Razvoj web aplikacije u Laravel razvojnom okruženju

Uremović, Alan

VELEUČILIŠTE U BJELOVARU
PREDDIPLOMSKI STRUČNI STUDIJ RAČUNARSTVO

RAZVOJ WEB APLIKACIJE POMOĆU LARAVEL RAZVOJNOG OKRUŽENJA

Završni rad br. 02/RAČ/2022

Alan Uremović

Bjelovar, veljača 2023.

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Student: **Alan Uremović** JMBAG: **0165075743**

Naslov rada (tema): **Razvoj web aplikacije u Laravel razvojnom okruženju**

Područje: **Tehničke znanosti** Polje: **Računarstvo**

Grana: **Informacijski sustavi**

Mentor: **Tomislav Adamović, mag. ing. el.** zvanje: **viši predavač**

Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:

1. **Krunoslav Husak, dipl. ing. rač., predsjednik**
2. **Tomislav Adamović, mag. ing. el., mentor**
3. **Ivan Sekovanić, mag.ing.inf.et comm.techn., član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 02/RAČ/2022

U sklopu završnog rada potrebno je:

1. Izraditi MySQL bazu podataka koja podržava sustav registracije, prijave, administracije i ovlasti
2. Primijeniti Laravel razvojno okruženje za spremanje korisničkih slika
3. Izraditi sustav blog postova putem Laravel razvojnog okruženja
4. U postojeću Laravel aplikaciju ugraditi mailgun server za slanje elektroničke pošte
5. Napisati Laravel migracijske skripte za izradu baze podataka
6. Ugraditi Eloquent alat za objektno-relacijsko mapiranje
7. Ugraditi u postojeću Laravel aplikaciju sustav za razmjenu poruka između korisnika

Datum: 30.06.2022. godine

Mentor: **Tomislav Adamović, mag. ing. el.**

Sadržaj

1. UVOD

U današnje vrijeme prije same izrade aplikacije, mora se postaviti pitanje: „Desktop aplikacije ili web aplikacije?“. Uz ovo pitanje dolazi i popratno pitanje, koji programski jezik izabrati. U svrhu izrade ovog završnog rada odabrana je web aplikacija, zbog samog rasta popularnosti web aplikacija, a odabrani programski jezik je PHP. Kako u današnje vrijeme postaje neuobičajeno programirati web stranice ili aplikacije od nule, odabrana je upotreba razvojnog okruženja zbog dodatnih funkcionalnosti koje su nativno uključene u razvojnom okruženju. Razvojna okruženja su dizajnirana da pomognu u razvoju web aplikacija, uključujući web servise i web API-e (*Application programming interface*). U odabiru PHP razvojnog okruženja može se birati između trenutno popularnih kao što su Symfony, CodeIgniter, CakePHP, Yii, Zend, Phalcon, Slim, FuelPHP, PHPixie i Laravel. Za potrebe ovog završnog rada odabrano je Laravel razvojno okruženje, zbog velike popularnosti, velike zajednice programera koja ga podržava duži niz godina i velike količine dokumentacije za učenje Laravel razvojnog okruženja.

U sklopu ovog završnog rada biti će objašnjeni osnovni koncepti razvojnih okruženja i Laravela, uz prednosti i mane Laravela kao i struktura projekta nastalog u Laravelu. Uz to, biti će objašnjen razvoj web aplikacije u Laravelu, uz popis programa i alata koji su potrebni za sam razvoj. Unutar poglavlja koje se bavi razvojem web aplikacije, svako podpoglavlje će se baviti s razvojem određenog dijela aplikacije: model, preusmjeravanje ruta, blade predlošci, pogledi i upravitelji.

2. WEB RAZVOJNA OKRUŽENJA I LARAVEL

U ovome poglavlju će ukratko biti opisana web razvojna okruženja i Laravel razvojno okruženje te prednosti i nedostatci istog uz prednosti i nedostatke PHP programskog jezika koji je sastavni dio Laravela.

2.1. Web razvojna okruženja

Web razvojna okruženja dizajnirana su da omogućuju razvoj web aplikacija, web servisa i web API-a. [1] Web razvojna okruženja sadrže prevoditelje, biblioteke koda, alate, programe podrške te API-e koji okupljaju različite komponente u svrhu razvoja projekta. Također razvojna okruženja mogu se podijeliti na dvije cjeline: programski kod koji je prethodno napisan i programski kod koji programer mora pisati. Kod koji je prethodno napisan služi kao temelj samom razvojnom okruženju, te programer nema potrebe zamarati se njime. Drugi dio programskog koda piše sam programer, ovisno o strukturi i cilju projekta. Može se zaključiti, da se razvojna okruženja koriste kako bi programeru pružala lakše i brže razvijanje aplikacije, bez nepotrebnog gubljenja vremena na implementacije redundantnog koda i implementacije koje su sadržane u prethodno napisanom programskom kodu razvojnog okruženja. Također postoje dvije različite vrste web razvojnih okruženja ovisno o potrebi programera, a dijele se na *front-end* razvojna okruženja i *back-end* razvojna okruženja. Ostala razvojna okruženja su CMS razvojna okruženja i razvojna okruženja za izradu mobilnih aplikacija.

Kada se koriste razvojna okruženja, potrebno je obratiti pažnju na koji način se izvodi kod na web aplikaciji. Odvija li se kod sa serverske strane ili s klijentske. U slučaju ovog završnog rada, korištenjem PHP-a i Laravel razvojnog okruženja, radi se o serverskoj strani izvršavanja koda, što znači da promjene unutar web aplikacije zahtijevaju slanje zahtijeva serveru te osvježavanje web stranice. Samim time može se zaključiti da je Laravel *server-side* razvojno okruženje. S druge strane kada se programski kod odvija na klijentskoj strani, obično se radi o JavaScript bibliotekama koje pokreću skripte unutar web preglednika, stoga te promjene ne zahtijevaju osvježavanje stranice. Razvojna okruženja koja koriste ovakvu izvedbu koda nazivaju se *client-side* razvojna okruženja.

2.2. Laravel

Laravel je besplatno razvojno okruženje koje je razvio Taylor Otwell, temeljeno na Symfony razvojnom okruženju, koje koristi MVC (engl. *Model-View-Controller*) arhitekturu. Prva verzija Laravela izdana je 9. lipnja 2011. godine, a trenutna verzija Laravel 9 izdana je 8. veljače 2022. godine. Zbog bolje i veće količine dokumentacije, za ovaj završni rad odabrana je Laravel 8 verzija razvojnog okruženja.

Laravel [2] je nastao s ciljem da programeru učini razvoj lakšim, omogući razvoj kompleksnijih i pouzdanijih aplikacija koje su istovremeno sigurne za korištenje i jednostavne za navigirati, zbog datotečne strukture Laravelovih projekata. Značajke koje predstavljaju svijetle točke Laravela su: modularnost, implementirana autentifikacija, predmemoriranje (engl. *cahcing*), upravitelj ovisnosti (engl. *dependency manager*), ugrađen PHPUnit za testiranje i ispravljanje grešaka, lako kreiranje lokalizacija, preusmjeravanje ruta (engl. *routing*), Blade mehanizam za izradu predložaka, migracije i Eloquent ORM podrška za baze podataka, te najbitnija značajka, sigurnost.

2.3. Prednosti i nedostatci Laravela

Laravel razvojno okruženje ima znatno više prednosti nego nedostataka. Za razliku od ostalih razvojnih okruženja Laravel pruža znatno bolju dugoročnu potporu te je razvijenu aplikaciju znatno lakše prilagoditi i modificirati, što podupire tvrdnju modularnosti Laravela. U pogledu sigurnosti, Laravel ima implementiran sustav autentikacije koji pruža dvije vrste enkripcije zaporke i jednostavnu izradu funkcionalnosti forme za prijavu ili registraciju. Također potrebno je spomenuti Eloquent ORM jer on pruža zaštitu od SQL injekcija. Uz to, potrebno je spomenuti zaštitu od *cross site* skriptiranja, koja sprječava injekciju malicioznih skripti prilikom korisničkog slanja kontaktne forme. Korištenje Blade predložaka, programeru je znatno pojednostavljen zadatak izrade pogleda i implementiranje funkcionalnosti upravljača, koje se završno povezuje s modelom i manipulacijom podataka koji se spremaju u bazu podataka. Bitno je spomenuti jednostavnu izradu migracija kroz koju je olakšana izrada tablica baze podataka i uređivanje tablice. Također jedna velika prednost je Laravelovo Artisan sučelje naredbenog retka kroz koji se obavlja stvaranje modela, migracija, upravljača, politika i ostalih.

Jedan od nedostataka Laravel razvojnog okruženja jest nadogradnja inačica. Bile to nadogradnje podinačica koje su bile potrebne od inačice 4.x kroz verziju 5.x ili nadogradnje inačica primjerice sa 8.x na 9.x. Odluka odabira inačice 8.x u svrhu izrade ovog projekta je velika količina dokumentacije i referentnih točki ukoliko se naiđe na neke probleme ili greške prilikom razvoja, ali bitno je spomenuti i da je nadogradnja s verzije 8.x na 9.x kompliciranа i nekoliko starijih funkcionalnosti što se tiče programskog koda i baze podataka postaju neiskoristive.

