Prednosti izrade web rješenja korištenjem Laravel programskog okvira

Šimić, Domagoj

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

Sveučilišni studij

PREDNOSTI IZRADE WEB RJEŠENJA KORIŠTENJEM LARAVEL PROGRAMSKOG OKVIRA

Diplomski rad

Domagoj Šimić

Osijek, 2017.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 06.07.2017.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Domagoj Šimić
Studij, smjer:	DRC - Programsko inženjerstvo
Mat. br. studenta, godina upisa:	D 815 R, 12.10.2015.
OIB studenta:	34511945152
Mentor:	Izv. prof. dr. sc. Krešimir Nenadić
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Ivica Lukić
Član Povjerenstva:	Doc.dr.sc. Mirko Köhler
Naslov diplomskog rada:	Prednosti izrade web rješenja korištenjem Laravel programskog okvira
Znanstvena grana rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	Navesti prednosti i nedostatke izrade web rješenja u Laravelu u odnosu na neke druge programske okvire ili bez upotrebe programskog okvira. Primijeniti Laravel programski okvir u izradi web rješenja s osvrtom na brzinu izrade, sigurnost, uporabu MVC paradigme, organizacije programskog koda te drugih prednosti Laravela.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	06.07.2017.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 19.07.2017.

Ime i prezime studenta:	Domagoj Šimić
Studij:	DRC - Programsko inženjerstvo
Mat. br. studenta, godina upisa:	D 815 R, 12.10.2015.
Ephorus podudaranje [%]:	0%

Ovom izjavom izjavljujem da je rad pod nazivom**: Prednosti izrade web rješenja korištenjem Laravel programskog okvira**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Krešimir Nenadić

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD

Posljednjih godina, razvoj web aplikacija sve više te polako zamjenjuje desktop aplikacije. Kako IT kompanije i individualni programeri koriste različite programske jezike u svrhu razvoja web rješenja, dolazi do natjecanja između performansi programskih jezika u okviru kvalitetnog i brzog razvoja. Kako pojedini jezici imaju svoje prednosti i nedostatke u vidu performansi, pojavljuju se različiti programski okviri čija je svrha unaprijedit performanse razvoja web rješenja. Neki od najpopularnijih programskih okvira su .NET (prvenstveno uz korist C# programskoj jezika), Spring (Java programski jezik), Rails (Ruby programski jezik), Django (Python programski jezik) i mnogi drugi. Iz razloga visoke kvalitete navedenih programskih jezika i njihovih programskih okvira, dolazi do razmišljanja u krugu programera kako PHP programski jezik više nije pogodan za razvoj web rješenja jer kako tehnologija napreduje tako napreduju i zahtjevi kojima se PHP može teško prilagoditi. Rezultat toga je nastanak različitih PHP programskih okvira kao što su Symfony, Yii, Zend te relativno novog Laravela koji svakim danom stječe sve veći krug korisnika. Zadatak ovog rada je predočiti razloge popularnosti kao i opisati nedostatke.

Kako bi se pokrili svi bitni aspekti koji predočavaju Laravelove koncepte i tijek razvoja web rješenja, rad je podijeljen u šest poglavlja od kojih prvo poglavlje sadrži kratki uvod u diplomski rad. Drugo poglavlje opisuje koncepte programskih okvira, daje kratki opis Laravel programskog okvira te njegovih prednosti i nedostataka, kao i prednosti i nedostatke samog PHP programskog jezika. Treće poglavlje opisuje postupke instalacije i konfiguracije nove Laravel aplikacije, osnovne koncepte Laravela kao što su MVC arhitektura i objektno-relacijsko mapiranje te daje uvid u osnovne sigurnosne probleme i kako ih nadvladati. Četvrto poglavlje sadrži opis i način razvoja web aplikacije za rezerviranje apartmana, putem kojeg čitatelj dobiva uvid u osnove tijeka i načina izrade web rješenja putem Laravel programskog okvira, kao i njegovih alata. Peto poglavlje daje kratku usporedbu kojom se predočava korisnost programskih okvira u razvoju web rješenja naspram razvoj bez programskog okvira te kratku usporedbu Laravela s drugim popularnim PHP programskim okvirima. Zadnje, šesto, poglavlje daje kratki sumirani osvrt na postignute ciljeve rada i mogućnost njegovog korištenja u svrhu učenja ili daljnjeg istraživanja.

2. PROGRAMSKI OKVIRI I LARAVEL

U ovom poglavlju objasnit će se koncepti programskih okvira, Laravel programski okvir te njegove prednosti i nedostatci kao i prednosti i nedostatci PHP programskog jezika.

2.1 Programski okviri

Programski okvir može se definirati kao set strukturiranih programa ili modula koji služi kao kostur pri izradi aplikacijskih rješenja. To su jedinstvene platforme za višekratnu uporabu koje u sebi sadrže kompajlere, programsku podršku, aplikacijska programska sučelja (eng. *Application Programming Interface*, API - skup funkcija ili procedura koje omogućuju aplikaciji pristup podatcima ili značajkama druge aplikacije, operacijskog sustava ili nekoj drugoj vrsti sustava) te druge alate koji omogućavaju razvoj aplikacijskih rješenja.

Arhitekturalno gledano, programski okviri se sastoje od takozvanih smrznutih i vrućih mjesta. Smrznuta mjesta podrazumijevaju sveukupnu arhitekturu sustava izgrađenog od različitih programskih komponenti i njihovih međusobnih veza. Vruća mjesta podrazumijevaju dijelove sustava gdje programeri dodaju svoj programski kod te specifične funkcionalnosti koje su potrebne za specifični projekt.

Općenito, programski okvir koristi se kako bi se pružio lakši i brži način razvoja aplikacijskih rješenja. Razvoj aplikacija se odvija brže jer se programer može posvetiti razvoju funkcionalnosti aplikacija bez značajnog gubitka vremena na razvoj detalja niže razine aplikacije. Također, savladavanjem programskog okvira njegovim korištenjem u prijašnjim aplikacijama imat će znatni odraz na brzinu razvoja sljedećih aplikacijskih rješenja.

Postoje različiti programski okviri [1]:

a) Javascript/Ajax okviri – Okvir koji koristi Javascript/Ajax tehnologije za dinamički razvoj web aplikacija i stranica na klijentskoj strani.
b) Web aplikacijski okviri – Koristi se većinom za razvoj dinamičkih web aplikacija i stranica.
c) Aplikacijski okviri – Koristi se za razvoj aplikacija za specifične operacijske sustave.

d) Okviri za upravljanje sadržajem (eng. *Content Management*, CM) – Kombinacija web aplikacijskog okvira i sustava za upravljanje sadržajem (eng. *Content Management System*, CMS). Koristi se za razvoj softvera za upravljanje web sadržajem.
e) Multimedijski okviri – Okviri korišteni za upravljanje i razvoj multimedijskog softvera.

U ovom radu pažnja će biti usmjerena na web aplikacijske okvire.

2.2. Laravel

Laravel je web aplikacijski PHP okvir koji je temeljen na Symfony programskom okviru. On je jedan on najpopularnijih PHP programski okvira uz Symfony, Zend i CodeIgniter. Kreator Laravela je Taylor Otwell koji ga je razvio s ciljem izrade web rješenja praćenjem MVC (eng. *Model-View-Controller,* Model-Pogled-Kontroler) arhitekture koja je trebala predstavljati alternativu CodeIgniter okviru. Prva inačica Laravela objavljena je 9. lipnja 2011. godine. Unatoč tome što je relativno novi okvir popularnost mu je značajno brzo porasla te je i dalje u porastu. Trenutna inačica Laravela je 5.4 koja je izdana u siječnju 2017. godine.

Laravel pruža jednostavan i brzi *routing engine* (jednostavan sustav povezivanja ruta aplikacije s kontrolerima), *service container* (upravljanje ovisnosti klasa te za injektiranje ovisnosti, eng. *Dependency Injection*), Eloquent tehnologija za objektno-relacijsko mapiranje podataka relacijskih baza (eng. *Object-relational-mapping*, ORM), migriranje nacrta (eng. *Schema*) baze podataka, robusno asinkrono izvođenje poslova (eng. *Queues*), razašiljanje događaja u stvarnom vremenu (eng. *Broadcasting*), *templating* (jednostavan sustav za kreiranje višekratno upotrebljivih predložaka) te mnogo drugo [2].

