Razvoj aplikacije u Laravel okruženju

Klarić, Magda

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO–MATEMATIČKI FAKULTET
MATEMATIČKI ODSJEK

Magda Klarić

RAZVOJ APLIKACIJE U LARAVEL OKRUŽENJU

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Saša Singer

Zagreb, veljača 2018.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Andreju, mojoj potpori u najljepšim i najtežim trenucima.
Roditeljima, braći, sestri i prijateljima koji su bili uz mene tijekom školovanja,
poštivali moj put i učinili to razdoblje ljepšim i lakšim.

Sadržaj

Sadržaj **iv**

Uvod **1**

1 Razvojno okruženje **2**
1.1 Pojam 2
1.2 Prednosti i mane 3
1.3 PHP razvojna okruženja 5

2 Laravel **7**
2.1 Povijest Laravela 7
2.2 Usporedba Laravela s drugim razvojnim okruženjima 9

3 Značajke Laravela **11**
3.1 Homestead 11
3.2 Artisan 15
3.3 Blade 17
3.4 Još neke značajke 20

4 MVC **22**
4.1 Model 23
4.2 View 26
4.3 Controller 26

5 Tok usmjeravanja (Routing) **31**
5.1 Rute koje usmjeravaju na kontroler 33
5.2 Parametri rute 34
5.3 Resursne rute 35

6 Baza podataka **38**

6.1 Postavljanje baze podataka 38
6.2 Migracije 40
6.3 Pokretanje neobrađenih SQL upita 45
6.4 Graditelj upita 47
6.5 Relacije 49
6.6 Seeding 58

7 Autentikacija **61**
7.1 Uključeni dijelovi za autentikaciju 61
7.2 Postavljanje autentikacije 63
7.3 Dohvaćanje autenticiranog korisnika 65
7.4 Zaštita ruta 66

8 Primjena **67**

Bibliografija **75**

Uvod

Zbog široke upotrebe PHP-a, programskog jezika namijenjenog primarno razvoju dinamičnih internet stranica i aplikacija te rastućeg broja korisnika, u zajednici su se počela razvijati mnoga PHP okruženja (frameworks). Svrha okruženja je olakšati, odnosno preskočiti kodiranje čestih i ponavljajućih situacija u izradi svake web aplikacije te ubrzati njen razvoj. Jedno od najpopularnijih i modernijih PHP okruženja koje se u današnjici ističe je Laravel — besplatno, open-source okruženje zamišljeno za razvoj web aplikacija na temelju MVC (Model-View-Controller) paradigme.

Kroz prva dva poglavlja ovog rada, cilj je čitatelja upoznati s pojmom razvojnog okruženja te nastankom Laravela kao PHP razvojnog okruženja. Osim toga, cilj je naglasiti prednosti i mane razvojnih okruženja te istaknuti svojstva koja ističu Laravel nad ostalim PHP okruženjima.

U nastavku rada detaljnije se opisuje pozadina izrade web aplikacije u Laravelu, njegove značajke te implementacija glavnih komponenti kroz modele, poglede i kontrolere, dijelove MVC paradigme. Opisan je proces instalacije i kreiranja novog projekta te je uključeno objašnjenje najvažnijih pojmova, usmjeravanje kroz web aplikaciju te implementacija baze podataka.

Na samom kraju opisana je aplikacija izrađena kao primjer razvoja aplikacije u ovom okruženju. Međutim, kôd aplikacije proteže se kroz cijeli rad, prikazan kroz jednostavnije i složenije primjere korištenja Laravelovih komponenata i alata za izradu aplikacije.

Konačni cilj ovog rada je pokušati čitatelju približiti Laravel i kroz praktične primjere demonstrirati način rada ovog okruženja, što će sve čitatelju možda jednog dana olakšati izradu kvalitetne web aplikacije.

Poglavlje 1

Razvojno okruženje

1.1 Pojam

Software framework ili razvojno okruženje (kraće, *okruženje*) je konkretna ili konceptualna platforma u kojoj se odabrani, česti i ponavljajući dijelovi koda opće funkcionalnosti mogu specijalizirati ili prenamijeniti od strane programera ili korisnika, tako kreirajući aplikacijski specifičan softver. Sastavljeno od kodova, programa podrške, kompajlera, biblioteka i aplikacijskih programskih sučelja (*Application programming interface – API*), čini univerzalnu softversku okolinu čija je funkcionalnost i namjena olakšati razvoj softverskih aplikacija, proizvoda i rješenja.

Razvojno okruženje odlikuju značajna svojstva koja ga razlikuju od standardnih biblioteka:

- **zadano ponašanje**: okruženje se prije prilagodbe ponaša na način da odražava korisnikove radnje
- **inverzija kontrole**: tok kontrole programa kod razvojnog okruženja nije određen od strane korisnika već od strane okruženja
- **mogućnost proširenja**: kako bi pružio specifičnu funkcionalnost programer može proširiti razvojno okruženje nadopunjujući zadani kod sa specijaliziranim kodom
- **neizmjenjiv kod**: korisnik može proširiti, ali ne bi trebao mijenjati kod razvojnog okruženja [27].

Razvojno okruženje pruža standardni način izgradnje i implementacije aplikacija, no njegova svrha je pojednostaviti razvojnu okolinu, dopuštajući programeru da se, umjesto pisanja repetitivnih metoda i savladavanja sintakse jezika na kojem se ono temelji, posveti zahtjevima samog projekta.

1.2 Prednosti i mane

Eksponencijalnim razvojem tehnologije nastaje sve veća potreba za ubrzanim razvojem novih aplikacija i softvera te brojem razvojnih programera. Puno softverskog koda je *open-source* – dostupan javnosti na uvid, korištenje, izmjene i daljnje razvijanje. Osim toga, u programiranju se redovito koriste i *boilerplate*-i, odsječci koda koji se učestalo pojavljuju na puno mjesta, u izvornom obliku ili uz minimalno prepravaka. Razvoj je olakšan jer se uzimaju gotovi blokovi koda, umjesto da se sve piše od početka. Također, koristi se automatsko ispravljanje grešaka kompajlerom te napredni IDE-i *(Integrated Development Enviroment)*, softverske aplikacije za razvoj koje sadrže programe za pronalaženje programskih pogrešaka. Sve to dovodi nas do zaključka da se pažnja i količina znanja, s detaljnog poznavanja sintakse i savladavanja implementacije jezika, preusmjerava na znanje korištenja razvojnih okruženja te poznavanje i pozivanje već postojećih API-ja.

Zbog tog ubrzanog razvoja, često se od programera očekuje najbrže rješenje, a upravo je ideja razvojnog okruženja da mu to omogući. Zadaci koji bi se inače rješavali satima sada se mogu izvršiti za nekoliko minuta s unaprijed izgrađenim funkcijama. Najpopularnije strukture su besplatne, a uzevši u obzir da razvojnom programeru to uvelike ubrzava vrijeme programiranja, trošak za konačnog klijenta zasigurno će biti manji. Osnovno načelo razvojnih okruženja je: ne treba ponovo otkrivati toplu vodu. Kao primjer, okruženje će programeru osloboditi 2 ili 3 dana koje bi morao provesti stvarajući obrazac za provjeru autentičnosti – zadatak koji je uobičajen i nespecifičan za aplikaciju, a vrijeme koje je uštedio programer može posvetiti specifičnijim komponentama.

Razmotrimo još neke prednosti. Često kada razvojni tim krene izrađivati softver, zbog manjka vremena događa se i manjak organizacije koda te dolazi do toga da je samo taj tim u mogućnosti održavati i nadograđivati aplikaciju s lakoćom. S druge strane, struktura koju razvojno okruženje osigurava za aplikaciju omogućuje svakom programeru, bilo da je sudjelovao u njezinom razvoju ili ne, sposobnost da lako usvoji, nadogradi i održava aplikaciju kada god je to potrebno. Dakle, razvojno okruženje pruža sigurnost razvijanja aplikacije koja će, zahvaljujući tom okruženju, biti strukturirana i usklađena s poslovnim pravilima.

Što je razvojno okruženje više rasprostranjeno, to je i zajednica njegovih korisnika veća. Opsežno korišteno okruženje na taj način cijelo vrijeme poboljšava svoje sigurnosne implementacije, jer testna skupina u ovom slučaju nije nekolicina korisnika, već cijela zajednica korisnika okruženja, kojoj je isto tako cilj osigurati što bolju sigurnost za svoje aplikacije i podatke. Kao i drugi distribuirani alati, razvojno okruženje obično uključuje dokumentaciju, grupu podrške i online forum, na kojem se od zajednice i samog razvojnog tima okruženja mogu dobiti brzi odgovori. Na

istim forumima korisnici mogu prijaviti sigurnosne nedostatke koji su uočeni, koje tim zatim popravlja ili daje alternativna rješenja. Na taj način korisnici pridonose sigurnosti razvojnog okruženja jer se ono cijelo vrijeme nadograđuje i unaprjeđuje, što rezultira čvrstim sigurnosnim implementacijama.

Razvojna okruženja su u današnjici glavni koncept razvoja objektno orijentiranih softvera. Obećavaju povećanje produktivnosti, kraće vrijeme razvoja i veću kvalitetu aplikacija. Mnogo primjera pokazuju nam da se ti ciljevi mogu postići, ali možemo uvidjeti i neke mane.

Zbog složenosti njihovih API-ja, namjeravano smanjenje ukupnog vremena razvoja ne može se postići jer dolazi do potrebe za dodatnim vremenom učenja za korištenje razvojnog okruženja. Vrijeme uloženo za učenje korištenja razvojnog okruženja isplativo je ukoliko će se to razvojno okruženje opetovano koristiti u naknadnim poslovnim zadacima. U suprotnom, to vrijeme može biti skuplje i duže od vremena za prenamjenu sličnog koda ili *boilerplate*-a s kakvim je projektni tim već upoznat. Isto vrijedi i za stadij nadogradnje aplikacije. Naime, programer koji bi trebao nadograditi aplikaciju to će napraviti s lakoćom jedino ako je već upoznat s tim razvojnim okruženjem, inače će osim vremena za upoznavanje s aplikacijom potrošiti i vrijeme za upoznavanje s razvojnim okruženjem. Nadalje, jasno je da upoznavajući se s razvojnim okruženjem, fokus na sam jezik izostaje. Koristeći okruženja bez temeljnog znanja o programskom jeziku, produbljuje se znanje o okruženju, no znanje o jeziku ostaje površinsko, što postaje mana u trenutku kada kod treba nadograditi nečime što okruženje nema osigurano.

Imajući na umu sve mogućnosti i potrebe koje će korisnik možda htjeti implementirati, u razvojna okruženja ugrađuju se velike zalihe koda, no vjerojatnost da će se za razvoj softvera koristiti sve funkcije koje razvojno okruženje uključuje je vrlo mala. Osim toga, događa se da zbog potrebe da se aplikacija prilagodi željama korisnika, konkurentna okruženja, kao i okruženja koja se nadopunjuju, ponekad budu zajedno upakirana u jedan proizvod. Zbog svega ovoga jedna od najvećih mana razvojnih okruženja je preuveličanje brojnosti linija koda i veličine programa, popularno nazvana terminom ***code bloat***.

Sve ove prednosti i mane potiču korisnika da prije odabira dobro razmisli hoće li pisati sam svoj kod ili će koristiti okruženje. U slučaju odabira okruženja potrebno je, također, dobro se informirati i odlučiti koje okruženje najviše odgovara korisnikovim potrebama.

1.3 PHP razvojna okruženja

O PHP-u

PHP je interpretirani skriptni jezik, kojeg je Rasmus Lerdorf stvorio 1995., primarno za razvoj dinamičnih internet stranica. Danas mu je fokus gotovo isključivo na serverskoj strani web-programiranja, no koristi se i kao programski jezik opće namjene.

Kratica PHP izvorno je stajala za osobnu početnu stranicu (*Personal Home Page*), no sada se radi o rekurzivnom akronimu *PHP: Hypertext Preprocessor*. Kroz njegovu široku upotrebu i velik broj korisnika, uočeni su problemi u dotadašnjem pristupu pisanja PHP aplikacija. PHP stranica je poput obične HTML[1] stranice, uz dodatne PHP naredbe koje se miješaju s HTML naredbama. Osim toga, obrađivanje korisničkog ulaza je isprepleteno sa samom logikom aplikacije, što sve zajedno vodi teškom snalaženju u kodu, poteškoćama u dodavanju novih funkcionalnosti i održavanju koda.

Razdvajanje funkcionalnosti

Glavni cilj bilo je postići da više programera može nezavisno raditi na aplikaciji. Da bi se to ostvarilo bilo je potrebno:

- odvojiti HTML gotovo potpuno od PHP-a, jer često web-dizajneri koji rade na izgledu aplikacije (HTML + CSS[2]) ne znaju dobro PHP i obratno,
- osigurati potpuno odvajanje logike aplikacije od obrade korisničkog ulaza.

Pojavio se MVC (*Model-View-Controller*) arhitekturalni obrazac namijenjen za implementaciju korisničkog sučelja, prilagođen i za protok podataka u web aplikacijama, čija je svrha upravo odvojiti pojedine dijelove aplikacije u komponente, ovisno o njihovoj namjeni. Na taj način omogućen je istovremen razvoj od više programera na proširivanju funkcionalnosti aplikacije te se u zajednici, u svrhu bržeg razvoja web aplikacija, počinju razvijati mnoga PHP razvojna okruženja bazirana na MVC paradigmi.

[1] HTML (eng. HyperText Markup Language), je prezentacijski jezik kojim se opisuje sadržaj i struktura web stranice.

[2] CSS (eng. Cascading Style Sheets) je stilski jezik za opis prezentacije dokumenta napisanog pomoću HTML jezika. Najčešće se koristi za postavljanje vizualnog stila web stranica i korisničkih sučelja.

Značajke PHP okruženja

Svoju popularnost današnja PHP okruženja temelje na sljedećim prednostima:

- MVC paradigma osigurava brz razvoj
- PHP okruženja pružaju CRUD (*Create, Read, Update, and Delete*) operacije, pa programer više ne mora pisati složene upite za preuzimanje podataka iz baze
- omogućuju rast i širenje tijekom vremena jer okruženje osigurava skalabilnost sustava
- okruženja se brinu za osiguranje web aplikacije od čestih sigurnosnih prijetnji
- ne ponavljaj se (*DRY – don't repeat yourself*) načelo osigurava da minimalno koda ima maksimalan utjecaj.

Kako odabrati PHP okruženje

Kao što je već zaključeno, korisnik mora paziti da odabere ono okruženje koje najviše odgovara njegovim potrebama. Prije samog odabira, potrebno je razmotriti nekoliko glavnih pitanja. Je li projekt namijenjen maloj, određenoj skupini korisnika bez intencije širenja, ili pak ima potencijala postati proširen među mnogo korisnika. Sukladno odgovoru, gleda se koliko je skalabilnost osigurana u različitim okruženjima. Koje su značajke i funkcionalnost okruženja? (Nudi li ono što korisniku treba?) Srž ponašanja okruženja ne može se mijenjati, što znači da je korisnik prisiljen raditi na predviđen način i poštivati ograničenja okruženja.

Korisnik koji nije stručnjak za programiranje u PHP-u uvijek bi se trebao odlučiti za upotrebu popularnog okruženja s dovoljno podrške i aktivnom korisničkom bazom. Uz to, bitno je uzeti u obzir i razvija li se i održava okruženje i dalje aktivno od glavnog tima. Postoje mnoga okruženja koja imaju malo ili nimalo podrške, te ona koje su stvorili pojedinci s ograničenim znanjem o PHP-u. Takve vrste okruženja mogu uzrokovati da korisnikovi programi ne funkcioniraju ispravno, ili u još lošijem scenariju može uzrokovati velike sigurnosne probleme s korisnikovom web stranicom. U suprotnome, popularna okruženja imaju veliku zajednicu podrške koje korisniku mogu dati dosta brze odgovore na sve nedoumice, i početnička pitanja.

Zadnja stvar koju se korisnik treba zapitati je i kakva je krivulja učenja odabranog okruženja, odnosno odgovara li mu brzina napredovanja u stjecanju iskustva u datom okruženju [8, 22, 17].

Poglavlje 2

Laravel

Laravel je besplatno, *open-source* PHP razvojno okruženje zamišljeno za razvoj web aplikacija na temelju MVC (*Model-View-Controller*) arhitekturalnog obrasca. Laravelov slogan je "PHP framework for Web Artisans"[1], gdje riječ Artisan stoji za vještog zanatskog radnika koji stvara funkcionalne i dekorativne stvari, te kroz godine razvija svoje vještine do umjetničke razine. Laravel, kroz svoju izražajnu i elegantnu sintaksu, pokušava izostaviti naporne dijelove razvoja, olakšavajući uobičajene zadatke koji se koriste u većini web projekata. "Vjerujemo da razvoj mora biti užitak, a kreativan doživljaj istinski ispunjujuć" stoji u Laravel dokumentaciji. Cijeli izvorni kod Laravela nalazi se na GitHub-u pod MIT licencom [12].

2.1 Povijest Laravela

Od svih programskih jezika, upravo je PHP-u bilo potrebno kvalitetno okruženje. Naime, PHP je dinamički jezik, što znači da se programeru lako može dogoditi da mu se u pisanju koda potkrade greška, no programski jezik ga na to neće upozoriti. Greška će se javiti tek u fazi pokretanja programa, što će programera prisiliti na pregledavanje cijelog, već napisanog koda i traženje problema. S druge strane, u drugim programskim jezicima, kod se piše tako da se greške mogu identificirati prije pokretanja programa. U tim će slučajevima kompajler prepoznati grešku, bilo u fazi korištenja koda ili kompajliranja, te će programera upozoriti na nužne popravke kako bi program mogao funkcionirati.

Prije stvaranja Laravela PHP zajednica bila je zbrka konkurentnih okruženja. U to vrijeme kreatori raznih okruženja međusobno su raspravljali i nadmetali se u tome čije će okruženje bolje izvršiti neki zadatak. Takvo stanje nije bilo plodno tlo za

[1] hrv. PHP okruženje za web zanatlije

ujednačeni i progresivni rast i napredak bilo kojeg od postojećih PHP okruženja. Programeri su bili razjedinjeni.

Taylor Otwell, .net razvojni programer iz Arkansasa koristio se CodeIgniter-om, jednim od tada najpopularnijih PHP okruženja i nije bio zadovoljan zbog nedostatka jednostavnosti i fleksibilnosti. Taylor je želio riješiti ove probleme koristeći uvođenjem jasne sintakse i strukture te istovremeno sve temeljito dokumentirati. Također, Taylor je namjeravao na efikasan način riješiti probleme koji su nastajali iz dinamičke prirode programiranja u PHP-u. Pri tome je, koristeći se svojim .net iskustvom, iskoristio ideje .net infrastrukture, na istraživanje kojih je Microsoft utrošio stotine milijuna dolara. Taylor je u konačnici želio stvoriti okruženje koje će moći koristiti i najneiskusniji programeri i koje će svim korisnicima iskustvo stvaranja web aplikacija učiniti zabavnim. Sve ove Taylorove zamisli o tome kako bi okruženje trebalo izgledati i funkcionirati pretočene su u potpuno novo okruženje, danas poznato kao Laravel.