2.4. MVC struktura

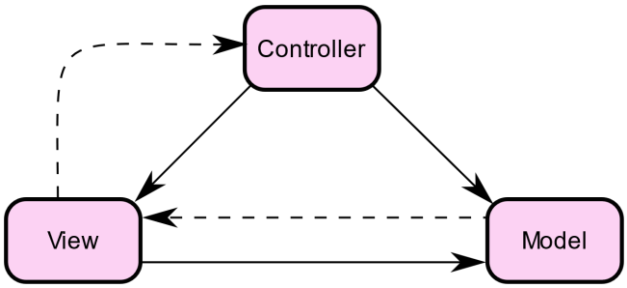
MVC (eng. *Model View Controller*) je [3] vrsta programske arhitekture kojom se opisuje način strukturiranja programskog rješenja. Trenutno je MVC struktura jedna od najpoznatijih u današnjem razvoju web aplikacija, a osim Laravela koriste je: CodeIgniter, Angular, Flask, Ruby on Rails te mnogi drugi. Primarni razlog korištenja MVC strukture jest odvajanje pojedinih dijelova aplikacije u manje cjeline ovisno o njihovoj namjeni.

Model predstavlja dio aplikacije koji je usko vezan uz tablice baze podataka. Obično se sastoji od jedne ili više klasa te unutar sebe sadržava metode kojima manipulira podatke unutar tablice baze podataka. Model također ima ulogu poveznice između pogleda i upravljačkog dijela programa.

Pogled (View) predstavlja korisničko sučelje ili ono što korisnik vidi prilikom ulaska u našu web aplikaciju. Sastavljen je od HTML-a i CSS-a, ali može sadržavati i manju količinu koda potrebnog za manje funkcionalnosti unutar samog pogleda koje nije nužno odvajati u zasebni upravljački dio. Također on predstavlja vezu između upravljačkog dijela i korisnika.

Upravljački dio (Controller) ili kontroler je odgovoran za zaprimanje zahtijeva od strane korisnika bilo to ispunjavanje forme, otvaranje poveznice ili ažuriranje podataka unutar baze podataka. Glavna uloga upravljačkog dijela je zapravo posrednička, služi kao veza između pogleda i modela.

Slika 2.1. prikazuje veze unutar MVC strukture. Strelice s isprekidanim linijama predstavljaju indirektnu povezanost, dok pune linije prikazuju direktnu povezanost. Prema slici može se stvarno vidjeti da upravljački dio služi kao poveznica između pogleda i modela.



Slika 2.1: MVC struktura

2.5. ORM, Eloquent i migracije

Objektno-relacijsko mapiranje [4] (ORM) je tehnika programiranja u kojoj se indentifikator metapodataka koristi za povezivanje programskog koda s bazom podataka. Samom uporabom Eloquent programskog koda može se vidjeti sličnost s objektno orijentiranim programiranjem iz razloga što modele predstavljaju klase. Eloquent [5] je prethodno implementirana ORM biblioteka unutar Laravela pomoću koje se ostvaruju veze s modelima i tablicama baze podataka. Način na koji Laravel stvara vezu između tablice baze podataka i modela je gramatička, što znači da naziv modela i tablice mora biti jednak. Ukoliko se model nazove „User“ tada će se tablica morati nazvati „users“, ali ako nam je model sastavljen od dvije riječi za model će biti korišten „PascalCase“, dok će tablica koristiti „Snake Case“ i dodati slovo „s“ na kraj imena modela. Tako će model „ForumChat“ biti preveden u „forum_chats“ tablicu. Glavna svrha Eloquenta je izbjegavanje pisanja SQL upita koji mogu nekad biti dugi i nepregledni. Neke od češće korištenih Eloquent metoda su:

- „all“ – kao rezultat vraća sve objekte modela kao Eloquent kolekciju modela, a ne kolekciju objekata. Primjer „$users=User::all()“ vraća kao rezultat kolekciju svih korisnika unutar naše tablice baze podataka.
- „get“ – ukoliko se koristi neki uvjet prilikom dohvaćanja nekog rezultata koristi se ova metodu. Primjer „$users=User::where('role',0)->get();“ će kao rezultat dohvatiti sve korisnike koji u tablici varijablu 'role' imaju postavljenu u 0. Ukoliko se izostavi „get“ iz ovog upita, neće se vratiti rezultat.
- „where“ – metoda koja postavlja uvjet upitu. Jednostavni oblik pisanja ove metode je „where('atribut_modela', 'vrijednost')“ i ona će provjeravati

jednakost. Ukoliko se želi drugačije pretraživanje između atributa modela i vrijednosti dodaje se relacijski operator.

- „first“ – ukoliko je potrebno dohvatiti prvi zapis iz tablice baze podataka, kao rezultat koristit će se ova metoda, primjerice „$user = User::all()“
- „save“ – metoda koja sprema novi model ili izmjenjuje postojeći, primjerice „$user->save();“ će spremiti „User“ model u bazu.
- „update“ – metoda koja se koristi za istovremenu izmjenu većeg broja zapisa. Često se koristi u kombinaciji s „where“ metodom.
- „delete“, „destroy“ – metode koje služe za brisanje modela iz baze podataka. Razlika između „delete“ i „destroy“ je da „delete“ treba kao argument imati dohvaćen objekt nad kojim će se izvršiti (npr. „user->delete();“ gdje je „$user“ prethodno dohvaćen u varijabu), dok metoda „destroy“ zahtijeva primarni ključ povezan s modelom (npr. „User::destroy(1);“). Isto tako „destroy“ metoda može primiti više primarnih ključeva (npr. „User::destroy([1,3,5]);“)
- „orderBy“ – metoda kojom se određuje kojim redoslijedom će se dohvaćeni podatci prikazati, uzlazno ili silazno s obzirom na odabrani stupac u tablici.

Lakoća manipulacije podataka je glavna značajka Eloquenta, jer pruža programeru izbjegavanje korištenja dugačkih SQL upita, što ubrzava pisanje upita i čini kod čitljivijim. Nakon što programer postavi veze između modela, te modela i tablica, omogućena je uporaba Eloquent metoda. Kao što je i prethodno spomenuto, korištenjem Eloquenta web aplikaciju se štiti od SQL injekcija.

U Laravelu [6] migracije su zamišljene kao sustavi kontrole verzije programskog koda i pružaju programerima mogućnost definiranja i dijeljenja sheme baze podataka. Naredba za stvaranje migracije, primjerice za tablicu koja će sadržavati sve naše objave na stranici je „php artisan make:migration create_posts_table“. Novonastala migracija će se pojaviti unutar „*database/migrations*“ direktorija i može se zamijetiti da ime migracije jest jednako datumu i vremenu kada je kreirana te naziv koji je postavljen unutar Artisan terminala prilikom kreiranja. Korištenjem „up“ metode u novostvorenoj migraciji mogu se dodavati ili modificirati tablice, pojedinačni stupci tablice i indeksi. Metoda „down“ funkcionira na principu da sve promjene provedene u „up“ metodi reverzira, što u većini slučajeva bude brisanje dodane tablice ili stupca. Prilikom stvaranja tablice u „up“ metodi je potrebno za sve planirane stupce koristiti varijablu „$table“ s odgovarajućim metodama koje govore Laravelu koji tip podataka će biti spremljen u pojedini stupac tablice.

Automatski generirani stupci prilikom kreiranja migracije su „id“ koji predstavlja unikatni identifikacijski broj koji se obično generira autoinkrementiranjem i „timestamps“ koji u tablici predstavlja dva stupca, u prvom je zapisan datum i vrijeme stvaranja dok u drugom stupcu zadnji datum i vrijeme izmjene nad podatkom. Prilikom stvaranje tablice objave navode se atributi: id, tekst objave, strani ključ koji će biti povezan s korisnikom i vremenske oznake (eng. *timestamp*). U programskom kodu 2.1. i 2.2. prikazane su metode za stvaranje i brisanje tablice baze podataka.

Programski kod 2.1.: Funkcija za kreiranje tablice „posts“ u bazi podataka

```
public function up()
    {
        Schema::create('posts', function (Blueprint $table) {
            $table->id();
            $table->text('content');
            $table->foreignId('user_id')
                  ->constrained()
                  ->onDelete('cascade');
            $table->timestamps();
        });
    }
```

Programski kod 2.2.: Metoda za brisanje tablice „posts“ u bazi podataka

```
public function down()
    {
        Schema::dropIfExists('posts');
    }
```

Osim migracije „create_posts_table“ u svrhu izrade ovog projekta kreirane su dodatne migracije:

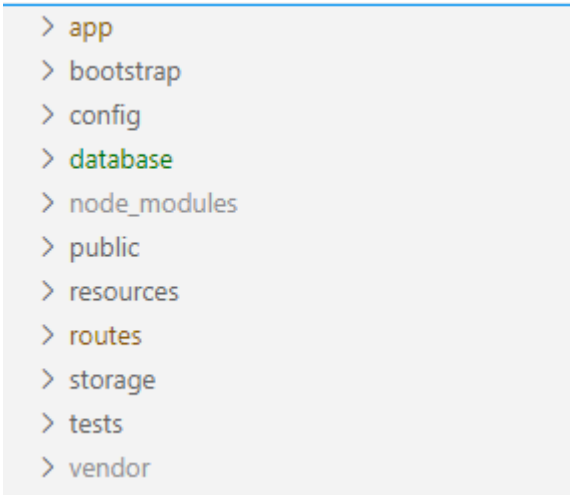
- „create_images_table“ – tablica u kojoj će biti sadržane putanje do slika, njihov naziv, te strani ključ koji je vezan uz objavu.
- „create_comments_table“ – tablica u kojoj se nalaze komentari pojedinih objava, uz zapise korisničkog imena i e-mail adrese korisnika.
- „create_forums_table“ – tablica u kojoj se spremaju sobe za čavrljanje, unutar njih nisu sadržane poruke korisnika.