2.3. Prednosti i nedostatci PHP-a i Laravela

PHP je kroz godine došao na loš glas zbog velikog broja loše razvijenih web aplikacija i stranica te zaostataka u usporedbi s drugim razvijenijim jezicima. Glavni nedostatci u odnosu na druge programske jezike su [3]:

a) Sigurnost – PHP programski jezik je otvorenog koda te je dostupan svima. Ukoliko postoji greška u izvornom kodu pojedinac je može proučiti kako bi našao slabosti aplikacija i/ili stranica razvijenih putem njega.

b) Nije pogodan za razvoj velikih aplikacija – PHP kod je teško za održavati, prvenstveno zbog njegovog nedostatka modularnosti.
c) *Weak Type* – Nema eksplicitnog zadavanja tipa. PHP sam interpretira vrijednosti što može rezultirati greškama. Primjer bi bio usporedba dva niza simbola, gdje se nizovi implicitno pretvaraju decimalni tip broja(eng. *Float*) te uspoređuju. Iako ta dva niza simbola nisu jednaka, ona mogu imati jednaku decimalnu vrijednost te rezultirati logičkom istinom.

Posljedica toga je bilo masovno prelaženje na druge programske jezike kao Ruby i Python. Kako ti jezici nisu bogati web značajkama kao PHP kreatori tih jezika su morali rekreirati osnovne gradbene blokove kao što su klase, reprezentiranje HTTP upita i odgovora. Rekreiranjem tih osnovnih blokova izbjegle su se greške PHP-a koji je bio pisan bez ikakvih temelja.

Unatoč tome PHP kroz zadnjih nekoliko godina opet počinje značajno rasti zbog novih značajki kao što su *closures* (funkcije koje se mogu spremiti kao varijable te nasljeđivati varijable iz lokalnog prostora putem use ključne riječi) i *traits* (mehanizam za omogućavanje korištenja metoda specifične klase unutar druge klase bez potrebe njenog nasljeđivanja/produžavanja, PHP klase su ograničene na mogućnost nasljeđivanja/proširenja samo jedne klase) te Composer upravitelja paketa.

Nakon velikog broja godina u kojima su ljudi razvijali njihov vlastiti pristup rješavanju uobičajenih zadataka poput upravljanja zahtjevima i odgovorima (na i od servera) programski okviri su preuzeli drukčiji pristup u kojemu se razvijaju komponente koje se mogu koristit u različitim projektima bez obzira na temelje. Jedan od njih je bio Symfony projekt koji je usvojio te principe kako bi se razvio čvrsti, ali i fleksibilni te testabilni HTTP temelj za razvoj PHP web rješenja. Uz Drupal i phpBB, Laravel je jedan od mnogi *open source* projekata (projekti čiji je izvorni kod dostupan javnosti na uporabu) koji je preuzeo HTTP temelje Symfony okvira. Osim HTTP okvira od Symfonya je preuzeo i mnoge druge manje komponente, ali i biblioteke kao što su SwiftMailer (za jednostavno slanje E-pošte), Carbon (formatiranje datuma i vremena) te mnoge druge.

Laravel se od drugih programskih okvira razlikuje i po tome što prihvaća nove značajke PHP te s time zahtijeva relativno novije inačice (najmanja inačica PHP 5.4). U prošlosti drugi programski okviri su gradili potporu za starije inačice PHP-a kako bi održali inverznu

kompatibilnost što dulje moguće. Međutim takav pristup značio je da ti isti okviri neće moći iskoristiti nove prednosti novih PHP inačica što bi rezultiralo ometanjem razvoja PHP-a.

Neke od tih značajka su:

a) **Imenski prostor** (eng. *Namespace*) - Omogućuje programerima izbjegavanje imenskih konflikta koje se mogu pojaviti ukoliko dvije ili više funkcija ili klasa imaju isto ime. U PHP-u imenski prostor je odvojen obrnutom kosom crtom koja predstavlja putanju unutar direktorija u skladu s PSR-4 konvencijom („<?php namespace Illuminate\Database\Eloquent ... ?>“). Kako bi se koristio kod iz drugog imenskog prostora koristi se *use* ključna riječ („use Illuminate\Database\Eloquent\Model“). Također prednost je i to što se uvedenim klasama može dodijeliti pseudonima (eng. *Alias*) kojim se izbjegava konflikt klasa iz drugog ili globalnog imenskog prostora korištenjem *as* ključne riječi nakon *use* izraza („use Foo\Bar as FooBar“).
b) **Anonimne funkcije** - Funkcije koje nemaju ime. U Laravelu, često se koristi u rutama (pr. „Route::get('/', function(){ return "Hello World."})“).
c) **Sučelja** (eng. *Interface*) - Specificiranje metoda koju klasa, koja implementira sučelje, mora implementirati.
d) **Preopterećenje** (eng. *Overloading*) - Omogućava pozivanje metoda koje nisu eksplicitno definirane u klasi. Takvi pozivi se obrađuju „__call“ metodom u klasi koja onda pokušava parsirati naziv kako bi se pokrenula jedna ili više poznatih metoda. Primjer „whereUsernameOrEmail($name, $email)“ pokreće "->where('username', $username)->orWhere('email', $email)“.
e) **Kraća sintaksa nizova** - PHP 5.4 uvodi kraću sintaksu. Umjesto „array('a'=>array(1,2,3,4))“ može se koristiti „['a'=>[1,2,3,4]]“.

Laravel kao programski okvir ima značajno više prednosti nego nedostataka. U usporedbi s WordPress, Drupal i Magento platformama, Laravel je bolje dugoročno rješenje unatoč mnogim značajkama koje navedene platforme pružaju zbog toga što Laravel pruža mnogo bolju dugoročnu potporu (eng. *Long Term Support*, LTS) te je aplikaciju razvijenu u njemu puno lakše za prilagoditi novim specifikacijama. U usporedbi s programskim okvirima poput Symfony, Yii i CodeIgniter glavna značajka prednosti bi bila u njegovoj značajnoj jednostavnosti i intuitivnosti naspram navedenih okvira.

Laravel projekt ima jako široku zajednicu iza sebe koja mu pruža stalnu potporu i poboljšanja. Zbog doprinosa velikog broja ljudi njegov kod je dobro organiziran što osigurava čvrstu provedbu najbolje prakse pisanja koda.

Laravel, kao programski okvir, nema mnogo loših strana, ali programer se može susresti s poteškoćama pri nadogradnji inačice aplikacije (npr. nadogradnja s inačice 4.x na 5.x). Nadogradnja s manje glavne inačice na novu je bitna jer novije inačice donose mnoga poboljšanja postojećeg okvira kao i kompatibilnost s novim alatima, ali to predstavlja i problem jer novije inačice većinom sadrže promjenu strukture okvira, određenih metoda pa sve do korištenja veće inačice samog PHP-a što rezultira zastarjelošću moguće korištenih metoda unutar aplikacije. Osim nadogradnja s glavne manje na veću inačice, problem se može pojaviti i pri nadogradnji kroz pod-inačice (npr. 5.3 na 5.4) iz razloga što se struktura okvira može promijeniti, ali u blažem obliku.

Nadogradnja inačice može predstavljati problem jer zahtijeva vrijeme i pažnju programera, ali ukoliko se održava redovno ipak će pružiti više koristi nego štete.

Za jedan od nedostatak se može uzeti i potreba za potrošnjom značajne količine vremena kako bi se savladalo njegovo korištenje na efikasan i efektivan način (brži tijek programiranja te pravilno strukturiranje koda) te potreba za stalnim učenjem kako dolaze nove inačice (nova inačica u pravilu dolazi svaki šest mjeseci).

3. PRIPREMA ZA RAZVOJ I OSNOVNI KONCEPTI LARAVEL WEB RJEŠENJA

Ovo poglavlje se bavi opisivanjem postupka pripreme i konfiguriranja Laravel aplikacija, sigurnosnih problema, MVC arhitekturalnog uzorka i objektno-relacijsko mapiranje koji čine temelj razvoja Laravela aplikacija.

3.1. Instaliranje potrebnih alata

Kako bi se započeo razvoj aplikacije potrebno je da sustav ispunjava određene preduvjete. To su (za Laravel 5.3) postojanje [4]:

a) PHP 5.6.4 ili veće inačice
b) OpenSSL PHP ekstenzije
c) PDO PHP ekstenzije
d) MBstring PHP ekstenzije
e) Tokenizer PHP ekstenzije
f) XML PHP ekstenzije