Taylor je započeo s jednostavnim upravljačkim slojem. Prva beta verzija Laravela bila je dostupna 9. lipnja 2011., a kasnije istog mjeseca uslijedila je i službena Laravel 1 verzija. Laravel 1 uključivao je ugrađenu podršku za autentifikaciju, lokalizaciju, modele, poglede, sesije, usmjeravanje i druge mehanizme, no nedostajala mu je podrška za kontrolere, što je spriječilo da to bude pravo MVC okruženje.

Laravel 2 objavljen je u rujnu 2011. donoseći mnoga poboljšanja kako od autora, tako i od zajednice. Glavne nove značajke uključivale su inverziju kontrole, sustav obrazaca nazvan Blade i podršku za kontrolere, što je učinilo Laravel 2 potpuno kompatibilnim s MVC standardom. Kao nedostatak, ukinuta je podrška za pakete ostalih proizvođača.

Laravel 3 objavljen je u veljači 2012. sa skupinom novih poboljšanja, uključujući sučelje komandne linije (Command-line interface, CLI) nazvano Artisan, ugrađenu podršku za više sustava upravljanja bazama podataka, migraciju baza podataka, podršku za rukovanje događajima i sustav pakiranja naziva Bundles.

Laravel 4, kodnog naziva Illuminate objavljen je u svibnju 2013. Ova je verzija Laravela napisana u cijelosti iznova na način da su njezini elementi prebačeni u skup zasebnih paketa distribuiranih putem Composer-a. Composer je paketni menadžer na razini aplikacije. Prije Composer-a nije bilo moguće uzeti dva odvojena paketa i zajedno koristiti različite segmente tih paketa da bi se dobilo jedno rješenje. Međutim, Composer je to omogućio i time postao ključna točka razvoja u kojoj su autori okruženja uvidjeli korist od suradnje, umjesto nadmetanja. Druga poboljšanja uključuju inicijalno punjenje baze podataka (eng. database seeding), podršku za redove poruka (eng. message queue), rad s e-poštom i podršku za odgođeno brisanje zapisa u bazama podataka, nazvano soft deletion.

Laravel 5 objavljen je u veljači 2015. sadržavajući razna nova poboljšanja. Nove

značajke ove inačice Laravela uključuju podršku za zakazivanje periodično izvršenih zadataka kroz paket nazvan Scheduler, abstracijski sloj nazvan Flysystem koji omogućuje daljinsko pohranjivanje na isti način kao i lokalni datotečni sustavi, poboljšano rukovanje paketima kroz Elixir i pojednostavljeno provjeravanje autentičnosti putem dodatnog Socialite paketa. Laravel 5 je, također, uveo novu strukturu stabla unutarnjeg direktorija za razvijene aplikacije.

Objavljene su i poboljšane verzije Laravela 5 i to Laravel 5.1. do 5.5., koje su donijele daljnja poboljšanja, poput dugotrajne korisničke podrške, poboljšanja brzine rada, automatskih fasada i mnoga druga. Posljednja verzija je Laravel 5.5., objavljena u kolovozu 2017. [15, 18].

2.2 Usporedba Laravela s drugim razvojnim okruženjima

Svako PHP razvojno okruženje sukladno svojoj osnovnoj ideji olakšava izrađivanje web stranica i web aplikacija. Međutim, svako od njih ima svoje specifičnosti i prednosti.

Primjerice, Symfony i Laravel imaju predodređene alate za generiranje predložaka, dok Yii takav alat nema, što ga čini fleksibilnijim jer omogućuje programeru da sam odabere alat koji mu najbolje odgovara. S druge strane, Symfony funkcionira na temelju ponovo iskoristivih komponenti i omogućava najbolju modularnost, dok Laravel i Yii koriste MVC obrazac. Nadalje, Yii je bez premca najbrže PHP okruženje, Laravel nudi materijale pomoću kojih se njegov rad može ubrzati, dok Symfony 2 nudi najbolju podršku glede baza podataka. Okruženja se mogu i proširivati, a u tom području Laravel nudi najviše potencijala s velikim brojem paketa koji se za to mogu iskoristiti.

Bez obzira na navedene specifičnosti, posebnu važnost pri usporedbi razvojnih okruženja treba posvetiti podršci programerske zajednice korisnika, kao što je spomenuto u sekciji 1.3 „Kako odabrati PHP okruženje“. Naime, ovaj je faktor ključan za dugovječnost, iskoristivost i napredak aplikacije.

U današnjem svijetu PHP okruženja, Laravel je apsolutni pobjednik po ovom pitanju. U relativno kratkom životnom vijeku, oko ovog razvojnog okruženja razvila se napredna i gotovo fanatična zajednica korisnika, koja je predvođena nekim od najutjecajnijih razvojnih programera u čitavoj PHP zajednici.

Jedan od njih je predavač svjetske klase, Jeffrey Way, tvorac web stranice Laracasts na kojoj se nalaze brojni izvori koji omogućuju trening u Laravel okruženju. Kako bi omogućio permanentno obrazovanje Laravelovih korisnika, Jeffrey se pobrinuo da se Laracasts stalno puni novim sadržajima. Iako su neki sadržaji naplatni,

radi se najčešće o onima koji prenose viši stupanj znanja, pa tako stranica i dalje obiluje velikim brojem besplatnih sadržaja za početnike.

Opisani razvoj ove zajednice korisnika ne bi bio moguć da Laravel nema besprijekornu dokumentaciju. Zahvaljujući tome, korisnici mogu lako doći do odgovora na većinu svojih pitanja u službenoj dokumentaciji razvojnog okruženja, a za sve ostalo mogu potražiti podršku na službenom forumu ili web stranicama poput Laracastsa. Upravo zbog toga Laravel je idealan za nove i neiskusne korisnike, no i za one s višim stupnjem znanja, jer je jednostavan, intuitivan i fleksibilan.

Konačno, snažan motiv da zajednica korisnika nastavi rasti i razvijati se je dugoročna podrška. Naime, Laravel je u svijetu razvojnih okruženja ostvario još jedan presedan, a to je zajamčena podrška u otklanjanju grešaka u trajanju dvije godine te sigurnosne nadogradnje u trajanju čak 3 godine od objave pojedine verzije Laravela. Ovime se Laravel zaokružuje kao potpuni proizvod koji stoji na raspolaganju svakome tko poželi razvijati web stranicu ili aplikaciju u PHP-u na jednostavan i učinkovit način [29, 8, 17].

Poglavlje 3

Značajke Laravela

Brz razvoj i konstantno unaprjeđenje Laravela osvojilo je mnoge programere diljem svijeta, te je početkom 2014. i 2015. godine Laravel proglašen najpopularnijim PHP okruženjem među programerima (Sitepoint online istraživanje [24, 25]).

Pažnju među programerima stekao je svojom pristupačnošću, koju ostvaruje kroz brojne značajke. Među njima su modularni sustav pakiranja s namjenskim upraviteljem ovisnosti[1], različiti načini pristupa relacijskim bazama podataka, alati koji pomažu u implementaciji i održavanju aplikacija. Daljnjim razvitkom popularnost ne izostaje jer ga se i u 2017. godini navodi kao najraširenije i najbolje PHP okruženje (Medium izvješće [28]).

U nastavku se nalazi opis i pobliže objašnjenje češće korištenih značajki u primjeni razvoja aplikacije.

3.1 Homestead

Jedna od uobičajenih početničkih pogrešaka kod prvog susreta s razvojnim okruženjem je da korisnik ne osigura kompatibilnost baze podataka i web poslužitelja sa zahtjevima okruženja.

Čak iako je korisnik stručnjak za PHP, trebao bi proći kroz dokumentaciju okruženja, kako bi potvrdio kompatibilnost, prije nego što isproba samo okruženje.

Laravel ima općenito stroži skup zahtjeva na poslužitelju od tipičnih PHP okruženja i projekata. No, to nije loša stvar, to znači samo da Laravel koristi više suvremenih, modernih značajki i da može napraviti više. Laravel koristi Composer za upravljanje ovisnostima, pa je potrebno prije korištenja Laravela instalirati i Composer na računalu.

[1] eng. *packaging system with a dedicated dependency manager*

Za instalaciju Laravela poslužitelj mora zadovoljavati sljedeće zahtjeve:

- PHP verzija $\geq$ 7.0.0
- OpenSSL PHP ekstenzija
- PDO PHP ekstenzija
- Mbstring PHP ekstenzija
- Tokenizer PHP ekstenzija
- XML PHP ekstenzija.

No, kako bi doskočio svim ovim zahtjevima i pokazao svoju elegantnost i jednostavnost Laravel pruža korisniku Homestead.

Homestead je službena unaprijed pakirana Vagrant[2] okolina koja korisniku pruža gotov razvojni okoliš sa svim alatima potrebnim za razvoj Laravel aplikacije, bez potrebe za instaliranjem PHP-a, web poslužitelja i ostalih poslužiteljskih softvera na lokalnom računalu. Može se pokrenuti na operativnim sustavima Windows, Mac OS i Linux, a od softvera, među ostalim, uključuje i:

- Ubuntu 16.04
- Git
- Nginx web poslužitelj
- PHP: PHP 7.2, PHP 7.1, PHP 7.0, PHP 5.6
- relacijske sustave upravljanja bazom podataka: MySQL, MariaDB, Sqlite3, PostgreSQL
- Composer
- Front-End[4] alate Node (uključuje Yarn, Bower, Grunt i Gulp)
- alate za cache: Redis i Memcached.

Instalacija i pokretanje projekta u Homestead okolini

Prije samog pokretanja Homestead okoline, potrebno je instalirati softver za pokretanje virtualnih mašina te Vagrant, s obzirom da je Homestead izrađen za Vagrant. Zatim treba dodati `laravel/homestead` okolinu Vagrant instalaciji koristeći jednostavnu naredbu u komandnoj liniji:

[2]Vagrant je open-source softverski proizvod za izgradnju i održavanje prijenosnih virtualnih okruženja[3] za razvoj softvera.

[3]Virtualno okruženje ili virtualni stroj je imitacija računalnog sustava. Virtualni strojevi temelje se na računalnim arhitekturama i pružaju funkcionalnost fizičkog računala. Njihova implementacija može uključivati specijalizirani hardver, softver ili kombinaciju.

[4]Izraz Front-End se odnosi na "prednji dio" sustava, odnosno, prezentacijski sloj aplikacije.

```
vagrant box add laravel/homestead
```

Sljedeći korak je kloniranje službenog Laravel repozitorija na računalo. Budući da Homestead zapravo djeluje kao upravitelj i za sve ostale projekte, njegov direktorij bi bilo dobro staviti negdje izvan direktorija svih projekata.

```
git clone https://github.com/laravel/homestead.git /Homestead
```

Idući korak je podesiti Homestead.yaml datoteku.

```
ip: "192.168.10.10"
memory: 2048
cpus: 1
provider: virtualbox
authorize: ~/.ssh/id_rsa.pub

keys:
    - ~/.ssh/id_rsa

folders:
    - map: ~/Code
      to: /home/vagrant/Code

sites:
    - map: homey.test
      to: /home/vagrant/Code/homey/public

databases:
    - homestead
```

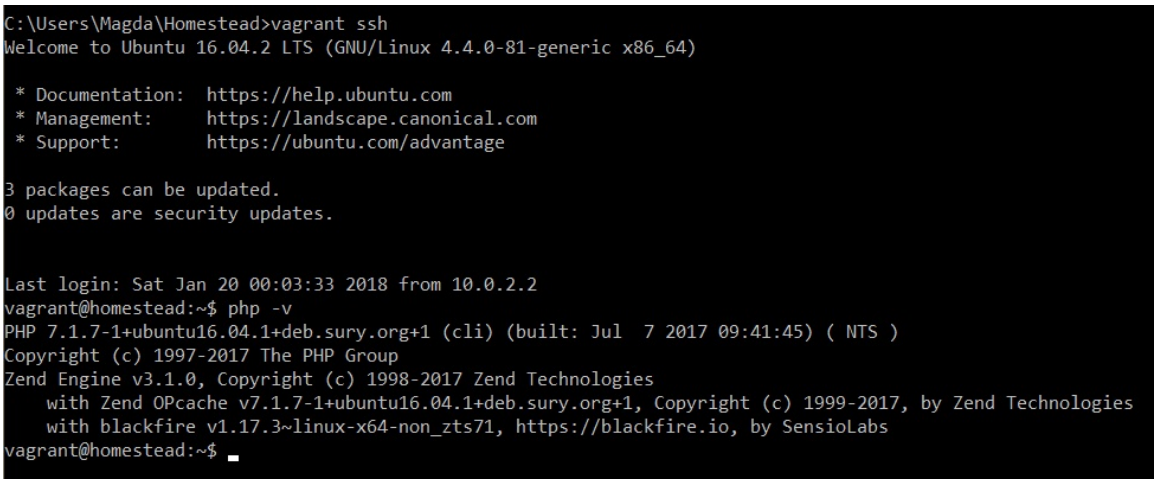
Svojstvo `provider` postavljamo na softver za pokretanje virtualne mašine koju smo instalirali. Svojstvo `authorize` postavljamo na put do našeg javnog ključa, a svojstvo `keys` postavljamo na put do našeg privatnog ključa na računalu. U svojstvu `folders` navodimo sve direktorije koje želimo podijeliti s Homestead okolinom. Prvi redak odnosi se na lokalni direktorij na našem računalu, a drugi redak pokazuje gdje će to biti na virtualnom serveru. Dok mijenjamo datoteke unutar tih direktorija, one će se sinkronizirati između našeg lokalnog računala i Homestead okoline. Unutar tih direktorija će stajati naši projekti. Svojstvo `sites` omogućuje jednostavno mapiranje domene u direktorij u Homestead okolini.

Sada ime koje smo odabrali kao domenu u datoteci Homestead.yaml moramo dodati u `hosts` datoteku na računalu. `hosts` datoteka preusmjerit će zahtjev za našu

Homestead web stranicu u našu Homestead okolinu. Redak koji dodamo ovoj datoteci izgledati će kao sljedeći:

```
192.168.10.10  homey.test
```

Sada je sve postavljeno. Idući korak je pokretanje virtualne okoline i stvaranje novog projekta. Iz Homestead direktorija pokrenimo naredbu `vagrant up`. Vagrant će dignuti virtualni stroj i automatski konfigurirati dijeljene direktorije i web stranice. Naredbom `vagrant ssh` možemo se spojiti s našom virtualnom Homestead okolinom:



Slika 3.1: Spajanje s virtualnom okolinom

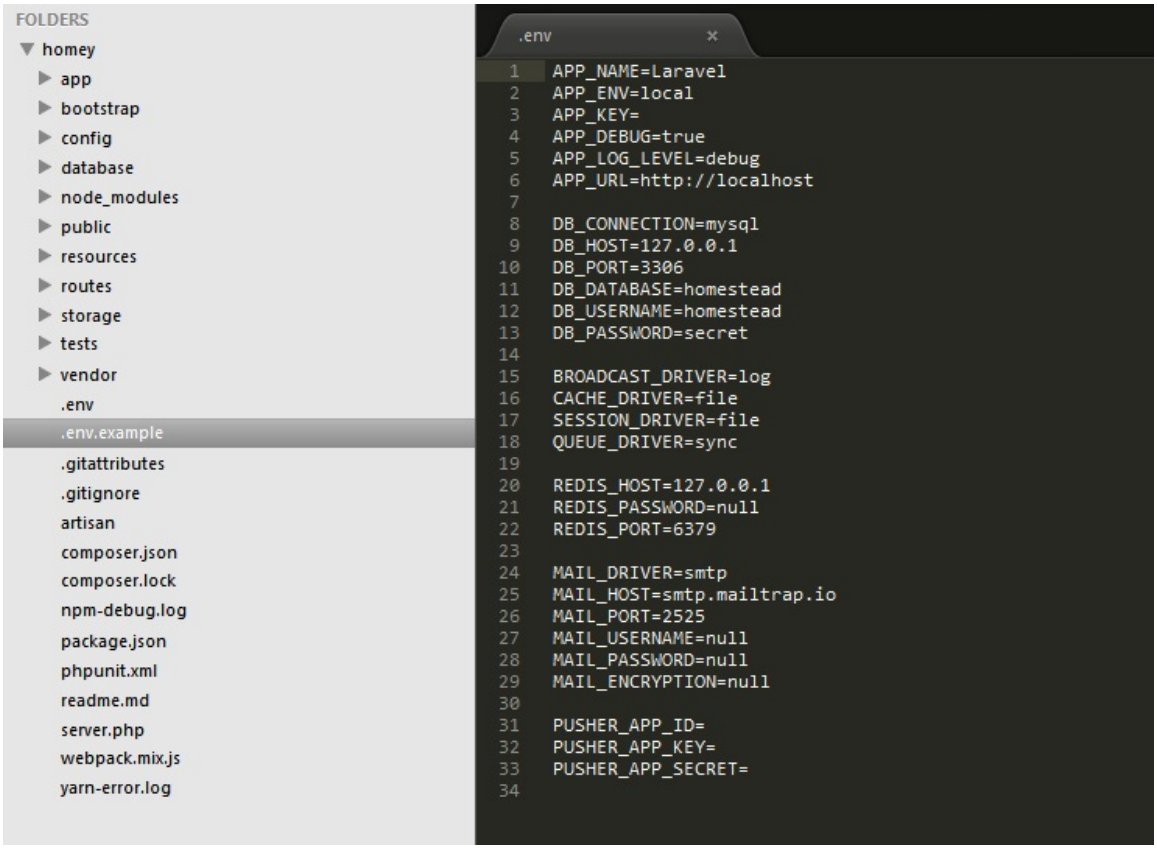
Novi projekt izradit ćemo pomoću Composer-a, u direktoriju koji smo namijenili za projekte, intuitivnom naredbom `composer create-project laravel/laravel ime-projekta`

```
cd Code
composer create-project laravel/laravel homey
```

Ovo će kopirati Laravel i sve njegove ovisnosti u direktorij novog projekta (Slika 3.2).

Sada je projekt kreiran te se našoj novoj web stranici može pristupiti putem web-preglednika na adresi `http://homey.test`

Ponekad možda želite podijeliti ono što trenutno radite s klijentom, ili među projektnim timom. Jednostavnost dijeljenja okoline osigurana je Homestead naredbom `share domena-projekta`, npr. `share homey.test`. Nakon pokretanja naredbe prikazat će se ekran koji prikazuje zapis aktivnosti i javno dostupne URL-ove za zajedničku web lokaciju.