- „create_forum_chats_table“ – tablica u kojoj se spremaju poruke svakog korisnika vezani uz pojedinu sobu za čavrljanje.
- „create_likes_table“ – tablica u kojoj se sprema broj pritisaka gumba „sviđa mi se“ za pojedinu objavu.

Da bi se „povukle“ sve migracije prema serveru potrebno je pokrenuti naredbu „*php artisan migrate“*.

2.6. Datotečna struktura projekta

Datotečna struktura novostvorenog projekta sadrži direktorije pod nazivom: „app“, „bootstrap“, „config“, „database“, „node_modules“, „public“, „resources“, „routes“, „storage“, „tests“, „vendor“. Najveći dio aplikacije samog projekta sadržan je unutar „app“ direktorija i primarno će se unutar njega odvijati najviše dodataka i promjena. Unutar direktorija „database“ nalaze se migracije kreirane u projektu, dok se u direktoriju „resources“ nalaze komponente pogledi, JavaScript skripte i CSS datoteke. Također je korišten i „routes“ direktorij u kojem postavljamo rute ovisno o potrebi web aplikacije, a one će biti dodatno objašnjene kasnije. Na slici 2.2 prikazana je datotečna struktura cijelog novostvorenog projekta.



Slika 2.2: Datotečna struktura Laravel projekta

2.7. Podešavanje konfiguracijske datoteke

Prije samog početka programiranja ili pokretanja web aplikacije, potrebno je podesiti konfiguracijsku datoteku Laravel projekta. Ona se nalazi u root direktoriju pod nazivom „.env“, a uz nju postoji [7] „.env.example“ konfiguracijska datoteka koja definira mnoštvo varijabli koje se prilikom instalacije Laravela automatski prepišu u „.env“ konfiguracijsku datoteku. Varijable koje je potrebno modificirati za normalan rad razvojnog okruženja:

1. „APP_KEY“ – Niz od 32 simbola služi u svrhe kriptiranja sesije i ostalih podataka. Ukoliko se aplikacija instalira preko Composer-a ili Laravel Installera ključ je automatski generiran. U suprotnom se ključ generira naredbom „php artisan key:generate“ kada se nalazi u korijenskom direktoriju projekta.
2. „DB_CONNECTION“ – Varijabla koja služi za povezivanje s određenom bazom podataka. Podržane baze podataka za povezivanje su sqlite, mysql i pqsql.
3. „DB_DATABASE“ – naziv baze podataka s kojom se ostvaruje povezivost
4. „DB_USERNAME“ , „DB_PASSWORD“ – korisničko ime i zaporka za povezivanje s bazom podataka
5. „MAIL_DRIVER“, „MAIL_HOST“,“MAIL_USERNAME“ i „MAIL_PASSWORD“ – parametri potrebni za upravljanje e-mailova. U projektu je korišten Mailgun kao e-mail server preko kojeg se obavlja smtp komunikacija. Prethodno je potrebno postaviti Mailgun na službenoj web stranici gdje se mogu dobiti podatci vezani uz zaporku potrebnu za funkcioniranje komunikacije.

Postoje i ostale varijable koje programer može postaviti, ali one nisu nužne jer ne ometaju proces razvoja aplikacije unutar razvojnog okruženja.

3. RAZVOJ FUNKCIONALNOSTI APLIKACIJE

U ovom poglavlju će biti opisan i objašnjen postupak instalacije potrebnih alata i programa, razvoja modela, pogleda i upravljača web aplikacije koja će služiti kao službena stranica Karate Kluba „Pitomača“. Web aplikacija će se ponašati kao društvena mreža u svrhu obrazloženja Laravelovih osnovnih koncepata i pogodnosti prilikom razvoja web aplikacije.

3.1. Potrebni alati i programi

Prije samog programiranja i instalacije Laravela, potrebno je imati instalirane programe i resurse. Za početak potrebno je imati instaliran Hypertext Preprocessor (PHP) programski jezik, u ovom slučaju aktualnu inačicu 8.1.9. Uz PHP potreban je Composer i opcionalno Node.js. Laravel pomoću Composera dohvaća s repozitorija potrebne pakete koji su ključni za nesmetan i efektivan rad razvojnog okruženja. Node.js s druge strane nije nužan za ispravan rad Laravela, ali on omogućuje instalaciju NPM-a. U svrhu razvoja ovog projekta nisu korišteni NPM paketi, stoga Node.js neće biti instaliran. Također će se koristiti programski alat WampServer koji u svojoj instalaciji ima uključen PHP, uz web poslužitelja Apache i MySQL baze podataka.

Nakon što su instalirani potrebni alati, može se stvoriti Laravel projekt. Stvaranje projekta odvija se u terminalu naredbenog retka, unutar kojeg se potrebno pozicionirati u željeni direktorij, te korištenjem Composerovih terminalnih naredbi može se stvoriti Laravel projekt. Postoje dvije mogućnosti kako kreirati Laravel projekt. Prva mogućnost je da se direktno preko [8] terminala i Composer-a upiše naredba „composer create-project laravel/laravel:^8.0 <naziv_projekta>“. Drugi način je da se Laravel instalira kao globalan Composerov dependency(ovisnost). Prilikom korištenja drugog načina koriste se dvije naredbe: „composer global require laravel/installer“, „laravel new naziv_projekta“. Nakon izvršenog prvog ili drugog načina kreiranja projekta, potrebno je pozicionirati se u korijenski direktorij i pokrenuti stranicu s naredbama: „cd naziv-projekta“ i „php artisan serve“. Kada je server pokrenut može se u internetski preglednik upisati http://localhost:8000/ te će se prikazati početna stranica novostvorenog projekta.

U svrhu bolje organiziranosti strukture samog projekta i jednostavnosti korištenja, koristi se *Visual Studio Code* uređivač programskog koda za potrebe organizacije

datotečne strukture unutar projekta, pisanje koda, korištenje ekstenzija i ugrađenog terminala za pokretanje naredbi. Instalacija samog alata je jednostavna i zahtijeva samo preuzimanje i pokretanje instalacijske datoteke sa službene web stranice. Nakon instalacije potrebno je konfigurirati *VS Code,* instalirati odabrane ekstenzije po potrebi i *VS Code* je spreman za korištenje.

3.2. Model

Kao što je prethodno napisano, model vrši manipulacije podataka nad tablicom baze podataka i on služi kao sama poveznica s pojedinačnom tablicom i njenim podatcima. U Laravelu stvaranje novog modela vrši se na sličan način kao i stvaranje migracija, pokretanjem Artisan naredbi u terminalu. Pokretanjem naredbe „php artisan make:model Post“ stvorit će se model pod nazivom „Post“. Ukoliko se uz model „Post“ želi omogućiti stvaranje pripadajuće migracije u prethodnu naredbu potrebno je dodati „-m“ na kraj. Također treba voditi brigu o sintaksi potrebnoj pri stvaranju modela. Standard koji se prati u Laravelu jest imenovanje modela korištenjem „PascalCase“ konvencije imenovanja.

Nakon što je model stvoren, potrebno je omogućiti pojedinim stupcima tablice baze podataka mogućnost manipulacije od strane korisnika, koja se postiže korištenjem slanja zahtijeva popunjenih formi. U okviru ovog projekta to su: prijava, registracija, kontakt forme, objavljivanje objava, slanje poruka unutar soba za čavrljanje, promjena korisničkih i administratorskih postavki. Zaštita manipulacije podataka postiže se korištenjem „$fillable“, „$hidden“ ili „$guarded“ varijabli. Podatci kojima korisnik manipulira upisani su unutar „$fillable“ varijable, dok podatci unutar „$hidden“ varijable budu sakriveni od korisnika, za razliku od „$guarded“ koji su zaštićeni i korisnik im ne može pristupiti niti ih modificirati. Kroz ovaj projekt Laravelove aplikacije u potrebnim modelima se primarno koriste „$fillable“ i „$guarded“ varijable. Nakon definiranja zaštite podataka potrebno je povezati modele, slično kako bi povezali tablice u relacijskim bazama podataka, no kao što je prethodno spomenuto, korištenjem Eloquenta dobiva se dojam korištenja objektno orijentiranog programiranja. U modelu „User“ koriste se četiri veze „jedan prema više“ (engl. *One to many*). U programskom kodu 3.3. može se vidjeti da su veze s ostalim modelima postignute pomoću javnih metoda koje su istog imena modela s kojim ih se spaja u množini. Unutar tih metoda može se vidjeti metoda „hasMany“ kojoj se prosljeđuje

klasa modela potrebnog za povezivanje. Ostali modeli koji nisu direktno povezani s „User“ modelom su „Image“, „Likes“ i „ForumChat“ modeli. Unutar ostalih modela korištena je samo „$fillable“ varijabla za masovnu serijalizaciju podataka.