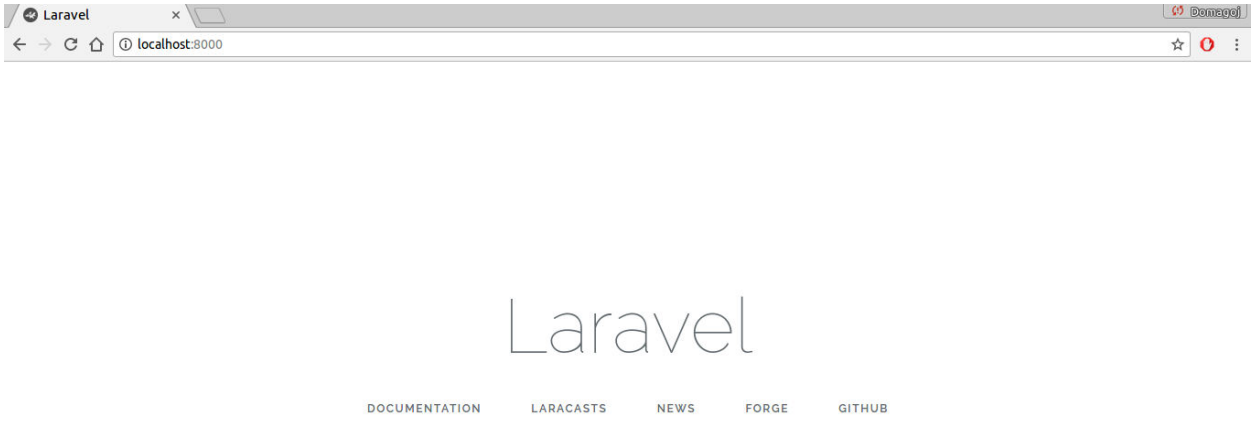
Također potrebno je instalirati određene resurse koji su preduvjet za korištenje programskog okvira. To su Composer upravitelj paketa i Node.js. Composer je potreban zato što pomoću njega Laravel pribavlja sve pakete koje su mu potrebna za ispravan rad, ali i koristi kad se želi u projekt na jednostavan način dodati i neke druge, specifične za projekt, pakete. Node.js nije nužan za rad samog programskog okvira ali on pruža mogućnost instalacije NPM upravitelja paketa koji se koristiti za lakše pribavljanje biblioteka koje će aplikacija koristit na klijentskoj strani te kompajliranje određenih Css i Javascript datoteka.

Nakon što je Composer instaliran, potrebno je otvoriti terminal te ući u direktoriju u kojem se želi instalirati novu Laravel aplikaciju. Aplikacija putem terminala može se instalirati naredbom

„composer create-project <željena_inačica> laravel/laravel <naziv_projekta>“ (primjer: „composer create-project laravel/laravel 5.3. apartmentsBooking“).

Za instaliranje NPM upravitelja paketa, potrebno je iz terminala ući u korijenski direktorij nove aplikacije i pokrenuti naredbu „npm install“.

Kada je aplikacija instalirana, da bi se pokrenula na lokalnom serveru, potrebno je pokrenuti naredbu „php artisan serve“. Kada je server pokrenut, ako se u pretraživač unese "http://localhost:8000/" bit će prikazana početna stranica novo-instalirane aplikacije (Sl. 3.1.).



Sl. 3.1. Početna stranica nove Laravel aplikacije

3.2. Konfiguriranje aplikacije

Konfiguriranje Laravel aplikacije obavlja se u „.env“ datoteci koja se nalazi u korijenskom direktoriju aplikacije. Za početak potrebno je postaviti:

a) „APP_KEY“ - Ključ za kriptiranje. Ukoliko aplikacija nije instalirana putem Composer-a ili Laravel Installer-a ključ generira se putem naredbe „php artisan key:generate“. Ovaj niz od 32 simbola osigurava kriptiranje podataka sesije i drugih podataka.
b) „DB_CONNECTION“ - Ovom varijablom određuje se baza podataka, to jest konekcija koja se želi koristiti. Predefinirane konekcije su mysql, sqlite i pgsql (postgreSql). U ovom primjeru za konekciju će biti odabran mysql.

c) „DB_DATABASE“, „DB_USERNAME“ i „DB_PASSWORD“ - naziv, korisničko ime i lozinka za ostvarivanje konekcije na bazu podataka.
d) „CACHE_DRIVER“ i „QUEUE_DRIVER“ - Web aplikacija za rezerviranje apartmana koristit će Redis za predmemoriranje podataka i implementaciju reda čekanja pa se ove vrijednosti postavljaju na "redis".
e) „MAIL_DRIVER“, „MAIL_HOST“, „MAIL_USERNAME“ i „MAIL_PASSWORD“ - Web aplikacija za rezerviranje apartmana koristit će smtp („MAIL_DRIVER“) za slanje E-pošte. Ostali parametri ovise o usluzi putem koje se E-pošta šalje.
f) „STRIPE_KEY“ i STRIPE_SECRET - Stripe ključevi za implementaciju plaćanja u aplikaciji.
g) „GMAP_KEY“ - API ključ za implementaciju Google karte.

Dodatne postavke, ali ne i obvezne programer može izvršiti u „config“ direktoriju (pr. „config/app.php“ gdje postavlja informacije o aplikaciji kao što su naziv, vremenska zona, lokalizacija i drugo).

3.3. Organizacijska struktura aplikacije

Osnovna struktura aplikacije sadrži sljedeće direktorije [5]:

a) „app“ direktorij - Direktorij u kojemu se nalazi osnovni kod razvijene aplikacije kao što su klase, kontroleri, *middleware* (mehanizam za filtriranje zahtjeva klijenta prema serveru) i drugo.
b) „bootstrap“ direktorij - U sebi sadrži datoteke za povezivanje (eng. *Bootstraping*) ključnih komponenti Laravela i konfiguriranje automatskog učitavanja (Eng. *Autoloading*). Također sadrži „cache“ direktorij s „route“ i „service cache“ datotekama u koje su spremljeni predmemorirani podatci aplikacije.
c) „config“ direktorij - Sadrži konfiguracijske datoteke aplikacije (npr. „app.php“ - općenite postavke aplikacije, „database.php“ - postavke konekcije na bazu podataka).
d) „database“ direktorij - Sadrži u sebi generirane migracije, seed-ove (mehanizmi ispunjavanja tablica baze podataka testnim objektima) i tvornice (eng. *factory,* koristi se za generiranje testnih objekata).
e) „node_modules“ - Aplikacija ga sadrži ukoliko je izvršena konzolna naredba „npm install“. Ovaj direktorij sadrži poddirektorije, to jest instalirane module (biblioteke) kao što su *Bootstrap* (frontend programski okvir - programski okviri za razvoj klijentske strane).

f) „public“ direktorij - Sadrži „index.php“ datoteku koja predstavlja ulaznu točku aplikacije (registriranje *Autoloader*-mehanizam automatskog učitavanja datoteka; povezuje aplikaciju - *bootstraping*). Također u njemu se nalaze svi javni resursi kao što su medijske, CSS i Javascript datoteke.
g) „resources“ direktorij - Sadrži sve poglede (okvirne i parcijalne) te ne kompajlirane resurse kao što su Less, Sass ili Javascript.
h) „routes“ direktorij - Sadrži sve definirane rute aplikacije. Uključuje „web.php“ (rute koje koristi statefull aplikacija), „api.php“ (rute za stateless RESTful API aplikacije) i „console.php“ (konzolne naredbe temeljene na *closure* i anonimnim funkcijama) datoteke.
i) „storage“ direktorij - Sadrži kompajlirane Blade predloške, predmemorirane datoteke i datoteke sesije te ostale datoteke generirane od strane Laravela. Sastoji se od „app“ (pohrana datoteka geniriranih od strane aplikacije), „framework“ (pohrana datoteka generiranih od strane Laravela) i „logs“ (zapisi aplikacije, npr. zapisi o greškama prilikom kompajliranja) direktorija.
j) „ tests“ direktorij - Sadrži testne skripte (pr. PHPUnit skripte).
k) „vendor“ direktorij - Sadrži Composer (aplikacijske) zavisnosti.

3.3.1. App direktorij

Najveći dio aplikacije sadržava App direktorij. Ovaj se direktorij predefinirano nalazi pod „App“ imenskim prostorom te ga Composer automatski učitava koristeći PSR-4 standard automatskog učitavanja.

App direktorij sadrži poddirektorije kao što su „Console“ (prilagođene Artisan naredbe), „Http“ (sadrži kontrolere, *middleware-e* i *form request*-ove; *Form Request* je mehanizam validacije i verifikacije zahtjeva) i „Providers“ (sadrži davatelje usluga aplikacije, eng *Service Providers*).

Dakako, pri korištenju Artisanovih make konzolnih naredbi (pr. „make:job“) dolaze do pojave novih direktorija (pr. „app/Jobs“).