Slika 3.2: Direktorij novokreiranog projekta

Homestead je jednostavan za male timove, prikladan je za projekte male do srednje veličine. On ne ostavlja prostor za greške Vagranta prouzročene različitim verzijama i različitim operacijskim sustavima. Iz primjera smo vidjeli da ga je lako podesiti, pokreće se izuzetno brzo i dovoljan je da svatko počne raditi s Laravelom u par minuta [13, 4, 19, 23].

3.2 Artisan

Kako bi se korisniku olakšala gradnja aplikacije, u Laravel je uključeno sučelje komandne linije zvano Artisan, koje pruža brojne korisne naredbe. Svaka naredba povezana je s nekom funkcionalnošću koja pojednostavljuje upravljanje aplikacijom. Uobičajene upotrebe Artisan-a uključuju upravljanje migracijama baze podataka, objavljivanje svojstva paketa i generiranje *boilerplate* koda za nove kontrolere i mi-

gracije. Na taj način programer je oslobođen od stvaranja odgovarajućih kostura kodova. Za popis svih dostupnih Artisan naredbi može se koristiti sljedeća naredba:

```
php artisan list
```

Svaka naredba uključuje i "help" zaslon koji prikazuje i opisuje raspoložive argumente i opcije naredbe. Za zaslon pomoći treba jednostavno ispred imena naredbe staviti `help`:

```
php artisan help down
```

```
vagrant@homestead:~/Code/homey$ php artisan help down
Usage:
  down [options]

Options:
      --message[=MESSAGE]  The message for the maintenance mode.
      --retry[=RETRY]      The number of seconds after which the request may be retried.
  -h, --help               Display this help message
  -q, --quiet              Do not output any message
  -V, --version            Display this application version
      --ansi               Force ANSI output
      --no-ansi            Disable ANSI output
  -n, --no-interaction     Do not ask any interactive question
      --env[=ENV]          The environment the command should run under
  -v|vv|vvv, --verbose     Increase the verbosity of messages: 1 for normal output, 2 for more
verbose output and 3 for debug

Help:
  Put the application into maintenance mode
vagrant@homestead:~/Code/homey$ php artisan down
Application is now in maintenance mode.
vagrant@homestead:~/Code/homey$ php artisan up
Application is now live.
```

Slika 3.3: Zaslon pomoći naredbe `down`

Sa slike 3.3 možemo vidjeti da Artisan naredba `down` postavlja aplikaciju u način za održavanje[5]. Kada korisnik ažurira aplikaciju ili izvršava održavanje, Artisan olakšava "onemogućavanje" te aplikacije, na način da će se za svaki zahtjev u aplikaciji vratiti "MaintenanceModeException" iznimka sa statusnim kodom 503, te će preusmjeriti korisnika koji je postavio zahtjev na stranicu s porukom "Be right back"[6]. Lako pokretanje moda za održavanje jedna je od funkcionalnosti koje pruža Artisan [11].

[5]Put the application into manintenance mode (hrv. postavi aplikaciju u mod za održavanje)

[6]hrv. Odmah se vraćam

Funkcionalnost i mogućnosti Artisan-a mogu se proširiti kreiranjem novih prilagođenih naredbi koje se, primjerice, mogu koristiti za automatizaciju ponavljajućih zadataka specifičnih za pojedine aplikacije.

3.3 Blade

U poglavlju 1.3 spomenuto je da je kod u kojem su ispremiješani PHP i HTML težak za snalaženje i nepraktičan za mijenjanje. No ta dva jezika nije moguće u potpunosti odvojiti s obzirom da se na web stranici moraju prikazivati nekakvi podatci, često dohvaćeni iz baze. Alat koji Laravel isporučuje za rješenje tog problema zove se Blade. Pomoću njega se na elegantan način PHP kod sažima s HTML kodom. Upravo zbog te funkcionalnosti, moglo bi se reći da je to jedna od najboljih značajki okruženja.

Blade je jednostavan, ali moćan alat za generiranje predložaka koji, za razliku od drugih popularnih PHP alata za generiranje predložaka, ne sprječava korištenje običnog PHP koda u pogledima[7]. Čak štoviše, svi Bladeovi pogledi kompajliraju se u običan PHP kod i spremaju se dok ne budu izmijenjeni. To znači da Blade uopće nema loš utjecaj na performanse aplikacije (eng. zero overhead), a osigurava "čistu" sintaksu.

Blade datoteke pogleda koriste `.blade.php` ekstenziju datoteke i obično su pohranjene u `resources/views` direktoriju.

Nasljeđivanje predložaka i sekcije

Dvije od glavnih prednosti korištenja Blade-a su nasljeđivanje predložaka i sekcije (eng. sections).

Izgled svake aplikacije možemo zamisliti kao tlocrt, odnosno raspored, budući da većina web aplikacija ima isti izgled na raznim stranicama. U tom rasporedu postoje stalni segmenti koji se ne mijenjaju, poput zaglavlja i navigacijske ploče, te segmenti koji se mijenjaju, u kojima je sama funkcionalnost aplikacije.

Kako stalne segmente ne bismo morali za svaki pogled ponovo pisati, Blade pruža nasljeđivanje predložaka, stoga je prikladno glavni izgled (eng. layout) aplikacije definirati kao jedan Blade pogled.

[7]Pogled je jedna od tri komponente MVC obrasca koja predstavlja prezentacijski sloj aplikacije. Najčešće, jedan pogled generira i predstavlja izgled jedne stranice web aplikacije. Za više informacija pogledati poglavlje 4.2.

Kod 3.1: Blade pogled glavnog izgleda

```
<!-- Spremljeno u resources/views/layouts/app.blade.php -->

<html>
    <head>
        <title>App Name - @yield('title')</title>
    </head>
    <body>
       @include('layouts.nav')
        <div class="container">
            @yield('content')
        </div>
    </body>
</html>
```

Naredbom `@yield('ime-sekcije')` definira se sekcija stranice koja će biti promjenjiva i koju će stranice, koje nasljeđuju taj izgled (eng. child page), ispuniti na svoj način. Prilikom definiranja pogleda-djeteta koristi se naredba `@extends`, kako bismo specificirali koji bi izgled taj pogled trebao naslijediti.

Naredbama `@section('ime-sekcije')` i `@endsection` definiraju se i ograđuju sadržaji kojima se proširuje taj izgled.

```
<!-- Spremljeno u resources/views/child.blade.php -->

@extends('layouts.app')

@section('title', 'Homey')

@section('content')
    <p>This is my body content.</p>
@endsection
```

Ovaj kod nasljeđuje izgled `layouts.app` te ispunjuje njegovu sekciju `title` riječju "Homey", dok sekciju `content` ispunjuje HTML paragrafom.

Uključivanje potpogleda

Blade-ova naredba `@include` omogućuje da se jedan Blade pogled uključi unutar drugog pogleda. Naredbom `@include('layouts.nav')` u kodu 3.1 uključuje se sljedeći kod za navigacijsku ploču u glavni izgled.

```
<!-- Spremljeno u resources/views/layouts/nav.blade.php -->

<nav class="navbar navbar-default">
  <div class="container-fluid">
    <div class="navbar-header">
      <a class="navbar-brand" href="{{ url('/home') }}">Homey</a>
    </div>
    <ul class="nav navbar-nav">
      <li> <a href="https://laravel.com/docs">Documentation</a></li>
      <li> <a href="https://laracasts.com">Laracasts</a></li>
      <li><a href="https://laravel-news.com">News</a></li>
      <li><a href="https://github.com/laravel/laravel">GitHub</a></li>
    </ul>
  </div>
</nav>
```

Iako će uključeni pogled naslijediti sve podatke dostupne u roditeljskom pogledu, može se, također, dodati i niz dodatnih podataka u uključeni pogled:

```
@include('ime-pogleda', ['ime-varijable' => 'podatci'])
```

Prikazivanje podataka, uvjetne naredbe i petlje

Osim nasljeđivanja predložaka, Blade preuzima ulogu prikazivanja podataka te pruža prikladne kratice za uobičajene PHP upravljačke strukture, kao što su uvjetne naredbe i petlje. Te kratice pridonose vrlo čistom načinu rada s PHP upravljačkim strukturama, bivajući istovremeno vrlo slične odgovarajućim strukturama, kako bi se korisnik što bolje snalazio. Na primjer, s uobičajenim PHP-om, da bismo prošli kroz popis ljudi i ispisali njihove nazive unutar liste, mogli bismo napisati:

```
<ul>
    <?php foreach($people as $p) : ?>
        <li><?php echo $p; ?></li>
    <?php endforeach; ?>
</ul>
```

Podatke proslijeđene u Blade pogled možemo prikazati pakirajući naziv varijable u vitičaste zagrade, dok se uvjetne naredbe, poput `if else`, i petlje, poput `foreach`, ugrađuju u kod jednostavno, dodajući na početak oznaku "`@`" ispred imena naredbe, te na kraju dodajući "`@end + ime-naredbe`", uz izostavljanje oznake za PHP kod. Na

taj način, korištenjem Bladea, prethodni kod možemo zamijeniti sljedećim:

```
<ul>
    @foreach($people as $p)
        <li>{{ $p }}</li>
    @endforeach
</ul>
```

3.4 Još neke značajke

Eloquent ORM

Eloquent ORM (Object-Relational Mapping[8]) je napredna PHP implementacija *active record* obrasca[9], koja pruža interne metode za provođenje ograničenja na relacije među objektima baze podataka. Slijedeći *active record* obrazac, Eloquent ORM predstavlja tablice baze podataka kao klase, s njihovim objektnim instancama vezanim za jedan red u tablici.

Laravel Fasade

Laravel fasade pružaju "statično" sučelje za one klase koje su dostupne u spremniku servisa. Laravel dolazi s mnogim fasadama koje omogućuju pristup gotovo svim Laravelovim značajkama, bez potrebe za pamćenjem dugih imena klasa koje se moraju ručno konfigurirati. One pružaju sažetost i izražajnu sintaksu, istodobno održavajući više testabilnosti i fleksibilnosti od tradicionalnih statičkih metoda. Sve fasade su definirane u `Illuminate\Support\Facades` prostoru imena, pa se mogu lako dohvatiti kroz cijelu aplikaciju.

CSRF

Laravel olakšava zaštitu korisnikove aplikacije od napada međustraničnim krivotvorenim zahtjevom (eng. cross-site request forgery, CSRF). CSRF je tip malicioznog iskorištavanja gdje se neovlaštene naredbe izvršavaju u ime autenticiranog korisnika.

[8] hrv. objektno-relacijsko mapiranje

[9] Active record obrazac je arhitekturalni obrazac korišten u softverima koji pohranjuju objektne podatke u relacijske baze podataka. Sučelje objekta, u skladu s ovim obrascem, obuhvaća funkcije kao što su umetanje, ažuriranje i brisanje, plus svojstva koja više ili manje odgovaraju stupcima u tablici baze podataka.

Laravel automatski generira CSRF ”token” za svaku aktivnu korisničku sesiju kojom upravlja aplikacija. Ovaj se token koristi da bi se provjerilo je li upravo autenticirani korisnik taj koji postavlja zahtjeve u aplikaciji.

Kad god korisnik definira HTML formu u svojoj aplikaciji, trebao bi uključiti skriveno CSRF token polje u obrascu, kako bi zaštitni CSRF middleware mogao provjeriti zahtjev. Korisnik se može koristiti `csrf_field` pomagačem kako bi generirao tokensko polje:

```
<form method="POST" action="/profile">
    {{ csrf_field() }}
    ...
</form>
```

`VerifyCsrfToken` middleware, koji je uključen u `web` middleware grupu, automatski će provjeriti odgovara li token pri unosu zahtjeva tokenu spremljenom u sesiji.

Poglavlje 4

Model – View – Controller

Promatrajući rani razvoj grafičkog korisničkog sučelja, Model–View–Controller (MVC) arhitekturalni obrazac, originalno namijenjen za implementaciju korisničkog sučelja (GUI), postao je jedan od prvih pristupa opisivanja i implementacije softverskih konstrukata u smislu njihovih odgovornosti. MVC arhitektura sedamdesetih je godina inicijalno uvedena u jezik Smalltalk-76, a potom implementirana unutar biblioteka verzije Smalltalk-80. Tek 1988. godine je MVC arhitektura prihvaćena kao općeniti koncept (Krasner, Pope, 1988)[1].

MVC je softverski arhitekturalni obrazac koji se koristi u softverskom inženjerstvu za odvajanje pojedinih dijelova aplikacije u komponente, ovisno o njihovoj namjeni. To je učinjeno kako bi se interni prikazi informacija odvojili od načina na koji se podaci prikazuju i prihvaćaju od korisnika. Dijeli aplikaciju na 3 osnovne komponente:

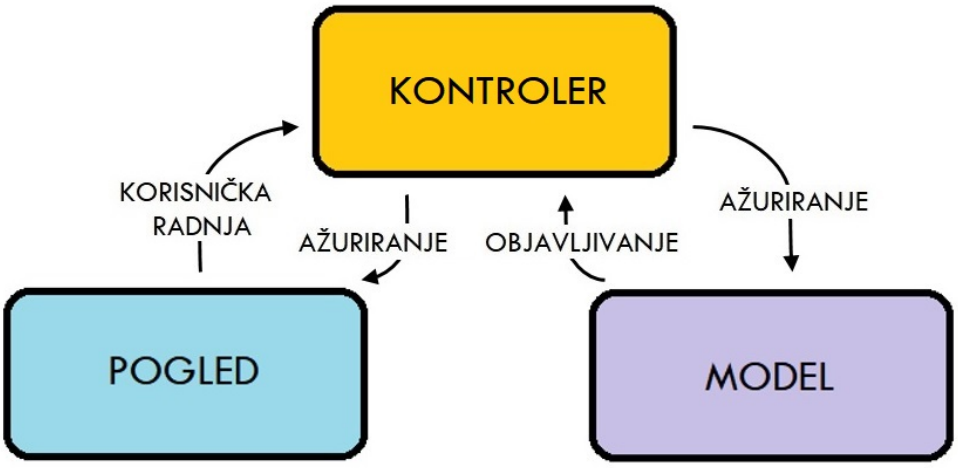
- Model (Model) — podaci i logika određene aplikacije,
- Pogled (View) — prikaz prethodno modeliranih podataka,
- Kontroler ili upravitelj (Controller) — upravlja korisničkim zahtjevima,

čime se omogućuje učinkovita upotreba koda i paralelni razvoj. Kroz MVC je dakle razdvojena obrada podataka od programiranja sučelja i zaprimanja zahtjeva korisnika. Kao i kod drugih softverskih obrazaca, MVC obrazac izražava samo jezgru rješenja problema, na taj način dopuštajući da se prilagodi svakom sustavu.

[1]Krasner, Glenn E.; Pope, Stephen T. (Aug–Sep 1988). "A cookbook for using the model–view controller user interface paradigm in Smalltalk-80". The Journal of Object Technology, SIGS Publications.

Odnos modela, pogleda i kontrolera

Proces započinje korisničkom radnjom koju analizira kontroler, a koja sadržava skup funkcija koje je potrebno izvršiti. U tom procesu kontroler komunicira s dvije komponente, modelom i pogledom. Od modela kontroler može tražiti podatke iz baze ili pak slati nalog sa zahtjevom za ažuriranje stanja modela. Model komunicira s bazom podataka, ukoliko je potrebno obrađuje podatke te po završetku objavljuje rezultat kontroleru. Kontroler zatim završava proces pozivajući komponentu pogleda da ažurira i generira zadani predložak po kojemu će se dobiveni podaci i prezentirati, te šalje taj pogled korisniku [16, 5, 3, 26].



Slika 4.1: Dijagram interakcija unutar MVC obrasca

4.1 Model

Model je središnja komponenta obrasca jer izražava ponašanje aplikacije u smislu domene problema, neovisno o korisničkom sučelju. Model implementira strukture podataka koje predstavljaju objekte s kojima se manipulira u aplikaciji (npr. User, Estate, ...). Osim toga, implementira i načine za dohvaćanje tih podataka iz baze te način za promjenu struktura. Na taj način model izravno upravlja podacima objekata koji sa svojim funkcionalnostima zapravo predstavljaju logiku i pravila web aplikacije.

Model je potpuno odvojen od korisničkog sučelja, ne sadrži HTML niti ikakav prezentacijski kod. On ne koristi GET i POST metode, odnosno, nema interakciju s korisnikom aplikacije.

Model u Laravelu

Svaka struktura podataka koju spremamo u bazu, u novu tablicu u Laravelu, ima odgovarajući Model koji se koristi za interakciju s tom tablicom. Modeli omogućuju upite za podatke u tablicama, kao i umetanje novih zapisa u tablicu. U Laravelu je to implementirano pomoću Eloquent ORM-a, u kojem je svaka klasa predstavljena kao tablica baze podataka i dodjeljen joj je Eloquent model koji proširuje `Illuminate\Database\Eloquent\Model` klasu, a instance klase su prikazane kao pojedini red u tablici.

Novi model kreiramo pomoću Artisan naredbe `make:model` na način:

```
php artisan make:model ImeModela
```

Novi modeli se zadano spremaju u `app` direktorij. Izgled novostvorenog modela je sljedeći:

```
<?php

namespace App;
use Illuminate\Database\Eloquent\Model;

class ImeModela extends Model
{
    //
}
```

Možemo primijetiti da u modelu nije specificirana tablica baze podataka s kojom je model povezan. Konvencija je da Eloquent koristi tablicu čije je ime jednako nazivu modela u množini (engleskog jezika), stoga bi ime modela trebalo biti u jednini (engleskog jezika). Kao primjer, aplikacija sadrži klasu `Estate`, te stoga Eloquent zaključuje da se zapisi pohranjuju u tablici `estates`, osim ako eksplicitno definiramo ime tablice u svojstvu `table` našeg modela:

```
protected $table = 'my_estates';
```

Također, Eloquent pretpostavlja da svaka tablica ima primarni ključ nazvan `id`, te pretpostavlja da je primaran ključ uzlazna cjelobrojna vrijednost, što znači da će svaki primarni ključ biti pretvoren u `integer` tip. Osim toga, očekuje se da u tablici postoje `created_at` i `updated_at` stupci. Kako bismo premostili ove konvencije, možemo definirati `protected $primaryKey` svojstvo i deklarirati željeni primarni ključ te `$incrementing` svojstvo postaviti na `false`. Kako bi se pokazalo da tablica nema potonje stupce, svojstvo `$timestamps` postavlja se na `false`.