Programski kod 3.1.: Programski kod „User“ modela

```
class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;

    protected $fillable = [
        'name',
        'email',
        'username',
        'password',
        'facebook_id',
    ];

    protected $hidden = [
        'password',
        'remember_token',
    ];

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

    public function chats(){
        return $this->hasMany(Forum::class);
    }

    public function likes(){
        return $this->hasMany(Likes::class);
    }

    public function forums(){
        return $this->hasMany(Forum::class);
    }
}
```

3.3. Postavljanje i preusmjeravanje ruta

Preusmjeravanje ruta u Laravelu omogućuje programeru da sve potrebne rute za funkcioniranje web aplikacije poveže s upravljačima. Najobičnije preusmjeravanje ruta [9] u Laravelu kao argument prima URI (engl. *Uniform Resource Identifier*) i metodu sadržanu u upravljaču. Funkcionalnost upravljača koji se povezuju s rutama unutar ovog projekta je prikazivanje pogleda i podataka aplikacije, ali i modifikacija podataka koji zahtijevaju korisnički unos. Lokacija svih ruta može se pronaći u „routes“ direktoriju unutar „web.php“ datoteke svakog projekta. Također može se koristiti i „api.php“ za registriranje ruta koje koriste RESTful API i „console.php“ za registriranje terminalnih Artisan naredbi.

Metode koje stoje na raspolaganju za potrebe obrade HTTP zahtijeva su sljedeće:

- „Route::get($uri, $callback)“ – GET HTTP zahtijev potreban za dohvaćanje stranice ili određenog resursa.
- „Route::post($uri, $callback)“ – POST HTTP zahtijev za spremanje resursa, obično u bazu podataka.
- „Route::put($uri, $callback)“ – PUT HTTP zahtijev za izmjenu resursa unutar baze podataka.
- „Route::patch($uri, $callback)“ – PATCH HTTP zahtijev za selektivnu izmjenu resursa.
- „Route::delete($uri, $callback)“ – DELETE HTTP zahtijev za brisanje resursa.
- „Route::options($uri, $callback)“ – OPTIONS HTTP zahtijev za otkrivanje dozvoljenih metoda nad zahtijevom.

U dolje prikazanom programskom kodu izdvojene su rute potrebne za funkcionalnost dohvaćanja, stvaranja i brisanja objava. Može se primijetiti da neke rute na kraju sadržavaju metodu „name()“ kojom se imenuje pojedina ruta za lakše kasnije korištenje unutar web aplikacije. Za potrebe ovog projekta korištene su metode „get“, „post“, „put“ i „delete“.

Programski kod 3.2.:Rute potrebne za dohvaćanje, stvaranje i brisanje objava

```
Route::get('/objave', [PostController::class, 'index'])
      ->name('post');
Route::post('/objave', [PostController::class, 'store']);
Route::delete('/objave/{post}', [PostController::class, 'destroy'])
      ->name('deletes');
```

Također prilikom objašnjenja ruta potrebno je napomenuti i [10] CSRF (engl. *Cross-site request forgeries*) zaštitu koja, kao što je prethodno spomenuto, onemogućuje lažiranje zahtijeva od strane napadača. Za svaku aktivnu sesiju uspoređuje se CSRF oznaka pojedinog korisnika. S obzirom da je CSRF oznaka pohranjena u aktivnoj sesiji pojedinog korisnika, napadač nije u mogućnosti pristupiti istoj. CSRF oznaka se kod krajnjeg korisnika generira kao sakrivena forma u kojoj je sadržan njen ključ aktivne sesije.

Također potrebno je napomenuti da HTML forme nativno ne podržavaju PUT, PATCH i DELETE metode, te ukoliko se koriste bez uporabe [11] „method_field“ polja, slanje ovih zahtijeva neće biti moguće. Stoga korištenjem „method_field“ polja postiže se lažiranje POST metoda i omogućuje slanje nativno nepodržanih metoda. U dolje prikazanom programskom kodu može se vidjeti korištenje „method_field“ polja u svrhu slanja zahtijeva za brisanje sobe za čavrljanje i korištenje CSRF oznake za sprječavanje protiv napada lažnih zahtijeva.

Programski kod 3.3.: Programski kod za brisanje sobe za čavrljanje

```
<form action=" {{ route('delete', $chat) }} " method="post">
      @method('DELETE')
      @csrf
      <button type="submit" class="text-danger bg-none">
            Delete
      </button>
</form>
```

U gore napisanom programskom kodu, može se primijetiti kako je korištena kraća nomenklatura pisanja „method_field“ i „CSRF“ polja. Duža verzija pisanja ovih polja zahtijeva postavljanje duplih vitičastih zagrada, primjerice za „@method('DELETE')“ u dužem obliku će biti „{{ method_field('DELETE') }}“. Duple vitičaste zagrade govore

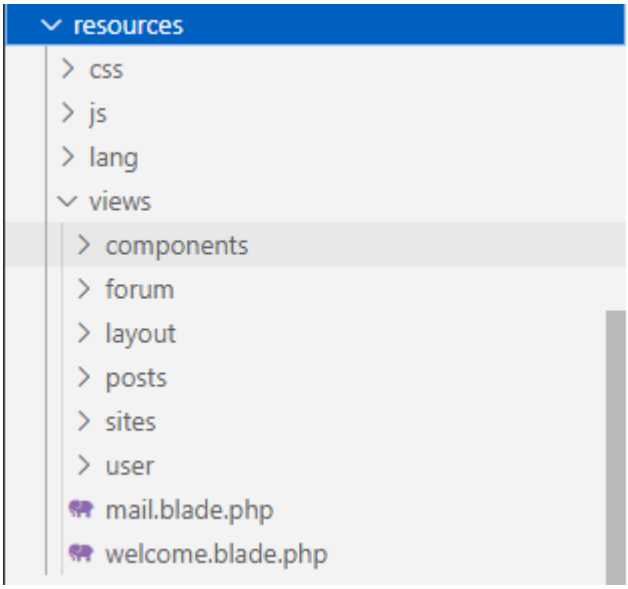
Laravelu da se radi o umetnutom PHP programskom kodu unutar pogleda koji su pisani HTML-om i CSS-om.

3.4. Stvaranje pogleda i Blade predložaka

U sklopu Laravel projekta [12] pogledi predstavljaju temeljni dio aplikacije, jer bez njih krajnji korisnik neće vidjeti web aplikaciju i neće biti u mogućnosti manipulirati s podatcima i formama. Svi pogledi unutar ovog projekta se nalaze u direktoriju „resources/views“. Prikaz pogleda krajnjem korisniku postiže se korištenjem upravljačkih dijelova aplikacije, koje će biti kasnije objašnjene detaljno.

Također, može se primijetiti dodatak „blade“ u imenovanju pogleda, što zapravo predstavlja Laravelov [13] *Blade Template Engine* koji omogućuje stvaranje glavnog predloška, proširivanje glavnog predloška prilikom stvaranja novih pogleda, prikaz podataka generiranih od strane upravitelja, umetanje logičkih petlji i čistog PHP programskog koda.

Stvaranje novog pogleda vrši se direktno u „resources/views“ direktoriju na način da se stvori nova datoteka te ju se nazove „*posts.blade.php*“ – pogled za prikazivanje objava. Radi lakšeg snalaženja unutar „views“ direktorija, stvoreni su poddirektoriji namijenjeni za pojedinu cjelinu aplikacije. U svrhu izrade ovog projekta stvoreni su poddirektoriji „components“, „forum“, „posts“, „sites“ i „user“ koji su prikazani na slici 3.1. Poddirektorij „layout“ je automatski generiran od strane Laravela.



Slika 3.1.: Prikaz strukture „resources/views“ direktorija

Bitno je napomenuti kako se unutar „layout“ poddirektorija nalaze glavni predlošci, koji služe da se izbjegne ponavljanje programskog koda. Tako primjerice u „master.blade.php“ postavljeni su header i footer stranice, to jest oni dijelovi stranice koji neće biti mijenjani unutar novokreiranih pogleda. Tako se može primijetiti da unutar „master.blade.layout“ pogleda postoji metoda „@yield('content')“ pomoću koje Laravel prepoznaje da se radi o dijelu koda koji će biti ubačen iz drugog predloška. Za potrebe prikazivanja sadržaja pomoću „@yield('content')“ metode potrebno je u drugom pogledu postaviti metode „@extends('layout.master')“, „@section('content')“ i „@endsection*“,* gdje prva metoda govori Laravelu u koji *„parent“* predložak se upisuje sadržaj *„child“* predloška, a druga i treća metoda enkapsuliraju sadržaj koji će biti umetnut u „master.blade.php“ datoteku. U ovom primjeru „*layout.master*“ predstavlja relativnu putanju do glavnog predloška, gdje nije potrebno upisivati datotečnu ekstenziju „*blade.php*“, već će sam Laravel zaključiti o kojem predlošku se radi. Unutar glavnog predloška pohranjene su CSS i JavaScript skripte čije funkcije budu korištene na svim novostvorenim pogledima.