Svi početni i Artisan generirani direktoriji su [5]:

a) „Console“ - Sadrži prilagođene Artisan naredbe kreirane putem konzolne naredbe "make:command" i Kernel.php u kojem se registriraju Artisan naredbe i definiraju planirani zadatci (eng. *scheduled tasks*).

b) „Events“ - Kreira se kada se pokrene konzolna naredba „event:generate“ ili „make:event“. Sadrži klase događaja (eng. *Events*). Događaji se koriste kako bi se aplikaciju upozorilo da odvijanje određenog događaja.
c) „Exceptions“ - Sadrži upravljač iznimkama (definira način obrade i zapisivanja iznimki) te se u njemu mogu definirati prilagođene iznimke aplikacije.
d) „Http“ - Sadrži kontroler, *middleware* i *form request* . U njemu se nalazi najveći dio logike aplikacije.
e) „Jobs“ - Kreira se kada se pokrene konzolna naredba „make:job“. Ovaj direktorij pohranjuje zadatke aplikacije za izvođenje u redu čekanja (eng. *Queueable Jobs*). Zadaci u redu čekanja mogu biti pokrenuti asinkrono ili sinkrono.
f) „Listeners“ - Kreira se kada se pokrene konzolna naredba „event:generate“ ili „make:listener“. Sadrži klase za upravljanje događajima. Slušatelji događaja (eng. *Event Listeners*) primaju instancu događaja te obavljaju određenu logiku kao odgovor na pokrenuti događaj.
g) „Mail“ - Kreira se kada se pokrene konzolna naredba „make:mail“. Sadrži klase koje predstavljaju poruke koje aplikacija šalje.
h) „Notifications“ - Kreira se kada se pokrene konzolna naredba „make:notification“. Sadrži obavijesti koje šalje aplikacija pri odvijanju određenih događaja. Omogućuje slanje obavijesti putem E-pošte, SMS-a, spremanjem u bazu ili Slacka (aplikacija za dopisivanje).
i) „Policies“ - Kreira se kada se pokrene konzolna naredba „make:policy“. Sadrži klase načina autorizacije (eng. *Authorization Policy*) koje se koriste za određivanje dozvole provođenja određene akcije nad određenim resursom.
j) „Providers“ - Sadrži sve davatelje usluga (eng. *Service Providers*) aplikacije. Davatelji usluga povezuju aplikaciju putem povezivanja usluga sa spremnikom za uslugu (eng. *Service Container*), registriranjem događaja ili obavljanjem nekih drugih aktivnosti u svrhu pripreme aplikacije na nadolazeće zahtjeve.

3.4. MVC uzorak dizajna

MVC (eng. *Model-View-Controller*) je arhitekturalni uzorak dizajna, formulirana 1970. godine od strane Trygve Reenskauga, čija je svrha odvajanja reprezentacije resursa od metoda koje se izvršavaju nad tim resursima. MVC je originalno bio namijenjen za izradu desktop aplikacija osobnih računala (GUI aplikacije, eng. *Graphical User Interface* - grafičko korisničko

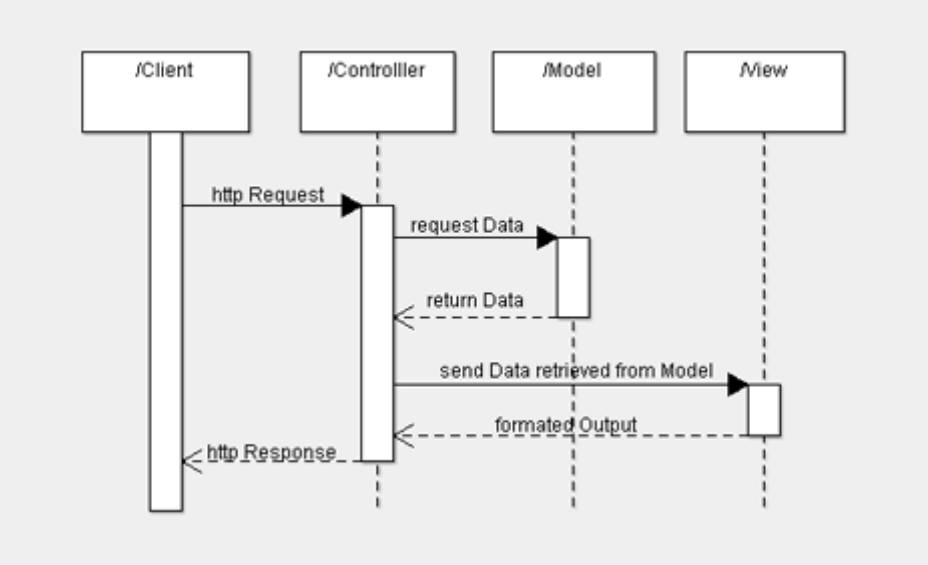
sučelje), ali se tokom vremena prilagodio za razvoj web, mobilnih te drugih aplikacija. Ovaj uzorak je došao do velike popularnosti u razvoju web stranica ili aplikacija zato što programerima omogućuje razvijanje klijentske i serverske strane web stranica ili aplikacije bez međusobnog ometanja.

Osnovni razlog razvoja ovog uzorka opisuju dva programska načela, a to su razdvajanje odgovornosti (eng. *Separation of Concernes*) i DRY (eng. *Don't Repeat Yourself*, Nemojte se ponavljati).

Razdvajanje odgovornosti je načelo koje govori da različiti aspekti aplikacija trebaju biti neovisni. Za primjer se može uzeti poslovna logika aplikacije i sučelje koji predstavljaju zadatke. Ti zadatci trebaju biti neovisni u smislu da je njihov kod međusobno odvojen. To znači da ukoliko jedan od zadataka bude zahtijevao prilagodbu, on neće utjecati na drugi.

DRY načelo koje govori da treba izbjegavati ponavljanje koda, te ga pisati, ukoliko je moguće, na način da se može višekratno iskoristit. Pri razvijanju kompleksnih aplikacija, one se dijele u različite komponente i podkomponente pa sve do razine koja predstavlja jednu odgovornost. Ta odgovornost sastoji se od znanja i reprezentacije, gdje znanje predstavlja dio koda, to jest kako je nešto postignuto, a reprezentacija varijablu koja je referenca na to znanje. Ukoliko se za primjer uzme spajanje na bazu podataka, znanje bi bila klasa koja inicijalizira spajanje na bazu, a reprezentacija varijabla pomoću koje se komunicira s bazom. Ovaj primjer prati DRY načelo na način ukoliko se promjene podatci za spajanje treba se mijenjati klasu, a ne njegovu reprezentaciju.

Princip rada MVC-a se temelji na međusobnoj komunikaciji modela, pogleda i kontrolera. Na slici 3.1. se može vidjeti tijek izvođenja manipulacije [6].



Sl. 3.2. Princip rada MVC-a

Slika opisuje korisnički zahtjev na server koji preuzima specifični kontroler. Taj kontroler obavlja određenu manipulaciju modela te putem rezultata manipulacije formatira odgovor, to jest pogled, koji vraća korisniku.

Model u MVC-u predstavlja objektnu reprezentaciju tablice baze podataka koji se koristi za CRUD (eng. *Create, Read, Update, Delete* - pisanje, čitanje, izmjena, brisanje) manipulaciju podataka. Model se također može opisati kao prolaz između zahtijeva i baze podataka. On predstavlja pivot MVC-a jer bez njega ne postoji veza između pogleda i kontrolera.

Pogled (eng. *View*) u MVC-u predstavlja mjesto gdje se dobiveni rezultat korisničkog zahtjeva integrira i prikazuje. Također on predstavlja vezu između korisnika i kontrolera (npr. korisnik klikom na gumb pokreće neku metodu kontrolera).

Kontroler u MVC-u koristi se za upravljanje podatcima koje korisnik unosi ili predaje te izvršava manipulaciju modela prema vrsti zahtjeva i/ili vraća odgovor korisniku putem pogleda. Kontroler je jedini dio MVC uzorka koji ima interakciju s korisnikom.

3.5. Objektno-relacijsko mapiranje i Eloquent

Objekto-relacijsko mapiranje je računalna tehnika kojom se podatci baze podataka predstavljaju kao objekti što omogućuje objektno orijentiranim programskim jezicima lakšu i bržu manipulaciju tim podatcima. Korištenjem objektno-relacijskog mapiranja dobiva se dojam korištenja objektnih baza podataka.

Objektno-relacijsko mapiranje je pogodno za korištenje uz MVC uzorak dizajna jer objekt koji reprezentira jedan ili više zapisa baze podataka predstavlja instancu modela te se sav kod vezan uz objekt nalazi u modelu čime se prati DRY načelo.

Negativna strana objektno-relacijskog mapiranja su nužnost pisanja koda za pretvaranje objekata i zahtijevane metode u sql upit i obrnuto ili učenje neke ORM biblioteke. Također, pošto objektno-relacijsko mapiranje apstrahira podatke i bazu podataka treba se paziti pri slaganju upita ili rada s upitima u petljama jer mogu biti vrlo zahtjevni. Treba se naravno uzeti u obzir činjenica da iako neke ORM biblioteke ili vlastita implementacija olakšavaju posao pri jednostavnijim upitima, neki kompleksniji upiti ipak mogu biti djelotvorniji ukoliko se pišu SQL upiti.