Eloquent modele možemo koristiti i kao graditelje upita na tablicu baze podataka povezanu s modelom. Najčešće metode korištene u upitima su `all` i `get`, gdje `all` dohvaća sve instance nekog modela, a kad se na upite dodaju ograničenja, koristi se metoda `get` kako bi se dohvatili rezultati:

```
<?php

use App\Estate;

$estates = App\Estate::all();

foreach ($estates as $estate) {
    echo $estate->title;
}

$cheapEstates = App\Estate::where('price', '<=', 40000)
               ->orderBy('title', 'desc')
               ->take(10)
               ->get();
```

Kad se u Eloquent modelima koriste metode, poput `all` i `get`, koje dohvaćaju višestruki rezultat, vratit će se instanca `Illuminate\Database\Eloquent\Collection` klase. To je korisno jer klasa `Collection` pruža mnoštvo dodatnih metoda za korištenje dohvaćenih podataka, poput `avg()` metode koja vraća srednju vrijednost po zadanom ključu, `contains()` metode koja utvrđuje sadrži li kolekcija određenu stavku, `first`, `random`, `groupBy` i mnogih drugih [6]. Također, sve kolekcije služe i kao iteratori, omogućujući korisniku da petljom prolazi kroz njih, kao da su jednostavna PHP polja.

Eloquent pruža još i funkcionalnost vraćanja iznimke ako model nije pronađen. Metode `findOrFail` i `firstOrFail` će dohvatiti prvi rezultat upita. Ako nijedan rezultat nije pronađen, `Illuminate\Database\Eloquent\ModelNotFoundException` bit će vraćena iznimka. U slučaju da iznimka nije uhvaćena, 404 HTTP odgovor automatski se vraća korisniku. Pri korištenju ovih metoda nije potrebno pisati eksplicitne provjere za vraćanje 404 odgovora:

```
$estate = App\Estate::findOrFail(1);
$smallEstate = App\Estate::where('surface', '<', 40)->firstOrFail();
```

4.2 View

View ili pogled je prikaz korisničkog sučelja. On predstavlja sloj za vizualizaciju modela, odnosno, njegov zadatak je prikazati podatke kroz grafičko korisničko sučelje u nekom pogodnom formatu. Obično postoji mnogo pogleda u jednoj aplikaciji. Najčešće je svaka podstranica u web aplikaciji predstavljena jednim pogledom, no moguće je više puta koristiti iste poglede za neke tipične radnje poput ispisa začelja, zanožja, navigacijske ploče. Takve poglede obično smatramo potpogledima i uključujemo ih pomoću Blade-a u glavni pogled na način opisan u sekciji Uključivanje potpogleda poglavlja 3.3.

Pogled, kao ni model, ne komunicira s korisnikom, odnosno ne koristi GET i POST metode. Jednako tako, ne pristupa bazi podataka, datotekama na disku i slično, već služi samo za dohvaćanje i prikazivanje gotovih podataka. U pogledu se ne radi obrada podataka pa se PHP koristi na vrlo trivijalnoj razini – većinski za ispis varijabli ili iteraciju po polju za ispis varijabli. No, u Laravelu i tu ulogu preuzima Blade, kako bi kod bio što elegantniji i čitljiviji na način opisan u sekciji Prikazivanje podataka, uvjetne naredbe i petlje poglavlja 3.3.

Pogled je dio aplikacije vidljiv korisniku, za razliku od kontrolera i modela koji su pozadinski dio aplikacije, te se stoga on definira pomoću prezentacijskog jezika HTML, stilskog jezika CSS, te skriptnih jezika CSS i JSON koji se izvode u web-pregledniku na strani korisnika. Pogledi su spremljeni u `resources\views` direktoriju i završavaju nastavkom `blade.php`.

4.3 Controller

Kontroler djeluje kao usmjeravanje prometa između Pogleda i Modela. Zadatak kontrolera je obrađivati podatke poslane od strane korisnika i prevesti ih u radnje koje model treba poduzeti. On, dakle, odgovara na HTTP zahtjeve i pristupa `GET`, `POST` i ostalim PHP varijablama koje reprezentiraju podatke koje je poslao korisnik.

Osim toga, kontroler radi upit na modelu, te na temelju odgovora priprema podatke i odabire prikladni pogled koji će ih prikazati. Možemo ga nazvati posrednikom između modela i pogleda.

Kontroler u Laravelu

U Laravelu je uobičajeno da između kontrolera i modela postoji 1:1 korespondencija, odnosno najčešće za svaku model klasu postoji odgovarajuća kontroler klasa. To rezultira lakšim rukovanjem s modelima jer u određenom kontroleru stoje funkcije za

upravljanje podacima odgovarajućeg modela. Kontrolerima se na taj način grupira logika rješavanja povezanih korisničkih zahtjeva u zajedničku klasu.

Svi kontroleri nasljeđuju baznu klasu `Controller` i pohranjuju se u direktoriju `app/Http/Controllers/`. Artisan naredbom `make:controller` kreiramo novi kontroler za upravljanje, npr. nekretninama:

```
php artisan make:controller EstatesController
```

te na taj način dobivamo novostvoreni kontroler:

```
<?php
namespace App\Http\Controllers;

use App\Estate;
use Illuminate\Http\Request;

class EstatesController extends Controller
{
    //
}
```

Sada, u tom kontroleru, možemo implementirati `index` metodu koja po konvenciji pokazuje sve instance određenog modela, u ovom slučaju sve nekretnine, te `show` metodu koja nam pokazuje određenu nekretninu:

```
<?php
...
class EstatesController extends Controller
{
   public function index()
    {
        $estates = Estate::all();
        return view('estates.index', compact('estates'));
    }

    public function show($id)
    {
        $estate = Estate::find($id);
        return view('estates.show', compact('estate'));
    }
}
```

Proučimo što u prijašnjem kodu zapravo radi kontroler kroz metodu `show`. Kontroler prima zahtjev od korisnika te zaključuje da mora dohvatiti iz baze podataka određenu nekretninu. Stoga, kroz Estate model, on radi upit na tablicu baze podataka za određenom nekretninom (`Estate::find($id);`). Zatim preuzima tu instancu klase modela nekretnine i prosljeđuje ju prezentacijskom sloju, odnosno pogledu `estates/show.blade.php`, kako bi taj pogled mogao generirati potreban HTML kod za prikaz nekretnine korisniku. Kontroler na taj način, zapravo, upravlja modelom i pogledima.

Resursni kontroler

Kada bismo htjeli implementirati kontroler koji za nekretninu definira sve CRUD metode:

- `index` – za prikazivanje liste svih resursa
- `create` – prikazuje formu za kreiranje resursa
- `store` – sprema novokreirani resurs u bazu
- `show` – prikazuje određeni resurs
- `edit` – prikazuje formu za uređivanje podataka resursa
- `update` – ažuriraj određeni resurs u bazi
- `destroy` – izbriši određeni resurs iz baze

to bismo učili jednostavno pomoću Artisan naredbe:

```
php artisan make:controller EstatesController --resource
```

Ta naredba će kreirati kontroler u `app/Http/Controllers/EstatesController.php`, koji će sadržavati metodu za svaku od raspoloživih resursnih operacija, te ga zato nazivamo resursnim kontrolerom.

Kod novostvorenog resursnog kontrolera:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Estate;
use App\User;

class EstatesController extends Controller
{
```

```
/**
 * Display a listing of the resource.
 *
 * @return \Illuminate\Http\Response
 */
public function index()
{
    //
}

/**
 * Show the form for creating a new resource.
 *
 * @return \Illuminate\Http\Response
 */
public function create()
{
    //
}

/**
 * Store a newly created resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    //
}

/**
 * Display the specified resource
 * @param  \App\Estate  $estate
 * @return \Illuminate\Http\Response
 */
public function show(Estate $estate)
{
    //
}

/**
```

```
     * Show the form for editing the specified resource.
     * @param  \App\Estate  $estate
     * @return \Illuminate\Http\Response
     */
    public function edit(Estate $estate)
    {
        //
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \App\Estate  $estate
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, Estate $estate)
    {
        //
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  \App\Estate  $estate
     * @return \Illuminate\Http\Response
     */
    public function destroy(Estate $estate)
    {
        //
    }
}
```

Ako dodatno želimo da metode resursnog kontrolera upućuju na odgovarajući model, pri generiraju kontrolera dodajemo `--model` opciju:

```
php artisan make:controller EstatesController --resource --model=Estate
```

Na korisniku je sada samo da metode kontrolera popuni na način prilagođen njegovoj aplikaciji.

Poglavlje 5

Tok usmjeravanja

Nakon što smo shvatili kako MVC funkcionira, potrebno je ljepilo koje će povezati te tri komponente, odnosno koje će raditi usmjeravanje kroz aplikaciju na određeni kontroler ili pogled. Laravel, još jednom, i to ima riješeno. Pruža nam pomoć u usmjeravanju, u stvaranju URL-ova za aplikaciju, pri izgradnji linkova iz predložaka te pri generiranju odgovora preusmjeravanja na drugi dio aplikacije.

Za usmjeravanje u Laravelu koristi se klasa `Route`. Instancu klase `Route` zvati ćemo 'ruta'. U najjednostavnijem slučaju ruta prihvaća dva parametra: URI (*Uniform Resource Identifier*)[1] i metodu:

```
Route::get('foo', function () {
    return 'Hello World';
});
```

Sve definicije ruta pohranjene su u `routes` direktoriju, koji sadrži četiri datoteke:

- `web.php`
- `api.php`
- `console.php`
- `channels.php`

Te datoteke okruženje samostalno automatski učitava. Rutama su dodijeljene grupe, ovisno o tome koje značajke pružaju. Datoteka `web.php` sadrži rute koje su smještene u `web middleware` grupu, koja omogućuje sesije i CSRF zaštitu. Većina, a ponekad i sve rute aplikacije, biti će definirane u toj datoteci. Rutama definiranim u njoj može se pristupiti unosom URL-a određene rute u preglednik.

[1] URI je jedinstveni identifikator. Sastoji se od niza znakova koji se koristi za označavanje resursa kao što su web stranice.

Na primjer, sljedećoj ruti:

Kod 5.1: Ruta za prikaz svih nekretnina

```
Route::get('/estates', 'EstatesController@index');
```

možemo pristupiti ako u svom pregledniku unesemo `http://homey.test/estates`.

Klasa `Route` nam omogućuje registraciju ruta koje odgovaraju bilo kojoj HTTP metodi:

```
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
Route::options($uri, $callback);
```

Ponekad je potrebno registrirati rutu koja odgovara na više ili čak na sve HTTP glagole. To činimo pomoću `match` i `any` metoda:

```
Route::match(['get', 'post'], '/', function () {
    //
});

Route::any('foo', function () {
    //
});
```

Kako datoteka `web.php` ne bi postala puna implementacije funkcija na koje preusmjeravamo, klasa `Route` omogućava nam preusmjeravanja i na Blade poglede i metode kontrolera. Blade pogledi mogu biti vraćeni iz rute koristeći globalni `view` pomagač:

```
Route::get('blade', function () {
    return view('child');
});
```

5.1 Rute koje usmjeravaju na kontroler

Zamislimo formu za kreiranje novog oglasa za nekretninu, te na dnu te forme gumb `Submit`, što znači da želimo spremiti taj oglas. Ono što bi aplikacija u trenutku pritiska gumba trebala napraviti je poslati te podatke kontroleru koji upravlja nekretninama, kako bi on dao naredbu modelu da ih spremi u bazu. Kada šaljemo podatke to radimo pomoću HTTP metode POST, a forma poprima sljedeći oblik:

```
<form method="POST" action='/estates'>

...
  <div class="form-group ">
    <button type="submit" class="btn btn-primary">Submit</button>
  </div>

</form>
```

Vidimo da je metoda postavljena na POST, dok za URI gledamo akciju koja je postavljena na `/estates`. S obzirom da ruta prihvaća kao parametre URI i metodu, potrebno je napisati rutu koja će podatke preusmjeriti odgovarajućoj metodi u odgovarajućem kontroleru:

Kod 5.2: Ruta koja preusmjerava na metodu kontrolera

```
Route::metoda('uri', 'ImeKontrolera@metodaKontrolera');

//Za primjer sa spremanjem novokreirane nekretnine ruta bi izgledala ovako:
Route::post('/estates', 'EstatesController@store');
```

Ovdje je vidljivo da nismo naveli puni prostor imena (eng. namespace) kontrolera pri definiranju rute kontrolera. To je omogućeno jer klasa `RouteServiceProvider` učitava datoteke ruta unutar grupe koja sadrži prostor imena, pa se specificira dio naziva klase koji dolazi nakon `App\Http\Controllers` dijela prostora imena. Kada bismo se odlučili ugnijezditi kontrolere dublje u direktorij `App\Http\Controllers` bilo bi potrebno koristiti ime klase u odnosu na `App\Http\Controllers` korijen prostora imena. Npr. ako je puna klasa kontrolera `App\Http\Controllers\Estates\AdminController`, rute do kontrolera će se registrirati ovako:

```
Route::get('foo', 'Estates\AdminController@metoda');
```

5.2 Parametri rute

Za dohvaćanje potrebnih informacija iz URL-a, poput npr. ID-a nekretnine, ponekad je potrebno izvući određene segmente URI-ja unutar rute. To činimo definiranjem parametara rute. Parametri rute uvijek su zatvoreni unutar vitičastih zagrada i trebali bi se sastojati od abecednih znakova. Ne smiju sadržavati znak '-', ali smiju sadržavati donju crtu '_':

```
Route::get('estate/{estate_id}', 'EstatesController@show');
```

Parametri rute umeću se u povratne pozive rute/kontrolere na temelju njihovog redoslijeda, odnosno, imena argumenata nisu bitna. Možemo definirati proizvoljan broj parametara koji je potreban za rutu:

```
Route::get('user/{user}/estates/{estate}', function ($userId, $estateId) {
    //
});
```

Ako korisnik želi definirati parametar čija prisutnost je opcionalna, to se dobiva jednostavno postavljanjem upitnika nakon naziva parametra. Osim toga, potrebno je navesti i zadanu vrijednost odgovarajućeg parametra:

```
Route::get('user/{name?}', function ($name = null) {
    return $name;
});

Route::get('user/{name?}', function ($name = 'Andrej') {
    return $name;
});
```

Također, mogu se postaviti uvjeti na format parametra pomoću metode `where` koja prihvaća naziv parametra i regularni izraz koji definira ograničenje parametra.

```
Route::get('user/{name}', function ($name) {
    //
})->where('name', '[A-Za-z]+');

Route::get('user/{id}/{name}', function ($id, $name) {
    //
})->where(['id' => '[0-9]+', 'name' => '[a-z]+']);
```

Ruta kontrolera s jednom metodom

Ako kontroler pak upravlja samo jednom metodom, nju možemo nazvati `__invoke`, a u registriranju rute za takav kontroler nije potrebno specificirati metodu:

```
<?php

namespace App\Http\Controllers;
use App\User;
use App\Http\Controllers\Controller;

class ShowProfile extends Controller
{
    /**
     * Show the profile for the given user.
     * @param  int  $id
     * @return Response
     */
    public function __invoke($id)
    {
        return view('user.profile', ['user' => User::findOrFail($id)]);
    }
}

//ruta u web.php :
Route::get('user/{id}', 'ShowProfile');
```

5.3 Resursne rute

Postoji konvencija koju se najčešće poštuje, gdje određena HTTP metoda zajedno s određenim URI-jem upućuju na očekivanu akciju. Vidjeli smo u primjeru sa spremanjem novokreirane nekretnine (kod 5.2) da metoda POST uz URI `/estates` vodi akciji `store`, te iz koda 5.1 da metoda GET uz isti URI vodi akciji `index`.

Kako ne bismo morali pamtiti te konvencije, ako koristimo resursni kontroler koji implementira metode za sve HTTP glagole, možemo jednostavno registrirati resursnu rutu do kontrolera:

```
Route::resource('estates', 'EstatesController');
```

Ova jedna ruta stvara višestruke rute za upravljanje različitim akcijama na resursu.

Generirani resursni kontroler već će imati kostur metoda za svaku od tih akcija, uključujući informacije o HTTP glagolima i URI-jima koje one obrađuju:

HTTP glagol	URI	Akcija	Ime rute
GET	/estates	index	estates.index
GET	/estates/create	create	estates.create
POST	/estates	store	estates.store
GET	/estates/{estate}	show	estates.show
GET	/estates/{estate}/edit	edit	estates.edit
PUT/PATCH	/estates/{estate}	update	estates.update
DELETE	/estates/{estate}	destroy	estates.destroy

Moguće je registrirati i više resursnih kontrolera odjednom prosljeđujući polje metodi `resources`

```
Route::resources([
    'estates' => 'EstatesController'
    'photos' => 'PhotosController',
]);
```

Parcijalne resursne rute

Pri deklariranju resursne rute moguće je navesti podskup akcija koje bi kontroler trebao izvršavati, umjesto cijelog skupa zadanih akcija:

```
Route::resource('estate', 'EstatesController', ['only' => [
    'index', 'show'
]]);

Route::resource('estate', 'EstatesController', ['except' => [
    'create', 'store', 'update', 'destroy'
]]);
```

Dopunjavanje resursnog kontrolera

Ako su u resursnom kontroleru definirane neke akcije osim zadanih akcija, potrebno je dodati rute do resursnog kontrolera za te akcije. Te rute potrebno je definirati prije poziva `Route::resource`, jer inače rute definirane `resource` metodom mogu nenamjerno imati prednost nad svojim dopunskim rutama:

```
Route::get('estates/popular', 'EstatesController@method');

Route::resource('estates', 'EstatesController');
```

Pred-memoriranje ruta

Kada aplikacija koristi isključivo rute temeljene na kontrolerima, preporuča se koristiti Laravelovo pred-memoriranje ruta (eng. Route caching). Upotreba pred-memorije ruta značajno će smanjiti vrijeme potrebno za registraciju svih ruta aplikacije, u nekim slučajevima čak do sto puta. Generiranje pred-memorije ruta dobiva se izvršavanjem Artisan naredbe:

```
php artisan route:cache
```

Nakon pokretanja ove naredbe, datoteka s pred-memorijom ruta biti će učitana na svaki zahtjev. Dakako, u trenutku dodavanja novih ruta, potrebno je generirati novu pred-memoriju ruta pa je stoga preporučljivo naredbu `route:cache` izvršiti samo jednom, pri objavi aplikacije. No, postoji i naredba `route:clear` za čišćenje pred-memorije ruta:

```
php artisan route:clear
```

Poglavlje 6

Baza podataka

Laravel čini interakciju s bazama podataka iznimno jednostavnom koristeći sirovi SQL, graditelj upita i Eloquent ORM. Sadrži migracije koje su poput verzije kontrole za bazu podataka, podršku za odgođeno brisanje zapisa u bazama podataka te podržava velik broj sustava za upravljanje bazama podataka:

1. MySQL
2. MariaDB
3. PostgreSQL
4. SQLite
5. SQL Server.

6.1 Postavljanje baze podataka

Konfiguracija baze podataka za aplikaciju nalazi se u `config/database.php`. U toj datoteci potrebno je definirati sve veze baze podataka, kao i odrediti koja će veza biti korištena prema zadanim postavkama:

```
'default' => env('DB_CONNECTION', 'mysql'),
```

Također, za lakše konfiguriranje, u datoteci se nalaze primjeri za većinu podržanih sustava baze podataka, npr. za sqlite i mysql:

```
'connections' => [

    'sqlite' => [
        'driver' => 'sqlite',
        'database' => env('DB_DATABASE', database_path('database.sqlite')),
        'prefix' => '',
    ],

    'mysql' => [
        'driver' => 'mysql',
        'host' => env('DB_HOST', '127.0.0.1'),
        'port' => env('DB_PORT', '3306'),
        'database' => env('DB_DATABASE', 'forge'),
        'username' => env('DB_USERNAME', 'forge'),
        'password' => env('DB_PASSWORD', ''),
        'unix_socket' => env('DB_SOCKET', ''),
        'charset' => 'utf8mb4',
        'collation' => 'utf8mb4_unicode_ci',
        'prefix' => '',
        'strict' => true,
        'engine' => null,
    ],

    ...
],
```

Prema zadanim postavkama, Laravelov primjer konfiguracije okoline spreman je za korištenje uz Laravelov Homestead. No naravno, korisnik može napraviti i bazu podataka dediciranu njegovom projektu. Iz Homestead virtualne okoline potrebno je spojiti se s MySQL-om preko `homestead` korisnika i kreirati bazu podataka:

```
mysql -uhomestead -p
// lozinka za homestead je 'secret'
create database homey;
grant usage on *.* to homey@localhost identified by 'password';
//odobravanje pristupa bazi podataka homey
grant all privileges on homey.* to homey@localhost;
//ponovno ucitavanje novododijeljenih dozvola pristupa
flush privileges;
```

Na taj način kreirana je baza podataka za projekt `homey`. Sada je još bitno zabilježiti odabranu lozinku, te urediti datoteku `.env` projekta, mijenjajući vrijednosti parametara baze podataka:

```
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=homey
DB_USERNAME=root
DB_PASSWORD=password
```

6.2 Migracije

Jedan od teških zadataka za razvojne programere je držati sinkroniziranom bazu podataka među razvojnim računalima. Umjesto da svaki programer unutar tima mora dodati stupac u lokalnu shemu baze podataka, Laravel Migrations (migracije) rješavaju taj problem. Migracije su kao kontrola verzije baze podataka, omogućujući timu da jednostavno izmijeni i podijeli shemu baze podataka aplikacije. Dok god se sav posao vezan za baze podataka piše u migracijama, moguće je jednostavno migrirati promjene u bilo koje drugo razvojno računalo.

Migracije se obično uparuju s Laravelovim alatom za izradu sheme, kako bismo lako izgradili shemu baze podataka aplikacije. Laravelova Schema fasada[1] pruža podršku bazama podataka za kreiranje i manipuliranje tablicama u svim Laravelovim sustavima baze podataka. Schema fasadu možemo lako dohvatiti koristeći:

```
use Illuminate\Support\Facades\DB;
```

Kreiranje migracija

Za kreiranje migracije, koja će ukazivati na novi dio sheme baze podataka, koristi se Artisan naredba `make:migration` :

```
php artisan make:migration create_estates_table
```

Nova migracija biti će pohranjena u `database/migrations` direktoriju. Svako ime datoteke migracije sadrži vremenski žig kako bi Laravel znao odrediti poredak migracija.

[1] Pogledati sekciju 3.4.

Pomoću migracije možemo automatski kreirati i tablicu baze podataka čija je shema prikazana tom migracijom. Opcijom `--create` zadajemo ime tablice koja će se kreirati:

```
php artisan make:migration create_estates_table --create=estates
```

Struktura migracije

Klasa migracije sastoji se od dvije metode: `up` i `down`. Metoda `up` koristi se za dodavanje novih tablica, stupaca ili indekasa tablici, dok bi `down` metoda trebala poništiti postupke izvršene metodom `up`. Unutar ovih metoda koristi se Laravelov alat za izradu shema zvan Schema Builder, namijenjen za stvaranje i modificiranje tablica.

Pokretanjem naredbe

```
php artisan make:migration create_estates_table --create=estates
```

kreira se zadana klasa migracije, koja proširuje klasu Migration:

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateEstatesTable extends Migration
{
    /* Run the migrations.*/
    public function up()
    {
        Schema::create('estates', function (Blueprint $table) {
            $table->increments('id');
            $table->timestamps();
        });
    }
    /* Reverse the migrations. */
    public function down()
    {
        Schema::dropIfExists('estates');
    }
}
```

Za kreiranje nove tablice baze podataka, koristi se `create` metoda Schema fasade. Prihvaća dva argumenta, prvo je ime tablice, a drugo je funkcija koja prima Blueprint objekt. Schema fasada sadrži različite vrste stupaca koje je moguće odrediti prilikom izrade tablica. Neke od njih su:

```
$table->boolean('confirmed');
$table->char('name', 100);
$table->dateTime('created_at');
$table->double('amount', 8, 2);
$table->increments('id'); // auto-inkrementirajuci pozitivan cijeli broj
$table->string('name', 100); // ekvivalent VARCHAR tipu sa opcionalnom
    duljinom
$table->text('description');
$table->year('birth_year');
```

Ostale je moguće pronaći na [14].

Osim vrsta stupaca, postoji i nekoliko modifikatora stupaca koje je moguće koristiti prilikom dodavanja stupaca u tablicu baze podataka. Najčešće korišteni su:

```
->default($value)   // Navodjenje zadane vrijednosti za stupac
->nullable()        // Dopusta da NULL vrijednost bude unesena u stupac
->unsigned()        // Postavlja INTEGER stupac kao UNSIGNED
->after('column')   // Postavlja stupac iza drugog stupca
```

Indeksi

Schema fasada podržava nekoliko vrsta indekasa. Za kreiranje jedinstvenog indeksa, dodaje se `unique` metoda na kraj definicije stupca:

```
$table->string('email')->unique();
```

Alternativno, indeks je moguće definirati i nakon definicije stupca.

```
$table->unique('email');
```

Kako bismo stvorili složeni indeks, moguće je metodi `index` proslijediti niz stupaca:

```
$table->index(['account_id', 'created_at']);
```

Korištenjem dosad spomenutih alata kreiramo tablicu za nekretnine:

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateEstatesTable extends Migration
{
   public function up()
    {
        Schema::create('estates', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('user_id')->index();
            $table->string('town')->index();
            $table->string('neighbourhood')->index();
            $table->string('address');
            $table->integer('price')->unsigned()->index();
            $table->string('title');
            $table->string('description');
            $table->timestamps();
        });
    }

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

Dodavanje novih migracija

Kad ne kreiramo novu, već samo mijenjamo postojeću tablicu, dodajemo nove migracije koje prikazuju mijenjanje te tablice. Tada, uz naredbu `migrate`, koristimo opciju `--table`:

```
php artisan make:migration add_price_to_estates_table --table=estates
```

Kad implementiramo metode `up` i `down`, koristimo `Schema::table()` metodu:

```
public function up()
{
    Schema::table('estates', function($table) {
        $table->double('surface')->index();
    });
}
```

Za dodavanje novog stupca iza stupca 'title' koristi se:

```
$table->double('surface')->index()->after('title');
```

Pri kreiranju migracije koja dodaje novi stupac u tablicu, ne smije se zaboraviti implementacija metode uklanjanja istog stupca:

```
public function down()
{
    Schema::table('estates', function($table) {
        $table->dropColumn('surface');
    });
}
```

Pokretanje i uklanjanje migracija

Za pokretanje izvršavanja migracija koje nisu dosad izvršene, poziva se Artisan naredba za migraciju:

```
php artisan migrate
```

Za poništavanje migracije, odnosno vraćanje procesa migriranja unatrag, može se koristiti nekoliko naredbi:

```
php artisan migrate:rollback
php artisan migrate:rollback --step=5
```

Naredba `rollback` uklanja posljednju skupinu migracija, što može uključivati više migracijskih datoteka. Moguće je ukloniti ograničen broj migracija dodavanjem `step` opcije `rollback` naredbi. Naredba koja će ukloniti sve migracije aplikacije zove se `migrate:reset`:

```
php artisan migrate:reset
```

Za učinkovito rekreiranje cijele baze podataka koristi se naredba `migrate:refresh`. Ova naredba uklanja sve migracije aplikacije, te zatim izvršava `migrate` naredbu:

```
php artisan migrate:refresh
```

Naredba:

```
php artisan migrate:fresh
```

će ispustiti sve tablice iz baze podataka, a zatim izvršiti `migrate` naredbu.

Model i migracije

Kad se generira model, može se istovremeno kreirati i kontroler za model, dodavanjem opcije `-c ili --controller`. Ako pak, uz model želimo kreirati migracijsku datoteku, treba dodati opciju `-m` ili `--migration`.

```
php artisan make:model Estate --controller
php artisan make:model Estate --migration
```

Ako, uz model treba kreirati i migracijsku datoteku i kontroler, dodaje se opcija `-mc`:

```
php artisan make:model Estate -mc
```

6.3 Pokretanje neobrađenih SQL upita

Nakon konfiguriranja veze s bazom podataka i kreiranja tablica, moguće je pokretati upite pomoću DB fasade. Ona pruža metode za sve vrste upita baze podataka: `select`, `update`, `insert`, `delete` i `statement`. DB fasadu dohvaćamo koristeći:

```
use Illuminate\Support\Facades\DB;
```

Naredba Select

Kako bismo pokrenuli osnovni upit, možemo koristiti `select` metodu DB fasade:

```
public function index()
  {
      $estates = DB::select('select * from estates where active = ?', [1]);
      return view('estates.index', ['estates' => $estates]);
  }
```

Prvi argument proslijeđen metodi je neobrađen SQL upit, a drugi argument je parametar koji treba vezati za upit. Obično su to vrijednosti ograničenja `where` klauzule. Za predstavljanje veze parametra, umjesto korištenja upitnika, upit je moguće izvršiti pomoću imenovanih veza:

```
$results = DB::select('select * from estates where id = :id', ['id' =>
   1]);
```

Metoda `select` uvijek će vratiti niz rezultata. Svaki rezultat unutar polja bit će PHP `StdClass` objekt koji omogućuje pristup vrijednostima rezultata:

```
foreach ($estates as $estate) {
    echo $estate->title;
}
```

Naredba Insert

Za umetanje podataka u bazu podataka koristi se metoda `insert` DB fasade. Kao i `select`, ova metoda prima neobrađeni SQL upit kao prvi argument, te vezu parametra kao drugi argument:

```
DB::insert('insert into users (id, name) values (?, ?)', [1, 'Dayle']);
```

Naredba Update

Za ažuriranje postojećih zapisa u bazi podataka koristi se metoda `update`. Ona vraća broj redaka na koje iskaz utječe:

```
$affected = DB::update('update users set votes = 100 where name = ?',
   ['John']);
```

Naredba Delete

Metoda `delete` koristi se za brisanje zapisa iz baze podataka. Slično kao i metoda `update`, vraća broj promijenjenih redaka.

```
$deleted = DB::delete('delete from users');
```

Generalna naredba

Neki iskazi baza podataka ne vraćaju nikakvu vrijednost, pa se za takve vrste operacija koristi `statement` metoda:

```
DB::statement('drop table users');
```

6.4 Graditelj upita

Osim neobrađenih upita na bazu podataka, Laravel pruža korisniku svoj graditelj upita, koji korisnika opskrbljuje tečnim sučeljem za kreiranje i pokretanje upita baze podataka.

Graditelj upita namijenjen je za obavljanje većine operacija baze podataka u aplikaciji te radi na svim podržanim sustavima baze podataka.

Laravelov graditelj upita koristi PDO[2] povezivanje parametara kako bi zaštitio aplikaciju od SQL injection napada[3].

Za započinjanje upita potrebno je iskoristiti `table` metodu na DB fasadi. `table` metoda vraća instancu graditelja upita za danu tablicu, omogućujući korisniku postavljanje više ograničenja na upit i konačno dobivanje rezultata pomoću `get` metode. `get` metoda vraća `Illuminate\Support\Collection`.

Metode graditelja upita

Za preuzimanje jednog retka tablice baze podataka koristi se metoda `first`. Ako nije potreban redak, već jedna vrijednost iz zapisa, koristi se metoda `value`. Za preuzimanje kolekcije koja sadrži vrijednost jednog stupca koristi se `pluck` metoda:

```
$user = DB::table('users')->where('name', 'John')->first();
$email = DB::table('users')->where('name', 'John')->value('email');
$roles = DB::table('roles')->pluck('title', 'name');
```

Graditelj upita, također, nudi raznovrsne skupne metode, poput `count`, `max`, `min`, `avg` i `sum`.

Nakon konstruiranja upita moguće je pozvati bilo koju od ovih metoda. Te metode moguće je kombinirati s ostalim metodama:

[2]PHP Data Objects je sučelje za pristup bazama podataka u PHP-u, bez vezanja kodova s određenom bazom podataka.

[3]SQL injection je tehnika ubacivanja koda koja se koristi za napade aplikacija koje se temelje na podacima. Napadi se vrše na način da se maliciozni SQL izrazi umetnu u polje za unos.

```
$price = DB::table('estates')
                ->where('town', 'Zagreb')
                ->avg('price');
```

Naredbe kombiniranja te unije

Graditelj upita omogućuje pisanje i `leftJoin`, `crossJoin` te `union` naredbi:

```
$users = DB::table('users')
            ->leftJoin('posts', 'users.id', '=', 'posts.user_id')
            ->get();

$users = DB::table('sizes')
            ->crossJoin('colours')
            ->get();

$first = DB::table('users')
            ->whereNull('first_name');

$users = DB::table('users')
            ->whereNull('last_name')
            ->union($first)
            ->get();
```

Naredba Where

Za dodavanje `where` klauzule upitu, korisnik može koristiti `where` metodu na instanci graditelja upita. Najosnovniji poziv `where` naredbe zahtijeva tri argumenta. Prvi argument je ime stupca, drugi argument je operator koji može biti bilo koji od podržanih operatora baze podataka. Konačno, treći argument je vrijednost koju treba procijeniti u stupcu:

```
$users = DB::table('users')->where('age', '>=', 24)->get();
```

Osim najjednostavnije verzije, za specifično definiranje uvjeta postoji još mnoštvo metoda, poput:

```
whereBetween / whereNotBetween
whereIn / whereNotIn
whereNull / whereNotNull
```

```
whereDate / whereMonth / whereDay / whereYear / whereTime
whereColumn
```

Naredbe Insert, Update i Delete

Metoda `insert` prihvaća polje imena stupaca i vrijednosti. Moguće je unijeti nekoliko zapisa u tablicu s jednim pozivom za umetanje, prosljeđivanjem polja polja. Svako polje predstavlja redak koji treba umetnuti u tablicu:

```
DB::table('users')->insert([
    ['email' => 'taylor@example.com', 'votes' => 0],
    ['email' => 'dayle@example.com', 'votes' => 0]
]);
```

Kao dodatak umetanju zapisa u bazu podataka, graditelj upita, također, može ažurirati postojeće zapise pomoću `update` metode. Ona, poput `insert` metode, prihvaća niz parova stupaca i vrijednosti, koji sadrže stupce koje treba ažurirati. Moguće je ograničiti upit za ažuriranje pomoću `where` klauzule:

```
DB::table('users')
    ->where('id', 1)
    ->update(['votes' => 1]);
```

Graditelj upita može se koristiti i za brisanje zapisa iz tablice pomoću `delete` metode, gdje se, kao i u `update` naredbi, može ograničiti upit dodavanjem klauzule `where`:

```
DB::table('users')->where('age', '<', 10)->delete();
```

U slučaju da je potrebno izbrisati cijelu tablicu, koristi se metoda `truncate` koja uklanja sve retke te resetira auto-inkrementirajući ID na nulu:

```
DB::table('users')->truncate();
```

6.5 Relacije

Tablice baza podataka često su međusobno povezane. Primjerice, korisnik može imati više nekretnina koje oglašava ili slika može biti povezana s nekretninom za koju je postavljena. Laravelov Eloquent ORM omogućuje da upravljanje i rad s ovim relacijama bude jednostavan te podržava nekoliko različitih tipova relacija:

- jedan naprama jedan (eng. One To One)
- jedan naprama više (eng. One To Many)
- više naprama više (eng. Many To Many)
- jedan naprama više kroz posrednu relaciju (eng. Has Many Through)
- polimorfne relacije (eng. Polymorphic Relations)
- više naprama više polimorfnih relacija (eng. Many To Many Polymorphic Relations).

Kao i sami Eloquent modeli, relacije također služe kao snažni graditelji upita. U Eloquent-u se to postiže definiranjem relacija kao metoda odgovarajućih modela, pa je time omogućeno moćno ulančavanje upita. Prije prikaza načina upotrebe tih metoda, potrebno je pojasniti na koji način se definiraju pojedini tipovi relacija.

Jedan naprama jedan

Jedan naprama jedan relacija je trivijalna relacija koja pokazuje povezanost jedne instance jednog modela s jednom instancom drugog modela.