Isto tako unutar pogleda moguć je prikaz podataka i korištenje logičkih petlji. Prikaz podataka moguć je korištenjem duplih vitičastih zagrada i imena varijable, primjerice prikaz imena korisnika „{{ $user->name }}“. Na isti se način pozivaju rute na koje stranica preusmjerava zahtijev korisnika. Logički izraz i petlje koje su sadržane u Bladeu su „@if“, „@for“, „@foreach“ i slično. Označavanje kraja logičkih izraza i petlji vrši se na način da se doda linija koda koja odgovara petlji i sadrži „end“ dio između „@“ simbola i naziva petlje, primjerice „*@foreach*“ označuje početak *foreach* petlje, a „*@endforeach*“ označava kraj. Parametri koji se koriste unutar zagrada petlji ne trebaju biti unutar dvostrukih vitičastih zagrada jer su one već implementirane putem Blade logičkih petlji. Sve Blade komponente koje rade s PHP parametrima se prevode u PHP kod, što znači da nije nužno koristiti Blade komponente, ali one čine kod urednijim i čitljivijim.

Također je bitno napomenuti da Laravel podržava komponente (engl. *components*) koje enkapsuliraju veliki dio programskog koda koji se koristi na više mjesta unutar web aplikacije. Stvaranje komponenti vrši se unutar terminala naredbom „php artisan make:component <naziv_komponente>“ te se može primijetiti da se stvorio predložak unutar „resources/views/components“ direktorija s datotečnom ekstenzijom „blade.php“ i unutar „app/view/components“ direktorija stvorena je datoteka za istu komponentu datotečne ekstenzije „php“, u kojoj se nalaze funkcije „__construct()“ i „render()“. Ukoliko

je potrebno koristiti anonimne komponente tada se može izbrisati datoteka koja je nastala unutar „*app/view/components*“ direktorija. Pisanje komponenti vrši se na isti način kao i pisanje HTML elemenata, primjerice komponenta za unos teksta se nalazi u direktoriju „*components/forms*“ tada će komponenta kao HTML element primiti naziv „*<x-forms.textbox/>*“. U dolje prikazanom kodu može se vidjeti korištenje komponente „*textbox*“ unutar forme za objavljivanje poruka u sobi za čavrljanje.

Programski kod 3.4.: Korištenje komponenti u formi

```
<form action="{{ route('store', [$chat->id]) }}" method="POST"
      enctype="multipart/form-data">
          @csrf
          <x-forms.textbox name="message" placeholder="Poruka..." />
          <button type="submit">Objavi poruku</button>
</form>
```

Može se primijetiti da komponente zatvaraju same sebe, to jest koriste „*self-closing tags*“ način pisanja elementa. S druge strane, ukoliko se koriste anonimne komponente unutar njihovog HTML programskog koda, potrebno je uključiti metodu „*@props()*“ koja za argumente prima varijable koje bi bile upisane u konstruktor metodu neanonimne komponente. Unutar „*@props()*“ metode anonimne komponente ili konstruktora neanonimne komponente se upisuju varijable koje se prosljeđuju komponenti za daljnje korištenje.

3.5. Upravitelji

Logika iza definicije ponašanja web aplikacije sadržana je unutar upravljača. Unutar upravljača moguće je grupiranje logike različitih zahtijeva unutar jedne klase. Može se zaključiti da je unutar jednog upravljača upisana logika za prikaz, stvaranje, modificiranje i brisanje podataka pojedinog modela. Stvaranje upravitelja u [14] Laravelu vrši se unutar Artisan terminala naredbom „php artisan make:controller <naziv_upravitelja>“. Kao što je „PascalCase“ konvencija imenovanja korištena za modele, tako je korištena i za nazive upravitelja, a najčešće nazivi upravitelja budu istoimeni modelima koji ih koriste, ali nije nužno da budu istoimeni. Tako će upravitelj vezan uz model „Post.php“ biti nazvan „PostController.php“. Ukoliko se želi

implementirati funkcionalnost CRUD naredbi pomoću upravitelja potrebno je koristiti sljedeće metode:

- „index“ – metoda za dohvaćanje i prikaz svih resursa uglavnom na pogledu.
- „create“ – metoda za prikaz forme za stvaranje novog podatka u tablici baze podataka.
- „store“ – metoda za spremanje novog podatka u tablici baze podataka.
- „show“ – metoda za dohvaćanje i prikaz jednog podatka pohranjenog unutar tablice baze podataka.
- „edit“ - metoda za prikaz forme za izmjenu pojedinog podatka unutar tablice baze podataka.
- „update“ – metoda za spremanje promjena pojedinog podatka u tablicu baze podataka.
- „destroy“ – metoda za brisanje pojedinog podatka iz tablice baze podataka.

U svrhu ovog projekta korištene su „index“, „store“, „update“ i „delete“ metode za postizanje funkcionalnosti cijele web aplikacije. Dalje u tekstu biti će objašnjene gore navedene metode za pojedini upravitelj i kako je postignuta pojedina funkcionalnost.

3.5.1. **Autentikacija i autorizacija korisnika**

Laravel pruža jednostavno stvaranje potrebnih upravitelja za potrebe registracije i prijave u web aplikaciju. Za stvaranje potrebnih upravitelja koristi se terminal. Za potrebe ovog projekta stvoreni su „RegistrationController“, „LoginController“ i „LogoutController“ korištenjem naredbi „php artisan make:controller <naziv_upravljača>“. Prikaz forme za registraciju postiže se korištenjem metode „*index*“ u kojoj se vraća „register“ pogled pomoću metode „view()“, a registracija i unos korisnika u bazu podataka vrši se preko metode „store(Request $request)“. Također unutar „RegistrationController“ i „LoginController“ koristi se „*middleware*“ mehanizam za filtriranje zahtijeva, tako da omogućuje samo gostima web aplikacije mogućnost registracije i prijave. Unutar „store“ metode provjerava se jesu li sva polja forme popunjena, te ukoliko jesu sprema korisnika u tablicu „User“ baze podataka.

Bitno je napomenuti da Laravel unutar sebe posjeduje sistem enkripcije, koji će se koristiti za kriptiranje zaporke korisnika. Enkripcija se postiže korištenjem metode „Hash::make()*“* koja prima string kao argument. Laravel za enkripciju koristi dva

upravljačka programa „*Bycrypt*“ i „*Argon2*“. Unutar ovog projekta korišten je upravljački program „*Bycrypt*“ za enkripciju zaporki. U svrhu ovog projekta prilikom uspješne registracije korisnika, istog će se automatski prijaviti u web aplikaciju i preusmjeriti na naslovnu stranicu web aplikacije. U dolje prikazanom programskom kodu prikazana je cjelovita struktura „RegistrationController“ upravitelja s gore spomenutim metodama.

Također u programskom kodu 3.5. može se primijetiti korištenje varijable „facebook_id“ koja će se kasnije koristiti prilikom registracije i prijave korisnika preko Facebook servisa. U dolje prikazanom programskom kodu stupac „facebook_id“ postavljen je u prazan string jer ukoliko se korisnik registrira preko e-mail adrese to polje neće biti popunjeno. Ukoliko se izostavi polje „facebook_id“ iz metode „User::create“ Laravel će javiti grešku jer nisu popunjeni svi stupci unutar tablice „User“ u bazi podataka. Taj problem se može riješiti na drugi način, da stupac „facebook_id“ posjeduje zadanu vrijednost unutar migracije koja je postavljena u prazan string.

Programski kod 3.5.: „RegistrationController“ upravljač sa pripadajućim metodama

```
class RegistrationController extends Controller
{
    public function __construct() {
        $this->middleware(['guest']);
    }

    public function index(){
        return view('sites.register');
    }

    public function store(Request $request){
        $this->validate($request, [
            'name' => 'required|max:255',
            'email' => 'required|email|max:255',
            'username' => 'required|max:255',
            'password' => 'required|confirmed'
        ]);

        User::create([
            'name' => $request->name,
            'email' => $request->email,
            'username'=> $request->username,
            'password'=> Hash::make($request->password),
            'facebook_id'=> '',
        ]);
```

```
            auth()->attempt($request->only('email','password'));

            return redirect()->route('home');
        }
    }
```

„LoginController“ upravljač je nešto jednostavniji od „RegistrationController“ upravljača iz razloga što se korisnika ne upisuje u bazu podataka, već se samo provjerava e-mail adresa i lozinka korisnika s unesenim vrijednostima. Unutar „LoginController“ upravljača također postoje „index“ metoda za prikazivanje forme za prijavu te „store“ metoda kojom se korisnik prijavljuje u web aplikaciju. Prijava korisnika vrši se unutar „if“ izraza, te ukoliko pristupni podatci nisu jednaki onima u bazi podataka, korisnik će biti preusmjeren na istu login formu sa porukom greške. Ukoliko korisnik unese točne podatke za prijavu biti će preusmjeren na naslovnu stranicu web aplikacije, isti slučaj kao i u „RegisterController“ upravljaču. U programskom kodu 3.6. prikazane su „index“ i „store“ metode potrebne za funkcioniranje prijave korisnika.