Eloquent je Laravelova ORM biblioteka putem koje se jedan model povezuje s jednom tablicom baze podataka. Laravel koristi gramatičke metode kako bi povezao kreiran model s tablicom baze podataka. Pri kreiranju modela treba se koristiti „CamelCase“ metoda. Ako se za primjer uzme kreirana tablica „inspiring_quotes“ Laravel bi zahtijevao da se model nazove „InspiringQuotes“ („CamelCase“ također može biti „inspiringQuotes“ ali se za imenovanje modela koristi metoda s prvim velikim slovom). Programer pri kreiranju modela ne treba ništa postavljati za povezivanje tog model s tablicom jer Laravel putem „snake_case“ gramatičke metode (razdvajanje riječi kod velikih slova s donjom crticom „_“ i dodavanjem „s“ na kraj) povezuje model s tablicom baze. Tako će Laravel „InspiringQuote“ model prevesti u „inspiring_quotes“ tablicu. Neke od osnovnih Eloquent metoda su [7]:

a) „all“ metoda - Vraća sve objekte modela, primjer „$users=User::all()“ kao rezultat vraća kolekciju svih korisnika. Važno je znati da vraća „Eloquent kolekciju modela“, a ne niz objekata.
b) „get“ metoda - Koristi se za dohvaćanje svih objekata kada se koriste neki uvjeti, primjer „$users=User::where('sex','m')->get()“ kao rezultat vraća kolekciju korisnika muškog roda.Važno je znati da vraća „Eloquent kolekciju modela“, a ne niz objekata.
c) „first“ metoda - Vraća samo prvi zapis kao rezultat, primjer „$user=User::first()“ vraća prvog korisnika. Rezultat je objekt.
d) „find“ metoda - Vraća logičku istinu ili laž (eng. *true or false*) ovisno o postojanju rezultata, primjer „$user=User::where('id',1)->find()“ kao rezultat vraća *true* ako postoji korisnik s "id"=1 a u suprotnom *false*.

e) „findOrFail“ i firstOrFail metoda - Ove dvije metode su zapravo iste te se koristi za provjeru postojanja zapisa u tablici, primjer „$user=User::where('name','John')->findOrFail()“ za rezultat vraća prvog korisnika s imenom „John“ ili vreća Eloquent iznimku „ModelNotFoundException“. Ukoliko se metoda koristi s uvjetom na primarnom ključu koristi se „$user=User::findOrFail(1)“.

f) „where“ metoda - Metoda kojom se postavlja uvjet. Za primjer se može pogledati "točka 2". Oblik joj je „where(<atribut_modela>,<relacijski_operator>,<vrijednost>)“, iako se u slučaju „==“ može koristiti skraćeni oblik „where(<atribut_modela>,<vrijednost>)“.

g) „take“ metoda - Metoda koja se koristi kako bi se ograničio broj rezultata kojih se žele dobiti, primjer „$users=User::take(10)->get()“ kao rezultat vraća kolekciju prvih 10 korisnika.

h) „orderBy“ metoda - Metoda kojom se organizira rezultat. Oblik joj je „orderBy(<atribut>,<'asc' ili 'desc'>)“, za primjer „$users=User::where('sex','m')->orderBy('name', 'desc')->get()“ za rezultat će se dobiti kolekcija korisnika muškog roda poredanih silazno po imenu.

i) „chunk“ metoda - Metoda kojom se segmentiraju dobiveni rezultati te se predaju anonimnoj funkciji za daljnju obradu, primjer „User::chunk(10, function($users){...})“ gdje „$users“ varijabla predstavlja kolekciju prvih deset korisnika od ukupnog broja na kojem će se primijeniti određena akcija u anonimnoj funkciji.

j) „cursor“ metoda - Koristi se kada se dohvaća velika kolekcija zapisa koja može zagušiti memoriju. Ova metoda vraća generator koji predstavlja kolekciju, a time čuva memoriju. Primjer „foreach(User::cursor as $user){...}“ u *foreach* petlju predaje generator koji reprezentira kolekciju umjesto stvarne kolekcije.

k) Agregatne metode - To su suma, količina i prosjek zapisa te najmanji i najveći zapis (eng. *count*, *sum*, *avg*, *max*, *min*). Primjer „$user=User::max(age)“ koristi *max* metodu te vraća starost najstarijeg korisnika dok bi *min* metoda vratila starost najmlađeg korisnika, *sum* metoda zbroj godina svih korisnika i *avg* metoda prosjek godina svih korisnika. Metoda *count* može primiti, ali ne treba, parametre te vraća količinu zapisa rezultata.

l) „save“ metoda - Metoda kojom se sprema novi ili izmjenjuje model, primjer „$user->save()“ sprema „$user“ model u bazu.

m) „update“ metoda - Koristi se za izmjenu više modela istovremeno, primjer „User::where('age', 20)->update(['name' => 'John'])“ svim korisnicima koji imaju 20 godina mijenja ime u „John“. Također umjesto niza atributa, u metodu se može predati

Laravelov „Request“ objekt ili programerov prilagođeni *form request* objekt čime se olakšava izmjena objekta.

n) „delete“ i „destroy“ metode - Obje metode služe za brisanje modela. Razlika je u tome da *delete* metoda treba prvo dohvaćen objekt na kojem će se izvršiti (pr. „$user->delete()“) dok se *destroy* metoda prima primarni ključ nekog objekta koji onda briše iz baze („User::destroy(1)“). Također *destroy* metoda može primiti i niz ključeva (pr. „User::destroy([1, 2, 3])“).

Značaj Eloquenta u razvoju Laravel web aplikacija je u tome što programerima znatno olakšava posao manipuliranja bazom podataka. Sve što programer treba napraviti je definirati veze između modela koji će se koristiti te ih povezati s pripadajućim tablicama. Nakon toga omogućena je uporaba Eloquent metoda čime se znatno ubrzava oblikovanje upita te se postiže ljepši i čitljiviji kod (izbjegavanje pisanja sql upita).

3.6. Sigurnost Laravel aplikacija

Web aplikacije razvijene putem Laravel programskog okvira jednako su podložne zlonamjernim napadima kao i sve druge web aplikacije. Unatoč tome što su aplikacije podložne napadima, Laravel pruža mehanizme i smjernice koje će pomoći pri osiguranju aplikacije od napada.

Najčešće vrste napada s kojima se web aplikacije susreću su [8]:

a) SQL injekcija - Slanje SQL koda kao parametra zahtjeva (pr. „;DROP TABLE <naziv_tablice>;“, obavit će se dva upita na bazu te bi drugi mogao obrisati tablicu) koji se direktno dodaje u SQL upit aplikacije bez modificiranja.
b) Presretanje komunikacije sa serverom - Ukoliko se napadač nalazi na istoj mreži koju žrtva koristi za komunikaciju sa serverom i ako se komunikacija odvija HTTP-om, napadač može presresti informacije koje se razmjenjuju sa serverom jer su informacije u običnom tekstualnom obliku.
c) Slanje izmijenjenih formi - Dodavanje dodatnih HTML polja forme s vrijednostima koje se onda šalju na server.
d) *Cross-site* napad lažiranjem zahtjeva (CSRF) - Slanje lažiranih zahtjeva koji izgledaju originalno te kao da dolaze od autorizirane osobe.
e) *Cross-site* Scripting (XSS) - Slanje skripti (Javascript skripte) koje aplikacija sprema te prikazuje drugim korisnicima.

a) Ukoliko se koristi Eloquent ili Laravelov graditelj upita (eng. *Query Builder*), aplikacija se štiti od SQL injekcija zato što obje metode u pozadini koristi PDO (PHP Data Objects) spajanje i komunikaciju s bazom. PDO koristi takozvane pripremljene izjave (eng. *Prepared Statements*) koje popunjava parametrima iz zahtjeva, ali tek kada su parametri analizirani i ovjereni. Unatoč ovome, pri korištenju Laravelove „DB::raw()“ metode programer piše čisti SQL upit te mora sam ovjeriti parametre upita ukoliko želi zaštitit aplikaciju.

b) Jedini način zaštite od presretanja komunikacije sa serverom je korištenje HTTPS za komunikaciju (Potrebno je imat SSL certifikat, skup datoteka kojima se povezuje enkripcijski ključ s organizacijom). Ukoliko postoji instalirani SSL certifikat na serveru, Laravel omogućuje preusmjeravanje s „http://“ adresa na „https://“ (Sl. 3.3.). To se obavlja „secure“ metodom koja provjerava gdje je zahtjev upućen, te ukoliko je upućen „http://“ ruti preusmjeriti na „https://“ inačicu rute kojoj se zahtjev šalje.

```
Route::filter('/', function() {
        if ( ! Request::secure()){
                return Redirect::secure(URI::current());
        }
});
```

Sl. 3.3. Preusmjeravanje s „http“ na „https“ rutu

c) Laravel pruža zaštitu od slanja izmijenjenih formi putem „$guarded“ svojstva u modelima. To svojstvo definira atribute modela koji se ne mogu izmijeniti zahtjevom.

d) Zaštita od CSRF napada vrši se putem CSRF oznake koja se šalje uz svaku formu te provjerava usporedbom s CSRF oznakom generiranom za korisničku sesiju.

e) Korištenjem Blade-a za unos PHP parametara u pogled (korištenje {{$parametar}}) koji se prikazuje korisnicima štiti aplikaciju od slanja skripti zato što parsira, to jest preskače HTML oznake (pr. „<script>…</script>“) putem kojih se može integrirati zlonamjerni kod u stranicu aplikacije. Ukoliko se koristi „{!!$parametar!!}“ Blade sintaksa za unos PHP parametra u pogled, treba se osigurati da su podatci zaštićeni jer Blade ne parsira HTML oznake parametra unutar njih.