Primjerice, u aplikaciji bi moglo biti definirano da korisnik ima jednu sliku. Dakle, `User` model je povezan s modelom `Avatar`. Da bi se definirala ova relacija, potrebno je dodati `avatar` metodu u `User` model. `avatar` metoda trebala bi pozvati `hasOne` metodu i vratiti rezultat:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    // Uzmi profilnu sliku korisnika
    public function avatar()
    {
        return $this->hasOne(Avatar::class);
    }
}
```

Argument proslijeđen `hasOne` metodi ime je povezanog modela. Jednom kada je relacija definirana, Eloquent definira **dinamička svojstva** pomoću kojih je moguće dohvatiti zapis iz baze povezanog modela. Dinamička svojstva omogućuju pristup

relacijskim metodama kao da su svojstva definirana na modelu:

```
$picture = User::find(1)->avatar;
```

Ovdje je, dakle, `avatar` pozvan kao svojstvo definirano na modelu, a ne kao metoda. Ovom relacijom omogućeno je pristupanje `Avatar` modelu iz `User` modela.

Eloquent određuje strani ključ relacije na temelju imena modela. U ovom slučaju, za `Avatar` model automatski je pretpostavljeno da ima `user_id` strani ključ. Ako bi korisnik želio zaobići ovu konvenciju, mogao bi proslijediti drugi argument `hasOne` metodi:

```
return $this->hasOne(Avatar::class, 'foreign_key');
```

Dodatno, Eloquent pretpostavlja da bi strani ključ trebao imati vrijednost koja se podudara s `id` (ili sa zadanim `$primaryKey`) stupcem roditelja. Drugim riječima, Eloquent će tražiti vrijednost korisnikovog `id` stupca u `user_id` stupcu od `Avatar` zapisa. Ukoliko bi korisnik htio da relacija koristi drugačiju vrijednost od `id`, mogao bi proslijediti treći argument `hasOne` metodi i odrediti svoj prilagođeni ključ:

```
return $this->hasOne(Avatar::class, 'foreign_key', 'local_key');
```

Inverz relacije

Relaciji `hasOne` (hrv. ima jedan) moguće je definirati inverznu relaciju koristeći `belongsTo` (hrv. pripada) metodu. Programer tada može definirati relaciju na `Avatar` modelu koja će mu omogućiti pristup instanci `User` modela čija je to slika.

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Avatar extends Model
{   // Dohvati korisnika kojem pripada slika
    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

U gornjem primjeru će Eloquent `user_id` `Avatar` modela pokušati pridružiti `id`-u `User` modela. Eloquent određuje zadano ime stranom ključu na način da provjerava ime metode relacije te pridodaje imenu metode sufiks `_id`. Međutim, ako strani ključ na `Avatar` modelu nije `user_id`, moguće je proslijediti prilagođeno ime ključa kao drugi argument do `belongsTo` metode:

```
public function user()
{
    return $this->belongsTo(User::class, 'foreign_key');
}
```

Kao i u prošloj relaciji, moguće je `belongsTo` metodi proslijediti treći argument, ukoliko roditeljski model ne koristi `id` kao svoj primarni ključ, time određujući prilagođeni ključ za roditeljsku tablicu:

```
public function user()
{
    return $this->belongsTo(User::class, 'foreign_key', 'other_key');
}
```

Zadani modeli

`belongsTo` relacija omogućava definiranje zadanog modela koji će biti vraćen ako je predmetna relacija `null`. Ovaj uzorak često je nazivan `Null Object pattern` i može pomoći u brisanju uvjetnih provjera u kodu. U slijedećem primjeru, `user` relacija će vratiti prazan `App\User` model ako nema korisnika pridruženog postu:

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

Kako bi se zadani model popunio atributima, moguće je `withDefault` metodi proslijediti niz:

```
public function user()
{
    return $this->belongsTo(User::class)->withDefault([
        'name' => 'Guest Author',
    ]);
}
```

Jedan naprama više

Relacija jedan naprama više koristi se kod definiranja relacija gdje jedan model ima bilo koju količinu drugih modela.

Primjerice, korisnik može imati beskonačan broj oglasa nekretnina. Kao i sve druge Eloquent-ove relacije, jedan naprama više relacije definirane su smještanjem metode u Eloquent model:

```
<?php

namespace App;
use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    public function estates()
    {
        return $this->hasMany(Estate::class);
    }
}
```

Kao i kod `hasOne` metode, moguće je zaobići strane i lokalne ključeve prosljeđivanjem dodatnih argumenata `hasMany` metodi:

```
return $this->hasMany('App\Comment', 'foreign_key');

return $this->hasMany('App\Comment', 'foreign_key', 'local_key');
```

Jednom kada je relacija određena, moguće je pristupiti kolekciji nekretnina koristeći dinamično `estates` svojstvo:

```
$estates = Auth::user()->estates;

foreach ($estates as $estate) {
    //
}
```

Naravno, zbog toga što relacije služe i kao graditelji upita, moguće je dodati dodatna ograničenja u dohvaćanju nekretnina, pozivajući se na `estates` metodu i daljnjim ulančavanjem uvjeta na upit:

```
$myDeluxeEstates = Auth::user()->estates()->where('surface','>', 200);
```

Inverz relacije jedan naprama više

Sada kada je moguće pristupiti svim nekretninama jednog korisnika, moguće je definirati relaciju koja će omogućiti nekretnini pristup korisniku koji ju je kreirao. Da bi se odredio inverz relacije `hasMany`, potrebno je definirati relacijsku metodu na modelu-djetetu koji poziva `belongsTo` metodu:

```
<?php

namespace App;
use Illuminate\Database\Eloquent\Model;

class Estate extends Model
{
    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

Sada je moguće, pristupanjem `user` dinamičkom svojstvu, pristupiti imenu korisnika koji je kreirao nekretninu:

```
$estate = App\Estate::find(1);

echo $estate->user->name;
```

Više naprama više

Kod više naprama više relacije, jednom elementu tablice istovremeno pripada više elemenata druge tablice, te jednom elementu druge tablice pripada više elemenata prve tablice. Stoga je ta relacija malo kompliciranija te zahtijeva jednu tablicu više.

Primjer takve relacije je učenik koji ide na više predmeta, gdje na iste predmete idu i drugi učenici. Za definiciju ove relacije potrebne su tri tablice baza podataka: `students`, `classes` i `class_student`. Tablica `class_student` izvedena je iz abecednog poretka povezanih imena modela te sadrži `class_id` i `student_id` stupce.

Više naprama više relacije definiraju se pisanjem metode koja vraća rezultat `belongsToMany` metode. Primjerice, moguće je definirati `classes` metodu na `Student` modelu:

```
<?php

namespace App;
use Illuminate\Database\Eloquent\Model;

class Student extends Model
{
    public function classes()
    {
        return $this->belongsToMany('App\Class');
    }
}
```

Jednom kada je relacija definirana, moguće je pristupiti studentovim predmetima koristeći se `classes` dinamičkim svojstvom:

```
$student = App\Student::find(1);
foreach ($student->classes as $class) {
    //
}
```

Naravno, kao i u ostalim tipovima relacija, moguće je pozvati `classes` metodu kako bi se nastavilo ulančavanje ograničenja upita na relaciju:

```
$classes = App\Student::find(1)->classes()->orderBy('name')->get();
```

Definiranje inverzne relacije

Da bi se definirao inverz više naprama više relacije, potrebno je pozvati metodu `belongsToMany` i na povezanom modelu. Nastavno na primjer s predmetima studenata, moguće je definirati `students` metodu na `Class` modelu:

```
<?php

namespace App;
use Illuminate\Database\Eloquent\Model;

class Class extends Model
{
    public function students()
    {
        return $this->belongsToMany('App\Student');
    }
}
```

Kao što je vidljivo, relacija je definirana točno kao njezin duplikat, s iznimkom pozivanja na `App\Student` model. S obzirom da se `belongsToMany` metoda ponovno iskorištava, sve uobičajene opcije prilagodbe ključa i tablice dostupne su pri definiranju inverza više naprama više relacija.

Dohvaćanje stupaca tablice posrednika

Rad s više naprama više relacijama zahtijeva prisutnost tablice posrednika. Eloquent omogućuje neke vrlo korisne načine interakcije s takvom tablicom. Primjerice, nakon pristupa relaciji svih predmeta jednog studenta, moguće je pristupiti tablici posredniku koristeći se `pivot` atributom na modelu:

```
$student = App\Student::find(1);

foreach ($student->classes as $class) {
    echo $class->pivot->created_at;
}
```

Potrebno je primijetiti da je svakom `Class` modelu kojeg dohvatimo dodijeljen `pivot` atribut. Ovaj atribut sadrži model koji predstavlja tablicu posrednika i može biti korišten kao svaki drugi Eloquent-ov model.

Prema zadanim postavkama, samo će ključevi modela biti prisutni u `pivot` objektu. Ako posredna tablica sadrži dodatne atribute, potrebno je odrediti ih pri definiranju relacije:

```
return $this->belongsToMany('App\Class')->withPivot('column1', 'column2');
```

O ostalim relacijama moguće je čitati u [7].

”Lijeno” protiv ”željnog” dohvaćanja

Na ovaj način, kroz Eloquent je uvelike olakšan pristup povezanim elementima različitih tablica. Umjesto pisanja upita bazi za traženje elementa preko njegovog ključa u drugoj tablici, programer jednostavno koristi dinamična svojstva kao da su svojstva glavnog modela. No, s Laravelom treba biti oprezan jer često elementi okruženja koji jako olakšavaju kreiranje sustava programeru, u suštini, usporavaju samu aplikaciju.

Najčešće se dogodi da se cijela aplikacija implementira, a greške se primijete tek kada bude puštena u korištenje i pojavi se velik broj korisnika. Zbog toga je veoma bitno proučiti kako obavljati upite na bazu da serverski dio aplikacije, odjednom, ne bi postao prespor za sve zahtjeve koje dobiva.

Naime, dinamična svojstva Laravela, osim svoje prednosti olakšavanja pristupa elemenata, imaju i svoju manu. Ona koriste ”lijeno dohvaćanje”, što znači da će dohvatiti podatke tek kada im se, zapravo, pristupi. Kad bismo, iz primjera gdje imamo korisnika i njegovu profilnu sliku, htjeli dohvatiti sve korisnike i njihove slike, pomoću dinamičnih svojstva to bismo napravili ovako:

```
$users= App\User::all();

foreach($users as $user{
    echo $user->avatar->path;
}
```

U ovoj petlji može se uočiti ”N+1” problem upita. Kada bi u aplikaciji postojalo deset korisnika, prvi upit na bazu bi dohvatio sve korisnike. Kako dinamična svojstva dohvaćaju podatke tek kada im se pristupi, svaka iteracija iduće petlje gdje se dohvaća slika korisnika, bila bi novi upit na bazu. Dakle, za deset korisnika, to je još deset upita na bazu pa je to sveukupno 11, odnosno, $10 + 1$ upita na bazu.

Kako bi izbjegli ovaj problem, programeri najčešće koriste ”željno dohvaćanje” (eng. ”eager load”) te time smanjuju broj upita na dva. Prilikom upita, moguće je specificirati koja relacija bi trebala biti ”željno dohvaćena” pomoću `with` metode:

```
$users= App\User::with('avatar')->get();

foreach($users as $user{
    echo $user->avatar->path;
}
```

Sada će samo dva upita biti izvršena:

```
select * from users

select * from avatars where id in (1, 2, 3, 4, 5, ...)
```

Ovo je samo jedan od načina na koji se aplikacija može ubrzati. Treba razmisliti i o tome želi li programer svaki put koristiti objekte koje Eloquent stvara iz upita, ili će aplikacija brže raditi ako se koriste neobrađeni upiti na bazu. Na programeru je da razmisli koje performanse aplikacija treba imati i na koji način će to implementirati, te zatim testirati razlike [21, 10, 9, 7].

6.6 Seeding

Kako programer ne bi morao sam ručno unositi testne podatke kroz aplikaciju, Laravel uključuje jednostavnu metodu popunjavanja baze podataka s testnim podacima koristeći Seeder klasu. Sve klase za popunjavanje pohranjene su u `database/seeds` direktoriju. One mogu biti imenovane na bilo koji način, no bilo bi dobro slijediti logičnu konvenciju, primjerice `UsersTableSeeder`, i sl. Osim toga, Laravel nudi `DatabaseSeeder` klasu koja, između ostalog, omogućuje kontroliranje redoslijeda popunjavanja baze.

Pisanje Seeder-a

Da bi korisnik generirao Seeder, potrebno je izvršiti `make:seeder` Artisan naredbu:

```
php artisan make:seeder UsersTableSeeder
```

Svi Seeder-i generirani na taj način bit će smješteni u `database/seeds` direktorij.

Seeder klasa, po zadanim postavkama, sadrži samo jednu metodu: `run`. Ova metoda bit će pozvana kada `db:seed` Artisan naredba bude izvršena. Unutar `run` metode, korisnik može uvrstiti podatke u svoju bazu podataka na način na koji mu odgovara. Kako bi unio podatke korisnik može koristiti graditelj upita ili Eloquent-ove tvornice modela (eng. Eloquent model factories).

Kao primjer, moguće je modificirati zadanu `DatabaseSeeder` klasu i dodati `insert` metodu baze podataka u `run` metodu:

```
<?php

use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;

class DatabaseSeeder extends Seeder
{
    public function run()
    {
        DB::table('users')->insert([
            'name' => str_random(10),
            'email' => str_random(10).'@gmail.com',
            'password' => bcrypt('secret'),
        ]);
    }
}
```

Korištenje tvornice modela

Ručno dodjeljivanje vrijednosti za svaki model može biti nepraktično. Umjesto toga, korisnici mogu koristiti tvornice modela kako bi na prikladan način generirali veliku količinu zapisa u bazi podataka. Nakon što je definirao svoje tvornice, korisnik može koristiti pomoćnu `factory` funkciju kako bi unio zapise u svoju bazu podataka.

Primjerice, moguće je kreirati 50 korisnika i pridružiti vezu za svakog korisnika:

```
public function run()
{
    factory(App\User::class, 50)->create()->each(function ($u) {
        $u->estates()->save(factory(App\Estate::class)->make());
    });
}
```

Pozivanje dodatnih Seeder-a

Kako Seeder klasa ne bi bila prevelika, možemo ju podijeliti na više klasa. Tada, unutar `DatabaseSeeder` klase, koristimo `call` metodu kojoj prosljeđujemo imena svih klasa za popunjavanje. Osim toga, na taj način određujemo da se popunjavanje odvija redoslijedom kojim je to navedeno u `call` metodi.

```
public function run()
{
    $this->call([
        UsersTableSeeder::class,
        EstatesTableSeeder::class,
        AvatarsTableSeeder::class,
    ]);
}
```

Pokretanje Seeder-a

Nakon što je klasa `Seeder` napisana, možda će biti potrebno regenerirati Composerov automatski učitavač (eng. autoloader) koristeći se `dump-autoload` naredbom:

```
composer dump-autoload
```

Korisnik može koristiti i `db:seed` Artisan naredbu da bi vidio svoju bazu podataka. Po zadanim postavkama, naredba `db:seed` pokreće `DatabaseSeeder` klasu koja se može koristiti kako bi pozvala druge seed klase. Međutim, korisnik bi se mogao poslužiti `--class` opcijom da bi specificirao posebnu Seeder klasu koja će biti pokrenuta pojedinačno:

```
php artisan db:seed
php artisan db:seed --class=UsersTableSeeder
```

Korisnik može popuniti svoju bazu podataka naredbom `migrate:refresh`, pomoću koje je moguće vratiti i ponovno pokrenuti sve migracije. Ova naredba je korisna za slučaj građenja baze podataka ispočetka:

```
php artisan migrate:refresh --seed
```

Poglavlje 7

Autentikacija

Autentikacija je postupak provjere je li pojedinac ili drugi entitet ono što tvrdi da jest. Provjera autentičnosti u kontekstu web aplikacija obično se provodi slanjem korisničkog imena ili identifikacijskog dokumenta i jednog ili više privatnih podataka kojima bi jedino korisnik mogao raspolagati. Laravel omogućuje jednostavnu primjenu autentikacije. Zapravo, gotovo je sve već podešeno u startu.

U svojoj jezgri, Laravelovi autentikacijski sustavi napravljeni su od "čuvara" i "opskrbljivača". Čuvari definiraju kako se korisnici autenticiraju pri svakom zahtjevu. Primjerice, Laravel se isporučuje sa `session` čuvarom koji održava stanje koristeći se pohranom sesije i kolačićima. Opskrbljivači definiraju kako se korisnici dohvaćaju iz trajne pohrane. Laravel se isporučuje s podrškom za dohvaćanje korisnika uz pomoć Eloquent-a i graditelja upita baze podataka. Međutim, programer je u mogućnosti slobodno definirati dodatne opskrbljivače, sukladno potrebama aplikacije. Konfiguracijska datoteka nalazi se u `config/auth.php`, gdje se nalazi i više dobro dokumentiranih opcija za dodatno podešavanje ponašanja autentikacijskih sustava.

Potrebno je napomenuti kako velikom broju aplikacija nikada neće ni trebati modifikacija zadanih autentikacijskih postavki.

7.1 Uključeni dijelovi za autentikaciju

Laravel, po zadanim postavkama, uključuje `App\User` Eloquent model pohranjen u `app` direktoriju. Ovaj se model može koristiti sa zadanim Eloquentovim autentikacijskim upravljačkim programom. Ako korisnikova aplikacija ne koristi Eloquent, korisnik može koristiti `database` autentikacijski upravljački program koji se koristi Laravelovim graditeljem upita.