Programski kod 3.6.: „LoginController“ sa pripadajućim metodama

```
class LoginController extends Controller
{
    public function __construct() {
        $this->middleware(['guest']);
    }

    public function index(){
        return view('sites.login');
    }

    public function store(Request $request){
        $this->validate($request, [
            'email' => 'required|email',
            'password' => 'required',
        ]);

        if(!auth()->attempt($request->only('email','password'))){
            return back()->with('status','Upisani podatci su netočni!');
        }

        return redirect()->route('home');
    }
}
```

Sljedeća funkcionalnost koja je potrebna u svrhu izrade ovog projekta je odjava korisnika. Za potrebe odjave korisnika potrebna nam je samo jedna metoda „store“ preko koje se korisnik odjavljuje, te ga se preusmjerava na naslovnu stranicu isto kao i u „RegisterController“ i „LoginController“ upravljačima. Unutar metode „store“ potrebna je samo metoda „auth()->logout();“ i korisnik je odjavljen.

S druge strane, prijava korisnika preko Facebook servisa je nešto malo kompliciranija. Za potrebe prijave korisnika preko Facebook servisa stvoren je „FaceBookController“ upravljač te on za razliku od ostalih prije spomenutih upravljača ne koristi metode „store“ i „index“, već koristi „redirect“ i „callback“ metode. Također unutar ovog upravljača koristi se „Socialite“ upravljački program za komunikaciju s Facebookovim servisima. Sve potrebne osobne podatke korisnika upravljač uzima sa Facebookovog računa korisnika koji se želi prijaviti, a sama prijava vrši se pomoću identifikatora Facebook korisničkog računa. U nastavku je prikazan programski kod i prateće metode potrebne za funkcionalnost prijave preko Facebook servisa.

Programski kod 3.7.: “FaceBookController“ upravljač za prijavu korisnika pomoću Facebook servisa

```
class FaceBookController extends Controller
{
    public function redirect()
    {
        return Socialite::driver('facebook')->redirect();
    }

    public function callback()
    {
        try {
            $user = Socialite::driver('facebook')->user();

            $saveUser = User::updateOrCreate([
                'facebook_id' => $user->getId(),
            ],[
                'name' => $user->getName(),
                'username' => $user->getName(),
                'email' => $user->getEmail(),
                'password' => Hash::make($user->getName().'@'.$user
                                        ->getId())
                 ]);
```

```
            Auth::loginUsingId($saveUser->id);

            return redirect()->route('home');
            } catch (\Throwable $th) {
                throw $th;
            }
    }
}
```

U gore prikazanom programskom kodu upravljača „FaceBookController“ može se primijetiti da se stvaranje zaporke vrši pomoću „Hash::make()“ metode, no ona za argumente u ovom primjeru prima ime korisnika i njegov id, kojeg spaja u string pomoću znaka „@“. Ovdje dolazi do problema, ukoliko napadač zna kojom vrstom kriptiranja je stvorena lozinka te koje argumente prima metoda za kriptiranje, napadač se može prijaviti pomoću tih podataka upisivanjem u formu prijave.

S druge strane odjava korisnika odvija se na vrlo jednostavan način, stoga će upravljač imati malu količinu programskog koda. Unutar „LogoutController“ upravljača koristi se samo metoda „store“ u kojoj se korisnik odjavljuje te ga preusmjeruje na naslovnu stranicu. Dolje je prikazan korišteni programski kod za postizanje funkcionalnosti odjave.

Programski kod 3.8.: “LogoutController“ upravljač za odjavu korisnika

```
class LogoutController extends Controller
{
      public function store(){
            auth()->logout();
            return redirect()->route('home');
      }
}
```

3.5.2. **Objave sa slikama, komentari i „sviđa mi se“**

Web aplikacija je zamišljena kao društvena mreža na kojoj administratori dijele objave s korisnicima koji kasnije mogu ostavljati komentare na objavama ili ih označiti sa „sviđa mi se“. Funkcionalnost je postignuta korištenjem „PostController“,

„CommentController“ i „LikeController“ upravljačima. Upravljači su kreirani na isti način kao i prethodni upravljači, pomoću „artisan“ terminalnih naredbi.

Upravljač „PostController“ sadrži metode „index“, „store“ i „destroy“ za postizanje funkcionalnosti izlistavanja svih objava, objavljivanja objave i brisanje objava. Pomoću „index“ metode se vraća popis svih objava unutar baze podataka. Za dohvaćanje objava u redoslijedu novije pa starije, koristi se metoda „latest“ na „Post“ modelu. Za brže upite bazi koristi se metoda „with“ koja za argumente prima stupce koji se žele dohvatiti iz baze. Korištenje metode „with“ omogućuje brže učitavanje web stranice, a samom metodom i načinom slanja upita postiže se „eager loading“. „Eager loading“ značajno smanjuje broj upita koji se šalju te samim time poboljšava performanse web aplikacije. Na samom kraju, poslije dohvaćanja svih objava i spremanje istih u varijablu, vraća se pogled objave te ih se u „foreach“ petlji izlista unutar pogleda.

Programski kod 3.9.: Programski kod za dohvaćanje svih objava

```
public function index(){
        $posts = Post::latest()->with('images','user','comments',
                        'likes')->paginate(10);
        return view('posts.posts',[
            'posts'=>$posts
        ]);
    }
```

Može se primijetiti da se unutar „index“ metode koristi metoda „paginate“ prilikom dohvaćanja podataka. Metoda „paginate“ znatno pojednostavljuje paginaciju. Da bi paginacija funkcionirala kod krajnjeg korisnika, potrebno je u pogled dodati metodu „links“ koja se poziva nad varijablom u kojoj su pohranjene sve objave. Laravel će sam prepoznati da se radi o paginaciji te će u pogledu samostalno generirati kod potreban za funkcioniranje paginacije.

Sljedeća metoda koja se koristi je „store“ i pomoću nje se postiže spremanje objava u bazu podataka. Tijek metode store je da se prvo provjere upisani podatci u formi objave, nakon toga u bazu sprema se sadržaj objave. Prilikom provjere gleda se je li okvir za unos teksta popunjen sadržajem te ukoliko nije web stranica baca poruku korisniku da okvir za unos ne smije ostati prazan. Slike, s druge strane su opcionalne u objavi, stoga ta varijabla može primiti vrijednost „null“. Ukoliko slika postoji, u objavi je potrebno provjeriti

datotečnu ekstenziju slike i veličinu. Ukoliko je provjera uspješna slika se sprema. Spremanje slika odvija se u bazi podataka i u „Public“ direktoriju projekta. Prvotno se slike spremaju u „Public“ direktorij u zaseban poddirektorij za svakog korisnika i svaku objavu. Nakon što se slika spremila u direktorij, slika se također sprema u bazu podataka gdje su pohranjene informacije o imenu, putanji do slike i vanjski ključ koji je povezan s objavom. Kao posljednji korak ove metode, preusmjerava se korisnika nazad na pogled gdje su izlistane sve objave.

Programski kod 3.10.:Metoda „store“ za spremanje objava i slika

```
public function store(Request $request){
        $this->validate($request, [
            'content'=>'required',
            'images'=>'nullable',
            'images.*'=>'image|mimes:jpeg,png,jpg,svg|max:4096'

        ]);
        $post = $request->user()->posts()->create([
            'content' => $request->content
        ]);

        if($request->hasfile('images')){
            foreach($request->file('images') as $image){
                $name=$image->getClientOriginalName();
                $image_path=$image->move(
                  public_path().'\uploads\\'.$request->user()
                  ->username.'\\'.$post->id.'\\',$name);
                Image::create([
                    'name'=>$name,
                    'path'=>$image_path,
                    'post_id'=>$post->id
            ]);
            }
        }
        return back();
    }
```

Nakon spremanja objava potrebno je dodati funkcionalnost brisanja objava, koja je enkapsulirana unutar „destroy“ metode. Metoda „destroy“ za razliku od ostalih, prije spomenutih, metoda koristi provjeru korisničkih podataka da utvrdi koji korisnik može izbrisati koju sliku, a logiku iza provjere podataka izvršavaju politike (*eng. policies*). Politike u Laravelu se stvaraju na isti način kao modeli i upravljači, pomoću „artisan“

terminalnih naredbi. Politike i njihova logika je vezana direktno uz pripadajući model te se ne može koristiti i pozivati u drugim modelima. Primjerice, postoji „PostPolicy“ politika koja rješava potrebe „PostController“ upravljača, ukoliko se želi tu istu politiku koristiti nad drugim upravljačem to neće biti moguće. U sklopu ovog projekta politike su korištene samo za provjeravanje korisničke mogućnosti brisanja objava, a logika je postignuta pomoću „if“ izraza u kojem se provjerava je li isti korisnički identifikacijski broj s identifikacijskim brojem korisnika koji je stvorio objavu.