4. RAZVOJ WEB APLIKACIJE ZA REZERVIRANJE APARTMANA

U ovom poglavlju opisat će se i objasniti postupak razvoja pojedinih modula web aplikacije za rezerviranje apartmana u svrhu pojašnjenja osnovnih koncepta i pogodnosti Laravel programskog okvira rokom razvoja web rješenja.

4.1. Implementacija autentikacije

Laravel, od inačice 5.2, dolazi s mogućnosti brze i jednostavne implementacije autentikacije korisnika. Pri instaliranju nove Laravel aplikacije kreira se model „User“ u „app“ direktoriju te „RegisterController“ (registriranje korisnika), „LoginController“ (autentikacija korisnika), „ForgotPasswordController“ (slanje E-pošte s linkom za resetiranje lozinke) i „ResetPasswordController“ (logika za resetiranje lozinke) kontroleri u "app/Http/Controllers/Auth" direktoriju.

Kako bi se implementirala autentikacija potrebno je pokrenuti konzolnu naredbu „php artisan make:auth“ koja generira „HomeController“ (za upravljanje zahtjevima nakon prijave, nije ga nužno koristiti) i potrebne poglede za prijavu i registraciju korisnika.

Kako bi implementirale rute za autentikaciju treba se dodati u „routes/web.php“ datoteku „Auth::routes();“ linija koda koja registrira sve rute za autentikacijski mehanizam.

Ukoliko programer želi može obrisati generirani „HomeController“ i koristit vlastiti kreirani kontroler te izmijeniti rute na koje se preusmjerava korisnik nakon registracije i prijave izmjenom zaštićene „$redirectTo“ varijable u „LoginController“ i „RegisterController“ kontrolerima.

Laravelova autentikacija dolazi uz nekoliko važnih i često korištenih metoda, a to su „user“, „id“, „auth“ i „check“ metode

Metoda „Auth::User()“ za rezultat vraća trenutno prijavljenog korisnika, a „Auth::id()“ vraća samo njegov primarni ključ „id“. Metoda „Auth::check()“ koristi za provjeru je li korisnik prijavljen u aplikaciji.

Middleware „auth“ omogućuje blokiranje pristupa korisnika na cijeli kontroler ili pojedinu metodu kontrolera ukoliko nije prijavljen u aplikaciju.

Ukoliko „User“ model i generirane migracije ne zahtijevaju daljnju prilagodbu može se izvršiti migracija tablica putem konzolne naredbe „php artisan migrate“ kojom će se kreirati „password_resets“ i „users“ tablice u bazi podataka. Nakon kreiranja tablica aplikacija sadrži potpuno implementirani mehanizam autentikacije [9].

4.2. Migracije i migriranje tablica baze podataka

U Laravelu migracije predstavljaju sheme tablica koje programer želi stvoriti. Migracije se mogu stvoriti na dva načina, a to su „make:migration“ i „make:model -m“ konzolne naredbe. „make:model -m“ način kreiranja migracije vezan je uz kreiranje modela pa će stoga biti opisan kasnije.

Kreiranje migracije s „make:migration“ načinom kreira migracijsku datoteku tablice u „database/migrations“ direktoriju. Ako se za primjer uzme kreiranje migracije za tablicu „reservations“, u konzoli se pokreće naredba „php artisan make:migration create_reservations_table“. Praks koja se koristi u imenovanju migracija je „create_<naziv_tablice>_table“.

Novo-generirana migracija u sebi će sadržavati „up“ i „down“ metode. Metoda „up“ koristi se za dodavanje nove tablice u bazu podataka, a „down“ metoda za brisanje stare inačice tablice iz baze podataka ukoliko postoji.

Obje metode koriste Laravelov „Schema Builder“ kako bi kreirali shemu nove tablice ili obrisali staru. U „up“ metodi koristi se „Schema::create("<naziv_tablice>, function(Blueprint $table){..})“ metoda za kreiranje tablice, a u down() metodi „Schema::dropIfExists('<naziv_tablice>')“ metoda.

Unutar „Schema::create()“ metode koristimo varijablu "$table" kako bi se kreirala željena shema tablice. Ako se za primjer uzme migracija „create_reservations_table“ (Sl. 4.1.), kreira se primarni ključ „id“, strani ključevi „apartment_id“ i „user_id“ te polja „check_in“, „check_out“, „info“, „status“, „created_at“ i „updated_at“.

```php
<?php

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

class CreateReservationsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('reservations', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('apartment_id')->unsigned();
            $table->foreign('apartment_id')->references('id')->on('apartments')
                    ->onDelete('cascade');
            $table->integer('user_id')->unsigned();
            $table->foreign('user_id')->references('id')->on('users')->onDelete('cascade');
            $table->date('check_in');
            $table->date('check_out');
            $table->text('info');
            $table->integer('status')->unsigned();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('reservations');
    }
}
```

Sl. 4.1. Migracija „reservations“ tablice

Metode za razvoj sheme tablice ovog primjera su [10]:

a) „$table->increments('<naziv_polja>')“ - Kreira polje koje predstavlja auto-inkrementalni primarni ključ.

b) „$table->integer("<naziv_polja>')“ - Kreira polje koje predstavlja cjelobrojnu vrijednost ekvivalentnu bazi podataka.

c) „$table->foreign('<naziv_polja')->references('<naziv_polja_tablice_koja_se_referencira>')-

>on('<naziv_tablice_koja_se_referencira>')->onDelete('cascade')“ - Kreira polje koje predstavlja strani ključ tablice koja se referencira (opcija „cascade“ označava brisanje svih zapisa tablice koje se odnose na referencirani zapis tablice ukoliko se taj zapis obriše).

d) „$table->date('<naziv_polja>')“ - Kreira polje koje predstavlja "date" vrijednost ekvivalentnu bazi podataka.
e) „$table->text('<naziv_polja>')“ Kreira polje koje predstavlja "text" vrijednost ekvivalentnu bazi podataka.
f) „$table->timestamps()“ - Kreira polja „created_at“ i „updated_at“. Ova polja se koriste za upisivanje vremenske oznake kreiranja i izmjene zapisa u bazi.

U svrhu razvoja ove web aplikacije, osim „create_reservations_table“ migracije, kreirane su:

a) „create_users_table“ - Migracija za tablicu korisnika generirana pri instalaciji Laravel aplikacije.
b) „create_password_resets_table“ - Migracija za tablicu korisnika generirana pri instalaciji Laravel aplikacije.
c) „create_apartments_table“ - Migracija za tablicu resetiranja lozinke generirana pri instalaciji Laravel aplikacije.
d) „create_photos_table“ - Migracija za tablicu u kojoj se spremaju informacije o slikama apartmana pohranjenima na serveru.
e) „create_infos_table“ - Migracija za tablicu u kojoj se spremaju informacije o davatelju usluge iznajmljivanja.
f) „create_carousels_table“ - Migracija za tablicu u kojoj se spremaju informacije o slikama početne stranice pohranjenima na serveru.
g) „create_reviews_table“ - Migracija za tablicu koja sadrži izjave korisnika o davatelju usluge iznajmljivanja.

Pomoću migracija programerima je omogućeno brzo i efikasno kreiranje i izmjena tablica baze podataka. Migracije su također, u slučaju postojanja suradnje na projektu, korisne jer programerima koji surađuju pruža lagan način za osvježavanje promjena u strukturi tablica koju su uveli drugi programeri na projektu.

4.3. Kreiranje i razvoj modela

Svrha modela je strukturiranje određenih podataka u specifičan objekt s njegovim metodama kako bi se učinkovito manipuliralo tim podatcima. Za kreiranje Laravelovog modela potrebno je pokrenuti konzolnu naredbu „php artisan make:model <naziv_modela>“. Također, pri imenovanju modela preporučuje se praćenje konvencije imenovanja koju koristi Laravel a to je korištenje „CamelCase“ s prvim velikim početnim slovom te korištenje jednine.

Kao što je prethodno spomenuto, pri kreiranju modela može se stvoriti i njegova vlastita migracija dodavanjem „-m“ u naredbu za kreiranje modela. Ako se za primjer uzme model „Reservation“, kreirat će se njegova migracija „create_reservations_table“.