Osim toga, Laravel dolazi s dvije datoteke migracija baze podataka, jedna za kreiranje `users` tablice, druga za kreiranje `password_resets` tablice. Korisnik bi trebao

provjeriti sadrži li njegova `users` (ili ekvivalentna) tablica stupac tipa `remember_token`. Taj će stupac biti korišten za pohranjivanje tokena za korisnike koji odaberu "zapamti me" opciju, kada će se prijavljivati u korisnikovu aplikaciju. Slijedi kod iz `create_users_table` migracije:

```
class CreateUsersTable extends Migration
{
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

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

Tu tablicu se, naravno, može nadopuniti svim spremljenim podacima o korisniku.

Laravel se isporučuje s nekoliko ugrađenih autentikacijskih kontrolera koji se nalaze u `App\Http\Controllers\Auth` prostoru imena:

- `RegisterController` obrađuje registracije novih korisnika
- `LoginController` obrađuje autentikaciju
- `ForgotPasswordController` obrađuje slanje poveznica e-poštom radi poništavanja lozinki
- `ResetPasswordController` sadrži logiku za poništavanje lozinki.

Za velik broj aplikacija neće nikada biti potrebno modificirati ove kontrolere.

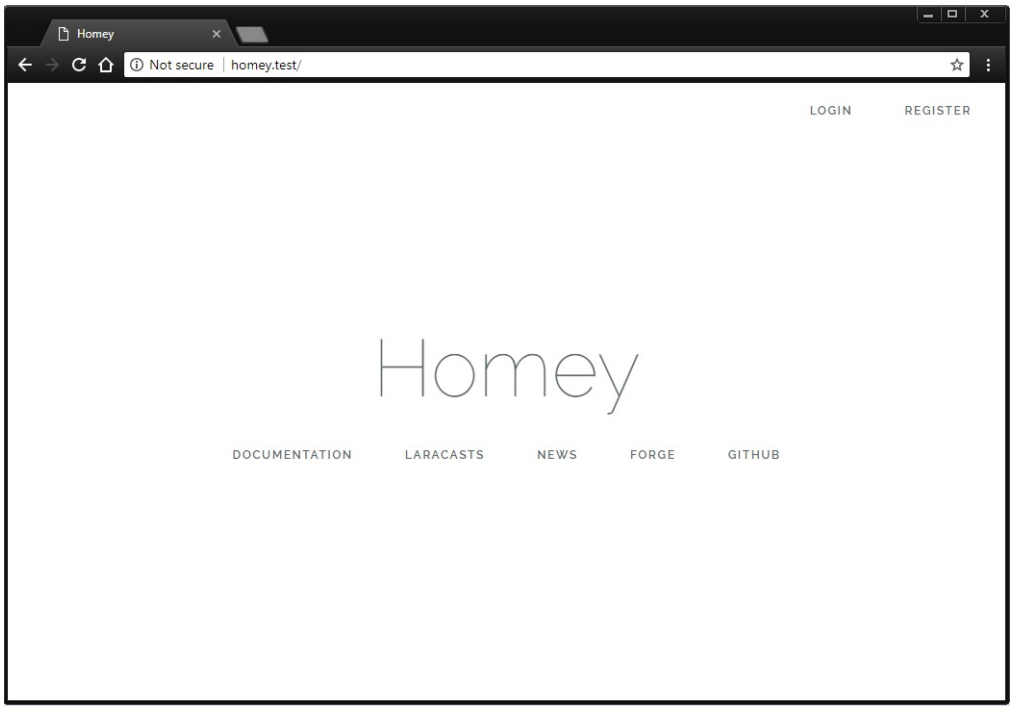
7.2 Postavljanje autentikacije

Laravel omogućuje brzi način za izgradnju[1] (eng. scaffolding) usmjeravanja i pogleda potrebnih za autentikaciju koristeći jednostavnu Artisan naredbu:

```
php artisan make:auth
```

Ova naredba trebala bi biti korištena na novonastalim aplikacijama jer će se njome postići kreiranje glavnog izgleda aplikacije, pogleda za registracije i pogleda za prijavu u aplikaciji. Svi pogledi koji su potrebni za autentikaciju biti će smješteni u direktorij `resources/views/auth`. Uz to, biti će kreiran i `resources/views/layouts` direktorij, koji će sadržavati novokreirani glavni izgled aplikacije. Svi ovi pogledi koriste Bootstrap CSS okruženje, no korisnik ih može prilagođavati po želji.

Naredba `make:auth` će, također, generirati usmjeravanja za sve krajnje točke autentikacije i `HomeController`, koji će, nakon prijave korisnika, obrađivati njegove daljnje zahtjeve na navigacijskoj ploči.



[1]Scaffolding je tehnika podržana od nekih MVC okruženja, u kojoj programer može odrediti kako se može koristiti baza podataka aplikacije. Okruženje koristi tu specifikaciju zajedno s predefiniranim kodnim predlošcima za generiranje konačnog koda. Aplikacija tada tretira predloške kao ”skele” na kojima će se graditi snažnija aplikacija.

S obzirom da ugrađeni autentikacijski kontroleri već sadrže logiku za autentikaciju postojećih korisnika i pohranjivanje novih korisnika u bazu, korisnik sada može svojoj aplikaciji pristupiti u pregledniku te registrirati i autenticirati nove korisnike aplikacije.

Prilagođavanje zadanih postavki

Kad je korisnik uspješno autenticiran, preusmjerit će se na `/home` URI. Post-autentikacijska preusmjerna lokacija može se izmijeniti definiranjem `redirectTo` svojstva u `LoginController`, `RegisterController` i `ResetPasswordController` kontrolerima:

```
protected $redirectTo = '/';
```

Ukoliko je za preusmjernu lokaciju potrebno prilagoditi logiku generiranja, moguće je definirati `redirectTo` metodu, umjesto `redirectTo` svojstva:

```
protected function redirectTo()
{
    return '/path';
}
```

Laravel, po zadanim postavkama, koristi `email` polje za autentikaciju. Ukoliko bi programer htio ovo izmijeniti, moguće je definirati `username` metodu u kontroleru `LoginController`:

```
public function username()
{
    return 'username';
}
```

Osim toga, kako bi se modificirala polja obrasca za registraciju novog korisnika u aplikaciju, ili kako bi se prilagodio način na koji se novi korisnici pohranjuju u bazu podataka, moguće je modificirati kontroler koji je odgovoran za potvrđivanje i kreiranje novih korisnika u aplikaciji, zvan `RegisterController`. U tom kontroleru postoji `validator` metoda koja sadrži pravila validacije za nove korisnike u aplikaciji, te `create` metoda koja je odgovorna za stvaranje novih `App\User` zapisa u bazi podataka koristeći `Eloquent ORM`. Programer može slobodno modificirati ove metode, vodeći se potrebama svoje baze podataka.

7.3 Dohvaćanje autenticiranog korisnika

Autenticiranom korisniku moguće je pristupiti koristeći se `Auth` fasadom:

```
use Illuminate\Support\Facades\Auth;

// Dohvacanje trenutno autenticiranog korisnika
$user = Auth::user();

// Dohvacanje ID-a trenutno autenticiranog korisnika
$id = Auth::id();
```

Alternativno, jednom kada je korisnik autenticiran, moguće je pristupiti autenticiranom korisniku putem `Illuminate\Http\Request` instance:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ProfileController extends Controller
{
    public function update(Request $request)
    {
        $request->user(); //vraca instancu autenticiranog korisnika
    }
}
```

Kako bi se provjerilo je li korisnik prijavljen u aplikaciju, moguće je koristiti `check` metodu iz `Auth` fasade, koja će vratiti `true` ako je korisnik autenticiran:

```
use Illuminate\Support\Facades\Auth;

if (Auth::check()) {
    //Korisnik je ulogiran
}
```

Osim toga, u pogledima je moguće odvojiti dijelove pogleda koje će vidjeti autenticirani korisnici, od dijelova koda koje će vidjeti gosti.

Utvrđivanje je li trenutni korisnik autenticiran ili gost provodi se pomoću Blade provjera za autentikaciju:

```
@auth
   //Ovdje ide dio pogleda koji ce vidjeti autenticirani korisnici
@endauth

@guest
   //Ovdje ide dio pogleda koji ce vidjeti gosti
@endguest
```

7.4 Zaštita ruta

Middleware[2] za preusmjeravanje može biti korišten kako bi omogućio samo autenticiranim korisnicima pristup datoj ruti. Laravel dolazi s `auth` *middleware*-om, koji je definiran u `Illuminate\Auth\Middleware\Authenticate`. S obzirom na to da je taj *middleware* već registriran u HTTP jezgri, sve što je potrebno napraviti je pridružiti *middleware* definiciji rute:

```
Route::get('profile', function () {
    // Ulaz dozvoljen samo autenticiranim korisnicima
})->middleware('auth');
```

Naravno, ukoliko se u rutama koriste kontroleri, moguće je pozvati `middleware` metodu iz kontrolerovog konstruktora, umjesto direktnog pridruživanja definiciji rute:

```
public function __construct()
{
    $this->middleware('auth');
}
```

Reguliranje prijave (Login Throttling)

Ukoliko se u aplikaciji koristi Laravelov ugrađen `LoginController` kontroler, u njega će već biti uključeno `Illuminate\Foundation\Auth\ThrottlesLogins` svojstvo. Po zadanim postavkama, korisnik se neće moći prijaviti u trajanju jedne minute ukoliko nekoliko puta unese krive podatke. Reguliranje je jedinstveno povezano (unique to) s korisnikovim imenom / e-mail adresom te IP adresom [20, 1].

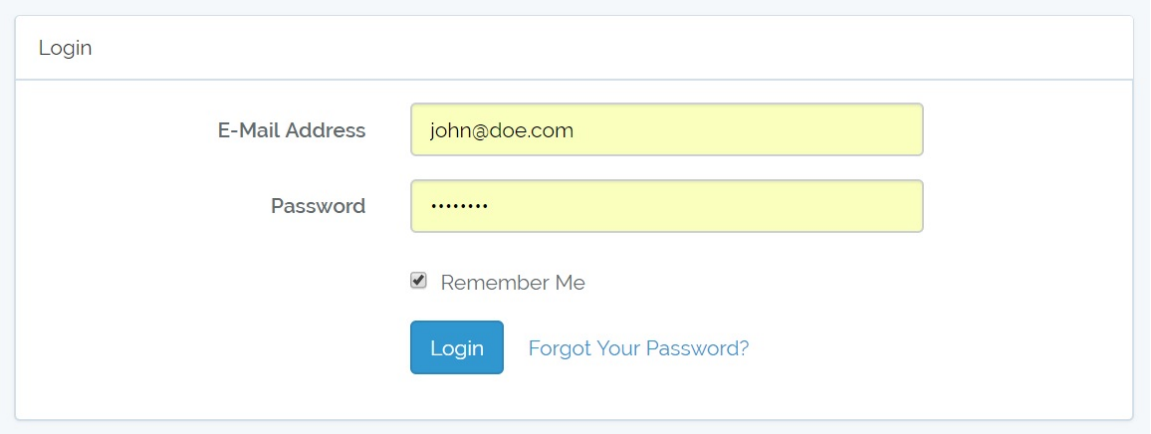
[2]Middleware pruža prikladan mehanizam za filtriranje zahtjeva HTTP-a koji se unose u aplikaciju. Na primjer, Laravel sadrži middleware koji potvrđuje da je korisnik aplikacije autenticiran. Ako korisnik nije autenticiran, middleware će preusmjeriti korisnika na zaslon za prijavu. Međutim, ako je korisnik autenticiran, middleware će omogućiti zahtjev da nastavi dalje u aplikaciju.

Poglavlje 8

Primjena

Kroz prethodna poglavlja objašnjen je način na koji se aplikacija razvija u Laravel okruženju te se kroz jednostavnije i složenije primjere koda moglo naslutiti o kakvoj aplikaciji je riječ.

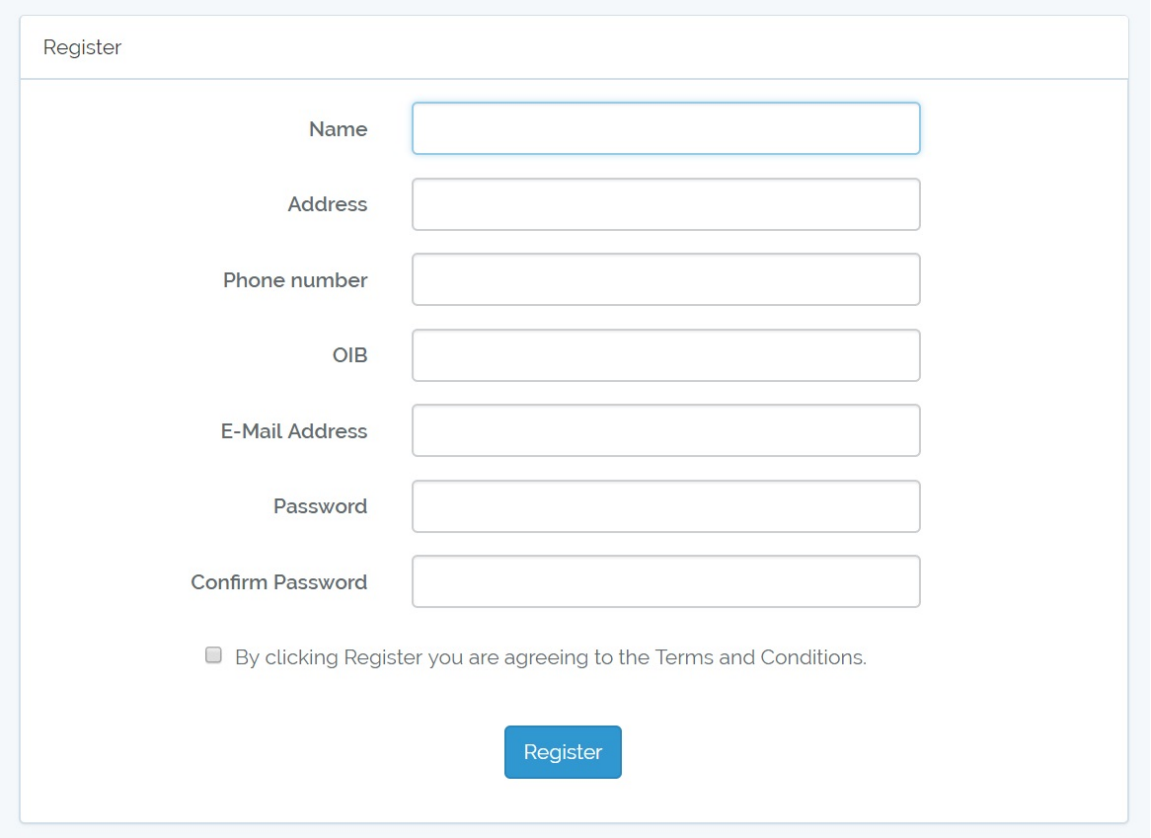
Primjer aplikacije izrađene u Laravelu je aplikacija "Homey", zamišljena kao svojevrsni oglasnik s nekretninama, čiji je cilj osim bazične funkcije oglasnika, osigurati sigurnije oglašavanje nekretnina na način da samo autenticirani korisnici mogu pregledavati oglase. Svaki korisnik koji se želi registrirati mora dati svoje osobne podatke, a zauzvrat na svakom oglasu može vidjeti točnu adresu nekretnine kako bi lakše odlučio je li dana lokacija ona koja odgovara njegovoj potrazi.



Slika 8.1: Forma za prijavu korisnika

Korisnik se u aplikaciju registrira sa sljedećim podacima: ime, adresa, broj mobitela, OIB, e-mail adresa i lozinka (Slika 8.2). Nakon što je korisnik registriran

prikazuje mu se naslovna stranica s porukom da je uspješno registriran. Pri prijavljivanju potrebno je unijeti e-mail adresu te lozinku, kao što je prikazano na slici 8.1.



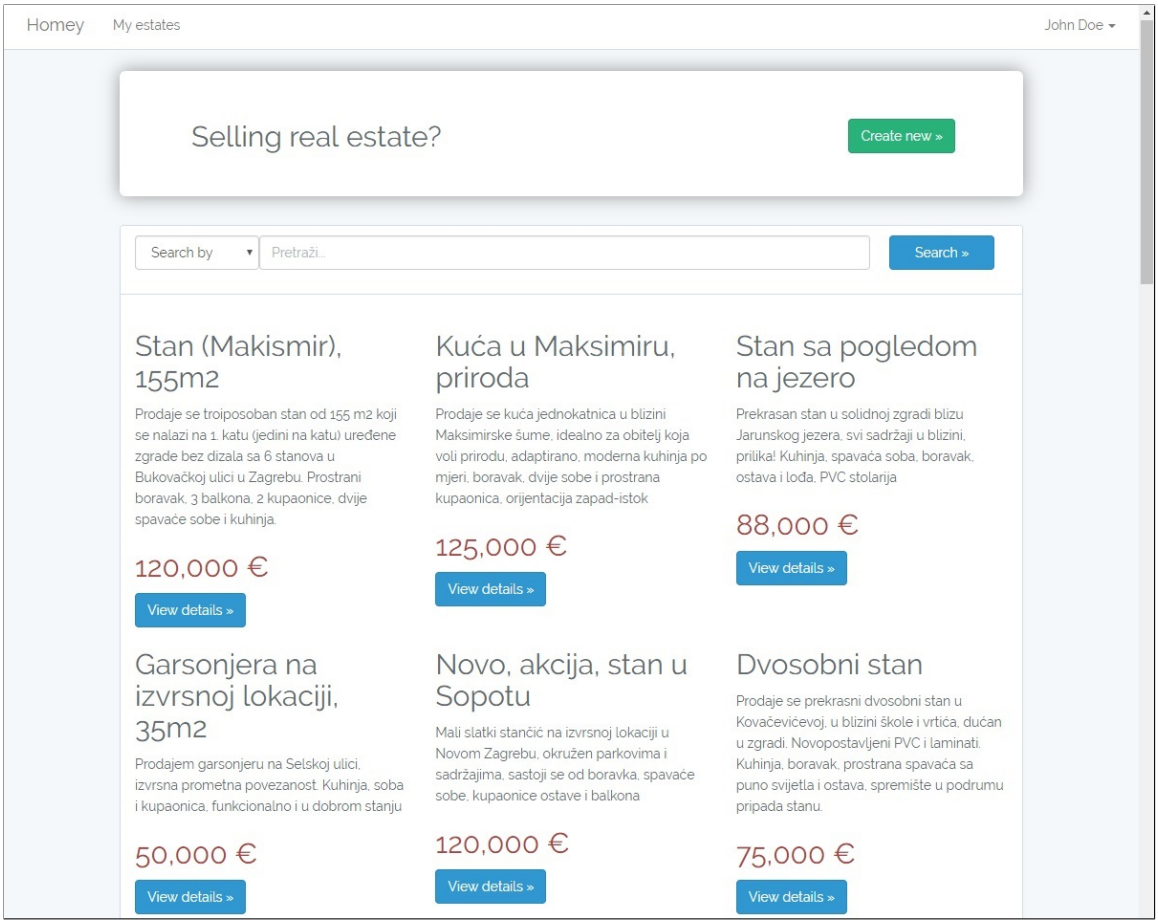
Slika 8.2: Forma za registriranje korisnika

Isječak koda kontrolera za registraciju:

```
protected $redirectTo = '/home';

public function __construct()
    {
        $this->middleware('guest');
    }

protected function validator(array $data)
    {
        return Validator::make($data, [
            'name' => 'required|string|max:255',
```

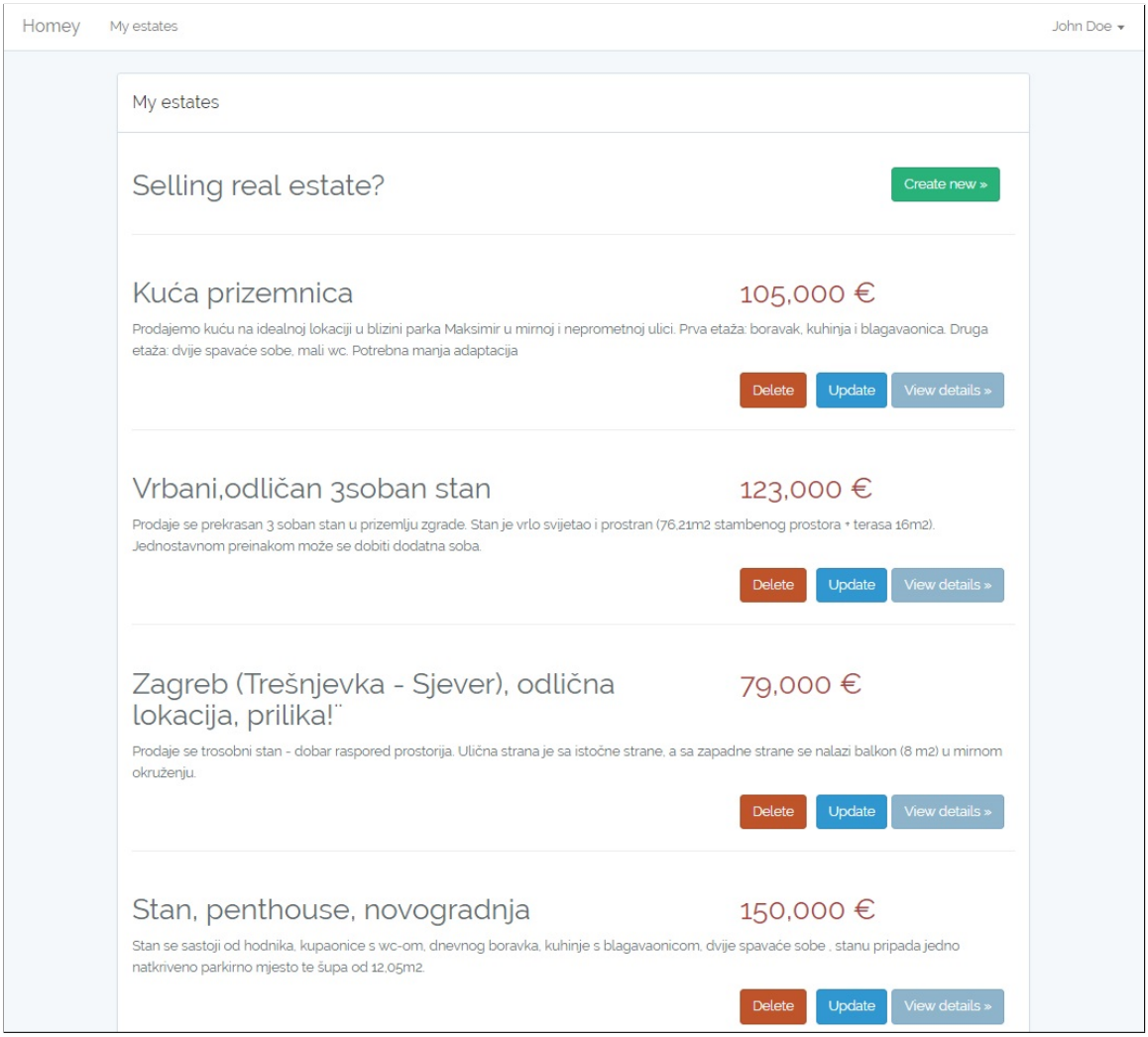
```
            'address' => 'required|string|max:255',
            'phone_number' => 'required|string|max:15',
            'OIB' => 'required|string|size:11|unique:users',
            'email' => 'required|string|email|max:255|unique:users',
            'password' => 'required|string|min:6|confirmed',
            'Terms' => 'required_without_all',

        ]);
    }
```

Na naslovnoj stranici prikazani su oglasi drugih korisnika, te postoji mogućnost pretraživanja nekretnina. Osim toga, s naslovne stranice moguće je doći do forme za kreiranje novog oglasa.