Programski kod 3.11.: Metoda politike za provjeru identifikacijskih brojeva korisnika

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

Pozivanje ove politike postiže se korištenjem metode „authorize“ koja kao argument prima ime metode unutar politike te argumente koji se prosljeđuju toj metodi. U ovom primjeru kao argument funkcije poslan je samo model objava iz razloga što će Laravel sam unutar politike varijablu „$user“ postaviti na trenutno prijavljenog korisnika. Nakon uspješne provjere unutar politike potrebno je izbrisati komentare, slike te samu objavu. Poslije uspješnog brisanja objave korisnika se preusmjerava na pogled koji izlistava sve objave.

Programski kod 3.12.:Metoda „destroy“ za brisanje objava i njenog kompletnog sadržaja

```
public function destroy(Post $post){
        $this->authorize('delete',$post);
        $post->comments()->delete();
        File::deleteDirectory(public_path().'\uploads\\'.$post->user
                                ->username.'\\'.$post->id);
        $post->images()->delete();
        $post->delete();
        return back();
    }
```

Zbog lakšeg snalaženja u programskom kodu i zbog bolje organiziranosti strukture projekta funkcionalnost komentara odvojena je u zaseban upravljač koji u sebi sadrži

metodu „store“. Unutar metode „store“, kao i u prošlim primjerima, provjera se je li okvir za unos teksta popunjen. Ako jest, sprema se novi komentar u bazu podataka sa sadržajem komentara, identifikacijskim brojem objave u kojoj se komentar nalazi, korisničkim imenom i e-mail adresom. Kao i kod ostalih metoda korisnika je potrebno preusmjeriti na stranicu koja izlistava sve objave.

Programski kod 3.13.: Metoda „store“ za spremanje komentara u bazu podataka

```
class CommentController extends Controller
{
    public function store(Request $request, $post_id){
        $this->validate($request, [
            'contents' => 'required|max:10000',
        ]);
        Comment::create([
            'username' => auth()->user()->username,
            'email' => auth()->user()->email,
            'contents' => $request->contents,
            'post_id'=> $post_id
        ]);
        return redirect()->route('post');
    }
}
```

Druga funkcionalnost koja je odvojena u zasebni upravljač je gumb „sviđa mi se“. Unutar upravljača „PostLikeController“ nalaze se dvije metode koje služe za označavanje objave sa „sviđa mi se“ i označavanje objave s „ne sviđa mi se“ koja se pojavljuje kao opcija samo u slučaju kada je objava već označena sa „sviđa mi se“. Za stvaranje „sviđa mi se“ zapisa u bazi podataka koristi se metoda „store“, dok se za brisanje tog podatka koristi metoda „delete“. Obje metode kao jedan od argumenata primaju objavu iz koje se uzima podatak o identifikacijskom broju objave koji je potreban za lakše pronalaženje objave prilikom brisanja zapisa iz baze podataka.

Programski kod 3.14.:Metode „store“ i „destroy“ upravljača „PostLikeController“

```
public function store(Post $post, Request $request){
    $post->likes()->create([
        'user_id' => $request->user()->id,
    ]);
    return back();
}

public function destroy(Post $post, Request $request){
    $request->user()->likes()->where('post_id', $post->id)
            ->delete();
    return back();
}
```

3.5.3. **Sobe za čavrljanje**

Da bi ovaj projekt poprimio značajke društvene mreže potrebno je dodati funkcionalnost izmjene poruka između korisnika. Sobe za čavrljanje zamišljene su da svi korisnici mogu razmjenjivati poruke sa svim ostalim korisnicima unutar soba za čavrljanje. Sobe za čavrljanje su javne što znači da bilo koji korisnik može čitati poruke ostalih korisnika. Funkcionalnost soba za čavrljanje smještena je unutar „ForumController“ i „ForumChatController“ upravljača. „ForumController“ upravljač koristi se za inicijalno stvaranje soba za čavrljanje, izlistavanje svih soba i brisanje istih. „ForumChatController“ upravljač služi za spremanje korisničkih poruka unutar soba za čavrljanje. Unutar ovih dvaju upravljača korištene su metode „index“, „store“i „destroy“ za potrebe izlistavanja svih soba za čavrljanje, spremanje i brisanje istih. Također je korištena specifična metoda „chatroom“ pomoću koje se izlistavaju sve poruke unutar jedne sobe za čavrljanje. Programski kod ovih upravljača nije potrebno dodatno objašnjavati jer se ne razlikuje uvelike od ostalih upravljača s istoimenim metodama.

3.5.4. **Slanje e-mail poruka administratoru**

Unutar ovog projekta potrebno je izraditi formu kojom korisnici ili gosti stranice mogu poslati upite prema administratoru sustava. Poruke poslane prema administratoru sustava moraju sadržavati određene informacije o korisniku kao što su: puno ime

korisnika, e-mail adresa korisnika, naslov poruke i sadržaj poruke. Poruke se šalju pomoću mail servera, a za ovaj projekt odabran je Mailgun poslužitelj jer pruža besplatnu opciju za korištenje usluga slanja, primanja i detaljan pregled svih poruka poslanih pomoću ovog poslužitelja. Prije samog slanja poruka potrebno je postaviti varijable za korištenje mail servera unutar „.env“ datoteke. Razmjena poruka se vrši pomoću „SMTP“ protokola, koji je najčešće korišten za razmjenu elektroničke pošte na internetu. Na niže prikazanoj slici mogu se vidjeti popunjeni argumenti potrebni za funkcioniranje slanja elektroničke pošte pomoću Mailgun poslužitelja. Na slici se mogu vidjeti zatamnjeni podatci koji su privatni i specifični za pojedini korisnički račun registriran na stranici Mailgun poslužitelja, stoga ih se ne smije dijeliti. Također može se primijetiti da se za kriptiranje elektroničke pošte koristi TLS protokol. Port koji se koristi za slanje elektroničke pošte jest standardni sigurni SMTP port 587.

```
30  MAIL_DRIVER=mailgun
31  MAIL_MAILER=smtp
32  MAIL_HOST=smtp.mailgun.org
33  MAIL_PORT=587
34  MAIL_USERNAME=[redacted]
35  MAIL_PASSWORD=[redacted]
36  MAIL_ENCRYPTION=tls
37  MAIL_FROM_ADDRESS=example@gmail.com
38  MAIL_FROM_NAME="${APP_NAME}"
```

Slika 3.2.:Parametri potrebni za funkcioniranje SMTP servera poslužitelja Mailgun

Sam izgled elektroničke pošte postavlja se u pogledu „mail_layout“ koji se kasnije poziva u metodi „build“ unutar klase „Email“. Potrebno je napomenuti da „Email“ nije model već „mailable“ klasa koja se generira pomoću „artisan“ terminalne naredbe „make:mail“ te se novonastala „Email“ klasa proširuje „mailable“ klasom. Unutar upravljača „EmailController“ korištena je metoda „contactSend“ u kojoj se provjeravaju podatci elektroničke pošte i nakon provjere šalju se na određenu e-mail adresu. Nakon slanja elektroničke pošte korisnika se vraća na stranicu na kojoj se prethodno nalazio prije slanja.

Programski kod 3.15.: Metoda „contactSend“ za slanje elektroničke pošte

```
public function contactSend(EmailRequest $request): RedirectResponse {
        $validated = $request->validated();

        Mail::to('test.test@gmail.com')->send(new Email($validated));

        return back();
}
```

3.5.5. **Postavke**

Unutar web aplikacije korisnicima je omogućeno pristupanje postavkama osobnog računa u kojima mogu promijeniti korisničko puno ime, korisničko ime, e-mail adresu i lozinku, ali mogu i obrisati korisnički račun. Običnim korisnicima web aplikacije izuzeta je mogućnost pristupa administratorskim postavkama u kojima administrator sustava može vidjeti sve registrirane korisnike unutar stranice i manipulirati podatkom je li korisnik gost, običan korisnik ili administrator sustava. Logika iza pokazivanja administratorskih postavki postavljena je unutar pogleda pomoću „if“ logičkih izraza koje provjeravaju status autoriteta prijavljenog korisnika. Metode za promjenu imena i e-mail adrese te metoda za brisanje korisnika iz baze podataka nije potrebno dodatno objašnjavati, jer se one ne razlikuju previše od ostalih metoda za spremanje podataka u bazu. Zahtijevnije metode su za promjenu lozinke korisnika i promjene autoriteta korisnika.

Prilikom promjene lozinke u metodi „updatePass“ koristi se više „if“ logičkih izraza od kojih prvi provjerava podudara li se stara zaporka korisnika koja je spremljena u bazu podataka sa starom zaporkom koja je upisana u tekstno polje, te drugi „if“ izraz koji provjerava podudara li se nova zaporka sa starom. Prilikom provjere korisnik će biti obavješten ukoliko je napravio pogrešku prilikom upisa stare zaporke ili ako se nova i stara zaporka podudaraju. Na kraju svakog „if“ izraza korisnika se vraća na stranicu na kojoj se prethodno nalazio, a to je u ovom slučaju stranica postavki računa.