Kada je model kreiran, ukoliko je praćena konvencija imenovanja, on će biti povezan s tablicom "reservations". Ako programer nije pratio konvenciju imenovanja, može sam postaviti željeni naziv tablice na koju se model povezuje korištenjem varijable „$table“. Osim naziva tablice Eloquent pretpostavlja za svaki model da njegova tablica ima stupac „id“ koji predstavlja primarni ključ te da je on auto-inkrementalna cjelobrojna vrijednost, kao i postojanje „created_at“ i „updated_at“ vremenskih oznaka. Ako stupac koji je primarni ključ ima drugačije ime, u modelu se može postaviti s varijablom „$primaryKey“. Ukoliko taj isti primarni ključ nije auto-inkrementalna cjelobrojna vrijednost, treba se uključiti u model varijabla „$incrementing“ te joj dati vrijednost logičke laži (eng. *false*). Za vremenske oznake ukoliko ih nema postavlja se varijabla „$timestamps“ na logičku laž.

Za primjer modela „Reservation“ ne treba dodati ni jednu od četiri navedene varijable, ali se preporučuje postavljanje „$table“ varijable.

Sljedeći korak u razvoju modela je definiranje kojim stupcima se želi omogućiti, onemogućiti ili sakriti grupni unos parametara (eng. *Mass Assignment* - automatsko dodavanje svih parametara HTTP zahtjeva modelu) od strane korisnika aplikacije. Potreba za zaštitom stupaca dolazi od mogućnosti slanja dodatnih HTTP parametara od strane zlonamjernog korisnika. Na primjer, korisnik šalje „is_admin“ parametar putem HTTP zahtjeva što može rezultirati kreiranjem novog korisnika s ulogom administratora.

Definiranje zaštite stupaca tablice u modelu obavlja se putem „$fillable“, „$guarded“ i „$hidden“ varijabli. Važno je za napomenuti da „$hidden“ svojstvo ne štiti stupac nego ga samo skriva pri serijalizaciji (postupak pretvaranja objekta ili neke strukture podataka u format pogodan za pohranu ili prijenos) modela u JSON format ili niz. Za model „Reservation“ koristi se samo „$fillable“ varijabla.

Zadnji korak u razvoju „Reservation“ modela je dodavanje njegovih veza s drugim modelima. Za ovaj model dodaje se dvije „jedan prema više“ veze a to su veze s modelima „Apartment“ i „User“ koje se dodaju na isti način. Primjer povezivanja s „User“ modelom:

a) U model „Reservation“ dodaje se metoda „user“ u kojoj se definira da svaka rezervacija pripada samo jednom korisniku, to jest instanci „User“ modela.
b) U model „User“ dodaje se metoda „reservation“ u kojoj se definira da svaki korisnik može imati više rezervacija, to jest instanci „Reservation“ modela.

Naravno postoje i drugi oblici veza kao što su „jedan prema jedan“ i „više prema više", ali se neće koristiti u svrhu razvoja ove aplikacije. Slika 4.2. prikazuje programski kod završenog „Reservation“ modela.

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Reservation extends Model
{
   protected $table = 'reservations';

        protected $fillable = [
                'apartment_id',
                'user_id',
                'check_in',
                'check_out',
                'info',
                'status'
        ];

   public function apartment() {
     return $this->belongsTo('App\Apartment');
   }

   public function user() {
     return $this->belongsTo('App\User');
   }
}
```

Sl. 4.2. Programski kod „Reservation“ modela

Osim ovih koraka moguće je i dodati specifične metode modela kao što su opsezi upita (eng. *Query Scopes*, prilagođene metode izrade upita, pr. definiranje metode upita za pronalazak zapisa s definiranim „id“ parametrom) te druge metode.

Kreiranje i razvoj modela na ovaj način znatno ubrzava cijeli proces i programeru omogućuje njegovu strukturnu i logičku razumljivost te efikasnu izmjenu logike u njemu.

Osim modela „Reservation“ u svrhu razvoja aplikacije izrađeni su „Apartment“, „Carousel“, „Info“, „Photo“, „Review“ i „User“ modeli.

4.4. Usmjeravanje zahtjeva

Laravelu pruža programeru znatno bržu i intuitivniju implementaciju usmjeravanja zahtjeva putem povezivanja putanje zahtjeva s kontrolerima te *closure* ili anonimnim funkcijama web aplikacije.

Kao što je prethodno napisano, usmjeravane se obavlja u tri datoteke unutar „routes“ direktorija [11]:

a) „web.php“ - Registriranje ruta koje koriste *statefull* aplikacije (iz razloga što je web aplikacije za rezerviranje apartmana *statefull* koristit će se ova datoteka za registraciju potrebnih ruta; *statefull* označava pamćenje klijentskog stanja na serveru, npr. Varijable sesije).
b) „api.php“ - Registriranje ruta koje koriste stateless RESTful API aplikacije (aplikacije bez pamćenja klijentskog stanja na serveru koje putem oznake i odgovarajuće putanje autentificiraju korisnika te vraćaju odgovarajuću reprezentaciju resursa).
c) „console.php“ - Registriranje konzolnih Artisan naredbi temeljenih na *closure* ili anonimnim funkcijama.

Metode omogućene u Laravelu za obradu isključivo jednog tipa HTTP zahtjeva su:

a) „Route::get($uri, $callback)“ - Upravljanje GET HTTP zahtjevima za dohvaćanje resursa ili stranice.
b) “Route::post($uri, $callback)“ - Upravljanje POST HTTP zahtjevima za spremanje resursa.
c) „Route::put($uri, $callback)“ - Upravljanje PUT HTTP zahtjevima za potpunu izmjenu resursa.
d) „Route::patch($uri, $callback)“ - Upravljanje PATCH HTTP zahtjevima za parcijalnu izmjenu resursa.
e) „Route::delete($uri, $callback)“ - Upravljanje DELETE HTTP zahtjevima za brisanje resursa.

f) „Route::options($uri, $callback)“ - Upravljanje OPTIONS HTTP zahtjevima za otkrivanje dopuštenih metoda nad resursom i/ili zahtjeva da mu se pristupi.

Na svim rutama unutar „web.php“ datoteke primjenjuje se „web“ *middleware* grupa. To znači da sve HTML forme koje su usmjerene na njih trebaju sadržavati polje CSRF oznake. CSRF oznaka štiti aplikaciju od napada lažiranjem zahtjeva tako što uspoređuje tu oznaku s CSRF oznakom koja se generira za svaku aktivnu korisničku sesiju. Pomoću njega se osigurava da aplikacija prima zahtjeve od isključivo samo autentificiranog korisnika bez mogućnosti izmjene tog zahtjeva od strane napadača.

Također, iz razloga što HTML forme ne podupiru PUT, PATCH i DELETE metode, treba u njima, ako se te metode koriste, biti dodano „method_field“ skriveno polje koje čija će se vrijednost koristit kao HTTP metoda zahtjeva.

Unutar URI-a mogu se koristiti takozvani zamjenski parametri (eng. *Wildcards*) unutar vitičastih zagrada (primjer „{id}“). Ti parametri se mogu proslijediti *closure* ili anonimnoj funkciji te kontroleru na korištenje.

Laravel putem zamjenskih parametara omogućuje i povezivanje s modelima (eng. *Model Binding* - putem injektirane varijable dohvaća se model te ga se injektira u rutu). Povezivanje modela može se obaviti eksplicitno putem „RouteServiceProvider“ klase (Sl. 4.3.) ili implicitno unutar pojedine metode.

```
public function boot(){
    parent::boot();

    Route::model('user', App\User::class);
}
```

Sl. 4.3. Primjer eksplicitnog vezanje modela u „RouteServiceProvider“ klasi

Eksplicitno vezanje rezultirat će injektiranjem „model“ modela u svaku rutu koja injektira "{user}" parametar (Sl. 4.4.).

```
Route::get('users/{user}', function($user){
    return $user;
});
```

Sl. 4.4. Anonimna funkcija vraća eksplicitno vezani „User“ model.

Implicitno vezanje vrši se na pojedinim rutama tako da se zamjenskom parametru definira tip (eng. *Type Hint*) unutar *closure* ili anonimne funkcije te kontrolerove metode (Sl. 4.5.).

```
Route::get('users/{user}', function(App\User $user){
        return $user;
});
```

Sl. 4.5. Anonimna funkcija vraća implicitno vezani „User“ model.

Postoje i metode koje nisu vezane uz jedan tip HTTP zahtjeva, a to su:

a) „Route::match($methodsArray, $uri, $callback)“ - Upravlja nizom definiranih HTTP zahtjeva na određenoj ruti.
b) „Route::any($uri, $callback)“ - Upravlja svim HTTP zahtjevima na određenoj ruti.
c) „Route::group($sharedAttributesArray, $callback)“ - Metoda kojom se registrira niz ruta koje dijele *middleware*, prefiks ili imenski prostor kako bi se izbjeglo njihovo ponavljanje na svakoj zasebno.
d) „Route::resource($resourceName, $controller)“ - Registriranje ruta za GET/HEAD, POST, PUT/PATCH i DELETE HTTP metode putem imena resursa za njegov kontroler.

Middleware se kod ruta može definirati kao jedan od atributa „$sharedAttributesArray“ niza unutar „group“ metode ili pomoću „middleware“ metode koja se veže na pojedinu rutu.

Ukoliko je implementiran Laravelov mehanizam autentikacije, ovdje se dodaje „Auth::routes()“ linija koda za registriranje ruta koje autentikacijski mehanizam koristi. Slika 4.6. prikazuje „web.php“ datoteku koja sadrži sve rute web aplikacije za rezerviranje apartmana.

