi fakulteta. Za realizaciju aplikacije korištene su HTML, CSS, JavaScript i PHP tehonlogije, njihova razvojna okruženja Bootstrap i Laravel, te MySQL baza podataka za spremanje sadržaja. Aplikacija se sastoji iz dva dijela. Prvi dio je dio na kojem student postavlja sadržaj koji smatra da bi bio koristan kolegama, te također taj student može vidjeti sadržaj koji su drugi studenti postavili. Drugi dio aplikacije je koncipiran kao forum gdje student (korisnik) postavlja pitanje na koje mu drugi studenti mogu odgovoriti ukoliko znaju, kao što i on sam može odgovoriti na pitanja drugih studenata.

Ključne riječi: Laravel, PHP, web tehnologije, MySQL, web forum

ABSTRACT

ONLINE LEARNING APPLICATION

The task of the graduate thesis was to realize an application called the Online Learning Application designed for students to facilitate their studying with easier finding of useful content and quick response on questions with faculty issues. To implement application were used HTML, CSS, JavaScript and PHP technology, their frameworks Bootstrap and Larvel and MySQL database for content storage. The application consists of two parts. The first part is a part where student places content that he thinks that will be useful to his colleagues, and also that student can see other content which some other students set. The second part of application is conceived as a forum where a student (user) raises a question that other students can answer if they know answer, just as he can answer on other students questions.

Keywords: Laravel, PHP, web technology, MySQL, web forum

ŽIVOTOPIS

Anna-Maria Dragić rođena je 11.06.1995. u Bad-Honnefu u Njemačkoj. U Odžaku u Bosni i Hercegovini pohađala je osnovnu i srednju školu. Pohađala je srednju školu Pere Zečevića smjer Tehničar za računarstvo, te je tijekom školovanja u 2. i 3. razredu srednje škole obavljala praksu u tvrtci Eurocompany u trajanju od 80 sati. Srednju školu završava 2014., te upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, preddiplomski studij smjer Komunikacije i informatika koji završava 2017. s završnim radom pod nazivom Aplikacija za dohvat i pohranu zemljopisnih koordinata i upisuje diplomski studij smjer Mrežne tehnologije. U trećem semestru diplomskog studija obavilja stručnu praksu u trajanju od 200 sati u tvrtci ATOS u Osijeku kao Frontend developer. Trenutno je završna godina diplomskog studija Komunikacije i informatika na Fakultetu elektrotehnike, računarstva i informacijskih tehologija u Osijeku. Osim hrvatskog jezika, studentica se koristi i engleskim jezikom.

_____

Anna-Maria Dragić