Programski kod 3.16.:Metoda „updatePass“ za promjenu korisničke lozinke

```
public function updatePass(Request $request)
  {
      $request->validate([
          'old_password' => 'required',
          'password' => 'required',
      ]);
      $user = User::find($request->id);
      $old_password = auth()->user()->password;
      $new_password = $request->password;

      if (Hash::check($request->old_password, $old_password)) {
          if (!Hash::check($request->password, $old_password)) {
              $user->password = Hash::make($new_password);
              $user->save();
              session()->flash('message', 'password updated
                              successfully!');
              return back();
          } else {
              session()->flash('message', 'new password cannot be the
                              old password!');
              return back();
          }
      } else {
          session()->flash('message', 'old password doesn\'t match');
          return back();
      }
  }
```

Može se primijetiti, da se kod provjere zaporki koristi metoda „Hash:check“ koja kao argumente prima prvo tekst zaporke bez kriptiranog sadržaja te drugo kriptiranu zaporku koju unutar metode dekriptira i uspoređuje. Ukoliko bi se unutar funkcije izostavila metoda „Hash:check“ provjera ispravnosti i podudaranje zaporki ne bi funkcionirala.

Metoda „updateAuth“ koja služi administratoru sustava za promjenu autoriteta ostalih korisnika stranice provjerava sve zapise varijable autoriteta unutar baze i forme unutar postavki. Ukoliko se jedan od zapisa razlikuje od onoga u bazi podataka, samo će ta izmjena biti pohranjena unutar baze podataka. Unutar metode prije same provjere podatka potrebno je dohvatiti sve podatke i spremiti ih u varijable zbog lakšeg dohvaćanja pojedinačnih zapisa kroz koje se prolazi unutar „foreach“ petlje.

Programski kod 3.17.:Metoda „updateAuth“ za promjenu autoriteta korisnika

```
public function updateAuth(Request $request){
    $ids = $request->id;
    $users = User::get();
    $auths = $request->authority;
    foreach ($users as $user) {
        for ($i = 0; $i < count($ids); $i++) {
            if ($user->id == $ids[$i] &&
                !($user->authority==$auths[$i])==false) {
                $user->authority = $auths[$i];
                $user->save();
            }
        }
    }
    return back();
}
```

Po završetku pisanja programskog koda ovog upravljača postignuta je potpuna funkcionalnost cijele web aplikacije.

4. ZAKLJUČAK

Razvoj web aplikacija u današnje vrijeme postaje kompleksniji i u skladu s time dolazi do potrebe razvoja alata koji će pomoći u samom razvoju web aplikacija, gdje razvojna okruženja predstavljaju vrlo važan i nekada neizbježan alat.

Ovaj završni rad daje uvid u koncepte razvojnih okruženja, prednosti i nedostatke Laravela i PHP programskog jezika koji mogu čitatelju služiti kao pomoć u dvojbi njihovog korištenja. Stoga se daju osnovni koncepti Laravela, postupci instalacije, konfiguracije i razvoja Laravel web aplikacije ili drugog web riješenja pomoću ovog razvojnog okruženja.

Glavni cilj ovog rada bio je dati uvid čitatelju u mogućnosti Laravel razvojnog okruženja, obrazložiti korištenje PHP programskog jezika koji je na lošem glasu kao neefikasan, što je postignuto uspješnim razvojem web aplikacije društvene mreže za potrebe Karate Kluba „Pitomače“, koju je lako nadograditi, izmjeniti i testirati. Također bitno je napomenuti uz lakoću nadogradnje, izmjene i testiranja, krajnje rješenje web aplikacije postignuto je s efikasnim i brzim performansama, koje su u današnje vrijeme neizbježne za normalno funkcioniranje web rješenja.

5. LITERATURA

[1] Intelegain Technologies. What are web frameworks and why you need them. [Online]. 2019. Dostupno na: https://intelegain-technologies.medium.com/what-are-web-frameworks-and-why-you-need-them-c4e8806bd0fb (12.8.2022.)
[2] Koffer P. A brief guide through Laravel[Online]. 2022. Dostupno na: https://mdevelopers.com/blog/a-brief-guide-through-laravel (12.8.2022.)
[3] Codecademy. MVC: Model, View, Controller [Online]. 2022. Dostupno na: https://www.codecademy.com/article/mvc (14.8.2022.)
[4] Liang M. Understanding Object-Relational Mapping: Pros, Cons, and Types [Online]. 2011. Dostupno na: https://www.altexsoft.com/blog/object-relational-mapping/ (16.8.2022.)
[5] Laravel. Eloquent: Getting Started [Online]. 2022. Dostupno na: https://laravel.com/docs/8.x/eloquent (16.8.2022.)
[6] Laravel. Database: Migrations [Online]. 2022. Dostupno na: https://laravel.com/docs/8.x/migrations (16.8.2022.)
[7] Laravel. Configuration [Online]. 2022. Dostupno na: https://laravel.com/docs/8.x/configuration (16.8.2022.)
[8] Laravel. Getting Started [Online]. 2022. Dostupno na: https://laravel.com/docs/8.x (13.8.2022.)
[9] Laravel. Routing [Online]. 2022. Dostupno na: https://laravel.com/docs/8.x/routing (16.8.2022.)
[10] Laravel. CSRF Protection [Online]. 2022. Dostupno na: https://laravel.com/docs/8.x/csrf (16.8.2022.)
[11] Laravel. Helpers [Online]. 2022. Dostupno na: https://laravel.com/docs/8.x/helpers (20.8.2022.)
[12] Laravel. Views [Online]. 2022. Dosutpno na: https://laravel.com/docs/8.x/views (20.8.2022.)
[13] Laravel. Blade Templates [Online]. 2022. Dostupno na: https://laravel.com/docs/8.x/blade (20.8.2022.)
[14] Laravel. Controllers [Online]. 2022. Dostupno na: https://laravel.com/docs/8.x/controllers (20.8.2022.)

6. OZNAKE I KRATICE

API – application programming interface

PHP – Hypertext Preprocessor

CMS – Content management system

MVC – Model-View-Controller

ORM – Object-relational mapping

HTML – Hypertext Markup Language

CSS – Cascading Style Sheets

SQL – Structured Query Language

URI – Uniform

REST – Representational state transfer

HTTP – Hypertext Transfer Protocol

CSRF – Cross-Site Request Forgery

CRUD – Create, read, update, delete

SMTP – Simple Mail Transfer Protocol

TLS – Transport Layer Security

7. SAŽETAK

Naslov: Razvoj web aplikacije u Laravel razvojnom okruženju

Rad se bavi analiziranjem koncepata razvojnih okruženja, prednosti i nedostataka Laravela i PHP programskog jezika u razvoju web aplikacija. Prvi dio ovog rada bavi se objašnjavanjem osnovnih koncepata kao što su MVC arhitekture, objektno-relacijsko mapiranje, tijek razvoja web aplikacije korištenjem Laravel razvojnog okruženja. Glavni dio rada sadrži korake i objašnjenja razvoja web aplikacije koja služi kao društvena mreža u kojoj je uključena autentikacija, autorizacija, migracije i modeli, usmjeravanje zahtijeva, upravljača, pogleda i ostalih komponenti.

Ključne riječi: Laravel, MVC, ERM, Eloquent, , PHP, razvojno okruženje

8. ABSTRACT

Title: Web application development inside Laravel framework

The thesis analises framework concepts, advantages and disadvantages of the Laravel framework and PHP programming language within development of web applications. The first part of the thesis contains explanations of basic concepts such as the MVC architecture, object-relational mapping and the development course of the web application using Laravel web application framework. The main part of the thesis contains steps and explanations of the social network web application development, which includes the implementation of authentication, authorization, migrations and models, request routing, controllers, views and other components.

Keywords: Laravel, MVC, ERM, Eloquent, Artisan, PHP, framework

IZJAVA O AUTORSTVU ZAVRŠNOG RADA

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, 30.9.2022	Alan Uremović	

Prema Odluci Veleučilišta u Bjelovaru, a u skladu sa Zakonom o znanstvenoj djelatnosti i visokom obrazovanju, elektroničke inačice završnih radova studenata Veleučilišta u Bjelovaru bit će pohranjene i javno dostupne u internetskoj bazi Nacionalne i sveučilišne knjižnice u Zagrebu. Ukoliko ste suglasni da tekst Vašeg završnog rada u cijelosti bude javno objavljen, molimo Vas da to potvrdite potpisom.

Suglasnost za objavljivanje elektroničke inačice završnog rada u javno dostupnom nacionalnom repozitoriju

Alan Uremović

ime i prezime studenta/ice

Dajem suglasnost da se radi promicanja otvorenog i slobodnog pristupa znanju i informacijama cjeloviti tekst mojeg završnog rada pohrani u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu i time učini javno dostupnim.

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 30.9.2022.

potpis studenta/ice