Slika 8.3: Prikaz naslovne stranice prijavljenog korisnika

Kroz navigacijsku ploču moguće je otvoriti stranicu s vlastitim oglasima nekretnina klikom na "My estates", naslovnu stranicu klikom na "Homey", te obaviti odjavu korisnika u padajućem izborniku koji se stvori klikom na ime korisnika.



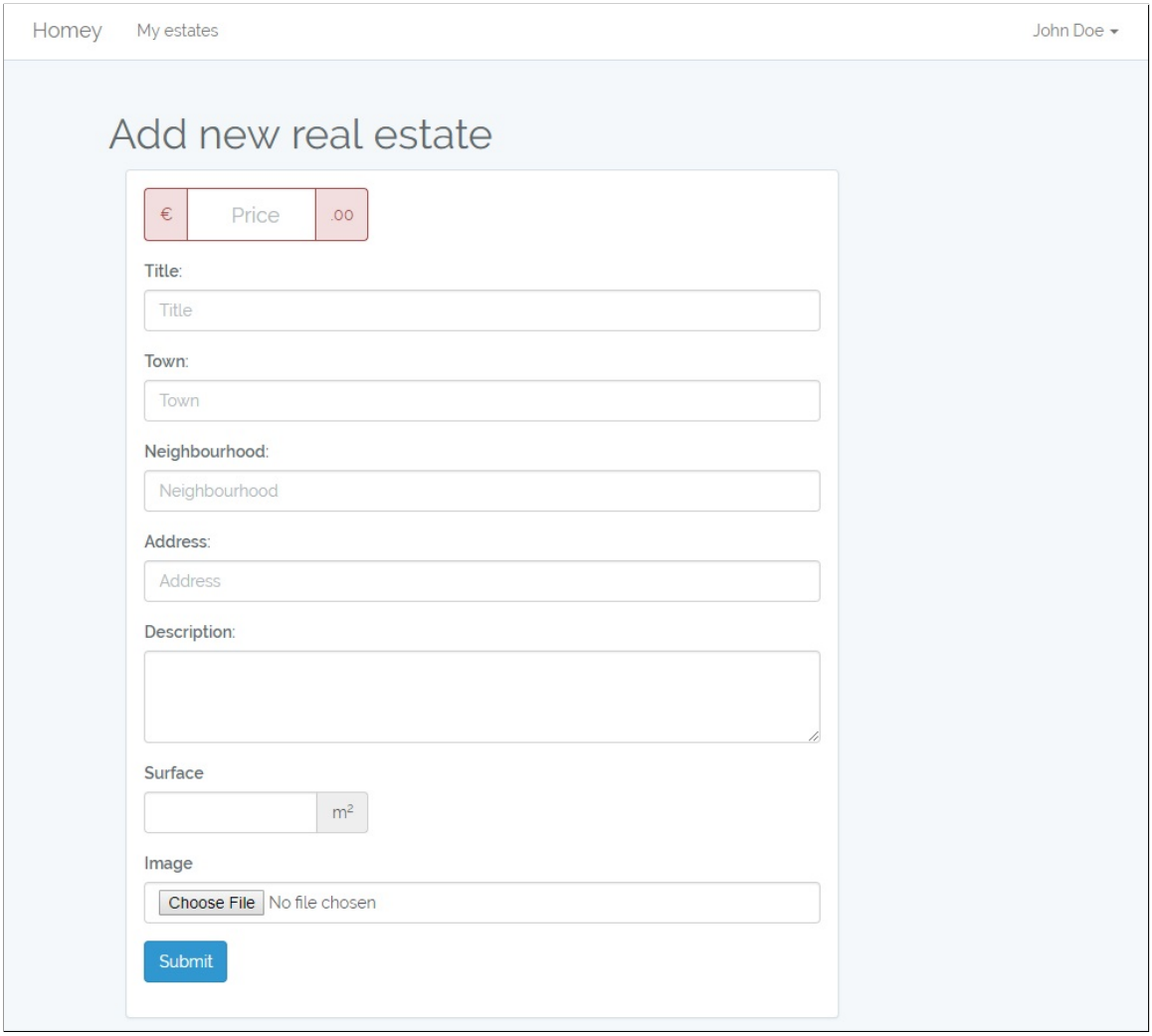
Slika 8.4: Prikaz stranice s oglasima nekretnina prijavljenog korisnika

Na stranici "My estates" moguće je za svaku nekretninu odabrati tri opcije: "Delete", "Update" i "View Details".

Odabirom opcije "View Details" prikazuje se stranica s detaljnim pregledom cijelog oglasa. Klikom na opciju "Delete" korisnik briše pripadajući oglas.

Odabirom opcije "Update" otvara se stranica koja sadrži formu, istu kao za kreiranje novog oglasa, prikazanu na slici 8.5. Međutim, u opciji ažuriranja sva polja

forme će biti već popunjena podacima pripadajućeg oglasa, kako bi korisnik mogao lako izmijeniti bilo koji detalj oglasa.



Slika 8.5: Prikaz forme za kreiranje novog oglasa

Pri izradi svih pogleda korišteno je Bootstrap razvojno okruženje [2].

Sljedeći isječak koda prikazuje implementirane neke od glavnih metoda resursnog kontrolera za upravljanje nekretninama:

```
<?php

namespace App\Http\Controllers;
```

```
use App\Estate;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Storage;

use Auth;
use App\User;

class EstatesController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }

    public function index()
    {
        $estates = Auth::user()->estates;

        return view('estates.index', compact('estates'));
    }

    public function create()
    {
        return view('estates.create');
    }

    public function store(Request $request)
    {

       $this->validate(request(), [

        'price' => 'required|digits_between:1,8',
        'title' => 'required|string|max:255',
        'town' => 'required|string|max:255',
        'address' => 'required|string|max:255',
        'neighbourhood' => 'required|string|max:40',
        'surface' => 'required|digits_between:1,6',
        'description' => 'string|max:511',
        'image' => 'image'

        ]);
```

```
        $estate = Estate::create([
            'user_id' => Auth::user()->id,
            'price' => request('price'),
            'title' => request('title'),
            'town'=> request('town'),
            'address'=> request('address'),
            'neighbourhood'=> request('neighbourhood'),
            'description'=> request('description'),
            'surface'=> request('surface')
            ]);

        if ($request->hasFile('image')) {
            $imagePath = $request->file('image')->store('public/images');
            $estate->images()->create([
            'path' => basename($imagePath),
        ]);
        }
        session()->flash('message', 'Estate is successfully created');
        session()->flash('alert-class', 'alert-success');

        return redirect('/estates');

    }

    public function show(Estate $estate)
    {
        $image=$estate->images()->first();
        if($image) $path=$image->path;
        else $path="empty";
        return view('estates.show', compact('estate', 'path'));
    }

    public function edit(Estate $estate)
    {
        if($estate->user_id == Auth::id())
            return view('estates.edit', compact('estate'));
        else return redirect('/estates');
    }

    public function update(Request $request, Estate $estate)
```

```
    {
        $estate->update($request->except('image'));

        if ($request->hasFile('image')) {

            $imagePath = $request->file('image')->store('public/images');
            $image=$estate->images()->first();

            //if image already exists
            if($image){
                Storage::disk('local')->
                    delete('public/images/'.$image->path);
                $estate->images()->
                    update(['path' => basename($imagePath)]);
            }
            else{
                $estate->images()->create(
                    ['path' => basename($imagePath)]);
            }
        }
        session()->flash('message', 'Estate is successfully updated');
        session()->flash('alert-class', 'alert-success');
        return redirect('/estates');
    }

    ...

}
```

Slijedom navedenih prikaza u ovom poglavlju iznesene su osnovne funkcionalnosti aplikacije Homey. U budućnosti postoji prostor za njen daljnji razvoj i nadogradnju, ali za potrebe ovog rada navedene funkcionalnosti dobro prikazuju razvoj aplikacije u Laravel okruženju.

Bibliografija

[1] *Authentication*, https://laravel.com/docs/5.5/authentication, posjećeno 25. 1. 2017.

[2] *Bootstrap documentation*, https://getbootstrap.com/docs/3.3/, posjećeno 15. 1. 2017.

[3] Zvonimir Bujanović, *PHP – Model-View-Controller*, https://web.math.pmf.unizg.hr/nastava/rp2d/slideovi/2017/Predavanje%2007%20-%20PHP%20-%20Model-View-Controller.pdf, posjećeno 15. 1. 2018.

[4] Nicholas Cerminara, *Getting Started with Laravel Homestead*, https://scotch.io/tutorials/getting-started-with-laravel-homestead#toc-clone-the-repo, posjećeno 5. 12. 2017.

[5] Zeekat Software Development, *The Model View Controller pattern in web applications*, https://zeekat.nl/articles/mvc-for-the-web.html, posjećeno 20. 12. 2017.

[6] *Eloquent: Collections – Available Methods*, https://laravel.com/docs/5.5/eloquent-collections#available-methods, posjećeno 11. 1. 2017.

[7] *Eloquent: Relationships*, https://laravel.com/docs/5.5/eloquent-relationships, posjećeno 15. 1. 2017.

[8] Michael J. Garbade, *How to choose a PHP framework*, https://opensource.com/business/16/6/which-php-framework-right-you, posjećeno 5. 1. 2018.

[9] Jozsef Hocza, *Eloquent Abused By Many Developers*, https://hocza.com/2016-11-08/eloquent-abused-by-many-developers/, posjećeno 15. 1. 2018.

[10] ———, *Why Laravel Is Slow – Eager Loading, Caching*, https://hocza.com/2016-11-05/why-laravel-is-slow-part-2/, posjećeno 15. 1. 2018.

[11] *Laravel Configuration*, https://laravel.com/docs/5.5/configuration#maintenance-mode, posjećeno 5. 12. 2017.

[12] *Laravel*, https://github.com/laravel/laravel/blob/master/readme.md, posjećeno 2. 12. 2017.

[13] *Laravel Homestead*, https://laravel.com/docs/5.5/homestead, posjećeno 5. 12. 2017.

[14] *Laravel Migrations #Columns*, https://laravel.com/docs/5.5/migrations#columns, posjećeno 13. 12. 2017.

[15] *Laravel*, https://en.wikipedia.org/wiki/Laravel, posjećeno 2. 12. 2017.

[16] *Model–view–controller*, https://en.wikipedia.org/wiki/Model-view-controller, posjećeno 20. 12. 2017.

[17] Anna Monus, *10 PHP Frameworks For Developers — Best of*, https://www.hongkiat.com/blog/best-php-frameworks/, posjećeno 7. 1. 2018.

[18] Jesse O'Brien, *A Brief History of Laravel*, https://medium.com/vehikl-news/a-brief-history-of-laravel-5d55970885bc, posjećeno 10. 12. 2017.

[19] Osteel, *How to start a new Laravel 5 project with Homestead – quick reference*, http://tech.osteel.me/posts/2015/04/23/how-to-start-a-new-laravel5-project-with-homestead-quick-reference.html, posjećeno 5. 12. 2017.

[20] Prosper Otemuyiwa, *Creating your first Laravel app and adding authentication*, https://auth0.com/blog/creating-your-first-laravel-app-and-adding-authentication/, posjećeno 20. 1. 2017.

[21] Sean Patterson, *Optimizing Laravel Performance*, https://www.freshconsulting.com/optimizing-laravel-performance/, posjećeno 15. 1. 2018.

[22] Joel Reyes, *Discussing PHP Frameworks: What, When, Why and Which?*, https://www.noupe.com/development/discussing-php-frameworks.html, posjećeno 6. 1. 2018.

[23] Bruno Skvorc, *6 Reasons to Move to Laravel Homestead*, https://www.sitepoint.com/6-reasons-move-laravel-homestead/, posjećeno 5. 12. 2017.

[24] _____, *The Best PHP Framework for 2014*, https://www.sitepoint.com/best-php-frameworks-2014/, posjećeno 15. 12. 2017.

[25] _____, *The Best PHP Framework for 2015: SitePoint Survey Results*, https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/, posjećeno 15. 12. 2017.

[26] A. Smijulj i A. Meštrović, *Izgradnje MVC modularnog radnog okvira*, Zbornik Veleučilišta u Rijeci **2** (2014), br. 1, 215–232.

[27] *Software Framework*, https://www.techopedia.com/definition/14384/software-framework, posjećeno 12. 12. 2017.

[28] Eleganz IT solutions, *Most Used PHP Frameworks in 2017*, https://medium.com/@vishva.eleganzit/most-used-php-frameworks-in-2017-73572e562fe9, posjećeno 15. 12. 2017.

[29] *Why Laravel Is The Best PHP Framework In 2018?*, https://www.valuecoders.com/blog/technology-and-apps/laravel-best-php-framework-2017/, posjećeno 5. 1. 2018.

Sažetak

Laravel je besplatno, *open-source* PHP razvojno okruženje zamišljeno za razvoj web aplikacija, koje, kroz svoju izražajnu i elegantnu sintaksu, omogućava programeru da brzo i efikasno implementira komponente aplikacije.

Rad započinje objašnjavanjem pojma i funkcije razvojnog okruženja. Potom se, kao tema rada, uvodi Laravel te se ukratko izlaže njegova povijest i usporedba s drugim PHP razvojnim okruženjima. U nastavku rada se na sistematičan način prikazuje razvoj aplikacije u Laravel okruženju. Prvo se govori o Laravelovim značajkama koje pridonose njegovoj funkcionalnosti. Zatim se objašnjava MVC arhitekturalni obrazac koji predstavlja jezgru arhitekture aplikacije izrađene u Laravelu te tok usmjeravanja kroz aplikaciju, koji MVC upotpunjuje do jedinstvene cjeline. Potom se prelazi na objašnjenje interakcije s bazama podataka koju Laravel omogućuje. Konačno, obrađuje se i autentikacija, kao nužan element svake moderne web aplikacije.

Rad završava poglavljem praktične primjene u kojem autor svoju aplikaciju, koju je u cijelosti razvio uz pomoć Laravela, koristi kao primjer razvoja aplikacije u ovom okruženju. Uz to, kod aplikacije protkan je kroz cijeli rad kako bi se čitatelju prikazali jednostavniji i složeniji primjeri korištenja Laravelovih komponenata i alata za izradu aplikacije.

Summary

Laravel is a free, *open-source* PHP framework designed for web application development which uses its expressive and elegant syntax to enable the developer to implement application components quickly and efficiently.

The thesis begins by explaining the concept and function of the framework. Then, as the main topic, Laravel is introduced with a brief overview of its history and comparison with other PHP frameworks. The thesis continues with a systematic elaboration of a Laravel framework application development. Firstly, Laravel's features that contribute to its functionality are discussed. Secondly, the MVC software architectural pattern which is the core of the architecture of a Laravel made application is explained, as well as the routing which completes MVC to a unique entity. Then the interaction with databases that Laravel provides is elaborated. Finally, authentication as a necessary element of any modern web application is discussed.

The thesis ends with a chapter of practical use in which the author presents the application that she fully developed using Laravel, to demonstrate how an application can be developed in this framework. Additionally, the application code is interwoven through the whole thesis so that simpler and more complicated examples of using Laravel's components and tools for creating the application are demonstrated.

Životopis

Rođena sam 30. siječnja 1994. u Zagrebu. Prva tri razreda osnovne škole pohađam u Osnovnoj školi Ive Andrića u Sopotu, a potom se selim na Maksimir, gdje do kraja osmog razreda pohađam Osnovnu školu Bukovac. Svoje obrazovanje nastavljam u XV. prirodoslovno–matematičkoj gimnaziji u Zagrebu koju pohađam u periodu od 2008. do 2012. godine. Zbog interesa za matematiku i znatiželje kako se ona može primijeniti u računarstvu, 2012. godine upisujem preddiplomski sveučilišni studij Matematike na Matematičkom odsjeku Prirodoslovno–matematičkog fakulteta u Zagrebu kojeg uspješno završavam 2015. Iste godine upisujem diplomski sveučilišni studij Računarstvo i matematika. Osim matematike, zanima me i ekologija pa u slobodno vrijeme volim boraviti u prirodi, brinuti se za biljke te fotografirati.