```php
<?php

Route::get('/','InfoController@home');

Route::resource('apartments', 'ApartmentsController');

Route::resource('users', 'ManageUsersController');

Route::resource('reservations', 'ReservationsController',['except' => ['create','store']]);

Route::group(['prefix' => 'photos'], function() {
   Route::post('{id}', 'PhotosController@store');
        Route::delete('{id}', 'PhotosController@destroy');
        Route::post('carousel/add', 'PhotosController@storeCarousel');
        Route::get('carousel/{id}', 'PhotosController@editCarousel');
        Route::patch('carousel/{id}/update', 'PhotosController@updateCarousel');
        Route::delete('carousel/{id}', 'PhotosController@carouselDelete');
});

Route::group(['prefix' => 'reservations'], function() {
        Route::get('create/{id}', 'ReservationsController@create');
        Route::post('store/{id}','ReservationsController@store');
        Route::patch('confirm/{id}','ReservationsController@confirmReservation');
        Route::delete('decline/{id}','ReservationsController@declineReservation');
        Route::post('checkout/{id}','ReservationsController@payReservation');
        Route::post('cleanup','ReservationsController@reservationsCleanup');
});

Route::group(['prefix' => 'reviews'], function() {
        Route::get('create', 'ReviewController@create');
        Route::post('store', 'ReviewController@store');
        Route::get('/','ReviewController@index');
        Route::get('review/{id}', 'ReviewController@show');
});

Route::get('contact', 'InfoController@show');
Route::post('contact/inquiry', 'InfoController@sendInquiry');

Route::get('info/edit', 'InfoController@edit');
Route::post('info/edit/store', 'InfoController@store');

Auth::routes();
```

Sl. 4.6. Rute web aplikacije za rezerviranje apartmana

Sve registrirane rute mogu se vidjeti u konzoli putem naredbe „php artisan route:list“ koja prikazuje metodu, *middleware*, URI, ime i akciju (metoda kontrolera ako postoji) svake rute.

4.5. Kreiranje i razvoj filtera HTTP zahtjeva

U Laravelu, mehanizam za filtriranje HTTP zahtjeva upućenih aplikaciji je *middleware*. Laravel aplikacija pri kreiranju već sadrži neke *middleware* kao što je „auth“ (omogućuje pristup ruti samo prijavljenim korisnicima). Svaki *middleware* nalazi se unutar „app/Http/Middleware“ direktorija.

Kako bi se kreirao novi middleware koristi se konzolna naredba „php artisan make:middleware <naziv_middleware-a>“. Za primjer će se uzeti „Admin“ *middleware*. Pri pokretanju "php artisan make:middleware Admin" generira se klasa „Admin“ s „handle“ metodom u koju se unosi autentikacijska logika.

Za ovaj primjer želi se kreirati *middleware* koji će dopuštati zahtjev isključivo administratorima aplikacije. To se postiže tako da se u bazi podataka, za tablicu „users“, provjeri atribut „role“ trenutno prijavljenog korisnika i usporedi s „2“ koji predstavlja vrijednost za administratorsku ulogu. Ukoliko vrijednost „role“ atributa prijavljenog korisnika bude „2“ propušta se njegov zahtjev, a u suprotnom ga se preusmjerava na početnu stranicu (Sl. 4.7.).

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Support\Facades\Auth;

class Admin
{

    public function handle($request, Closure $next)
    {
        if ( Auth::check() && Auth::user()->isAdmin())
        {
            return $next($request);
        }
        return redirect('/');
    }
}
```

Sl. 4.7. „Admin“ *middleware*.

Kako bi se kreirani *middleware* koristio na ruti treba mu se dodijeliti ključ unutar „app/Http/Kernel.php“ datoteke. Ova datoteka sadrži tri niza u kojemu se može registrirati *middleware*, a to su [12]:

a) „$middleware“ - Niz u kojem se registrira *middleware* koji se primjenjuje na sve rute aplikacije.

b) „$middlewareGroups“ - Niz u kojem se registrira grupa *middleware*-a (pr. Laravelova „web“ grupa) s ključem.

c) „$routeMiddleware“ - Niz u kojem se registrira pojedini *middleware* s ključem (koristi se za registriranje „Admin“ *middleware*-a, Sl. 4.8.).

```
protected $routeMiddleware = [
        'auth' => \Illuminate\Auth\Middleware\Authenticate::class,
        'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
        'bindings' => \Illuminate\Routing\Middleware\SubstituteBindings::class,
        'can' => \Illuminate\Auth\Middleware\Authorize::class,
        'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
        'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
        'admin' => \App\Http\Middleware\Admin::class,
        'has_role' => \App\Http\Middleware\HasRole::class,
];
```

Sl. 4.8. „$routeMiddleware“ niz

Nakon što je *middleware* registriran može se koristiti „middleware“ metodom unutra „web.php“ datoteke ili pojedinog kontrolera. Metoda „middleware“ kao parametar prima ključ/klasu jednog ili niza više *middleware*-a. Ukoliko je prvi parametar ključ/klasa jednog *middleware*-a metoda može primiti i drugi parametar koji predstavlja niz ograničenja kao što su „only“ (*middleware* se koristi samo na definiranim metodama) i „except“ (*middleware* se koristi na svim metodama osim definiranih) atributi. Na slici 4.9. prikazano je korištenje „middleware“ metode u konstruktoru „ManageUsersController“ kontrolera.

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

Sl. 4.9. Uporaba „Admin“ *middleware*-a

4.6. Kreiranje i razvoj mehanizam validacije i autorizacije HTTP zahtjeva

U Laravelu, mehanizam validacije i autorizacije HTTP zahtjeva je *form request*. Kako bi se kreirao *form request* pokreće se konzolna naredba „php artisan make:request <naziv_zahtjeva>“. Za primjer će se uzeti kreiranje i razvoj „AddReservationRequest“ *form requesta*. Nakon pokretanja naredbe „php artisan make:request AddReservationRequest“ kreira se datoteka „app/Http/Requests/AddReservationRequest.php“ koja sadrži klasu „AddReservationRequest“ s dvije metode, a to su [13]:

a) „authorize“ - U ovoj metodi provjerava se ima li strana koja šalje zahtjev dozvolu za to.

b) „rules“ - Ova metoda vraća niz pravila koja se primjenjuju pri validaciji forme.

Za primjer „AddReservationRequest", unutar „authorize“ metode provjerava se je li korisnik prijavljen u aplikaciju. Ukoliko je prijavljen zahtjev će biti propušten, a u suprotnom će se vratiti „503“ HTTP odgovor (neautoriziran zahtjev). Slika 4.10. prikazuje „AddReservationRequest“ klasu.

```php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;
use Illuminate\Support\Facades\Auth;

class AddReservationRequest extends FormRequest
{
    public function authorize()
    {
        if (Auth::check()) {
            return true;
        }
        return false;
    }

    public function rules()
    {
        return [
            'date_range' => 'required'
        ];
    }
}
```

Sl. 4.10. „AddReservationRequest“ klasa.

Metoda „rules“ vraća niz pravila koje se koristi za validaciju parametara poslanog zahtjeva (forme). Za ovaj primjer postavlja se da parametar „date_range“ ne može biti prazan („required“), to jest imati vrijednost „“.

Najčešća korištena pravila za validaciju osim prethodno navedenog su [13]:

a) „date“ - Vrijednost polja mora biti obradiva od strane PHP funkcije "strtotime()".
b) „email“ - Vrijednost polja mora biti formatirana kao adresa E-pošte.
c) „file“ - Vrijednost polja mora biti uspješno prenesena datoteka.
d) „integer“ - Vrijednost polja mora biti cijeli broj.
e) „numeric“ - Vrijednost polja mora biti numerička.
f) „string“ - Vrijednost polja mora biti niz simbola.

Nakon što je *form request* kreiran može se koristiti u metodama kontrolera tako da varijabli „$request“ tipa „Request“ promijenimo tip u „AddReservationRequest“ (Sl. 4.11.).

```
public function store($id, AddReservationRequest $request){…}
```

Sl. 4.11. Primjena „AddReservationRequest“ *form request*-a u „store“ metodi „ReservationsController“ kontrolera.

Na ovaj način zahtjev će se, prije predaje metodi kontrolera na obradu, ovjeriti i autorizirati. Ovime programer izbjegava pisanje logike za autentikaciju i validaciju u kontroleru te ga, ukoliko je potrebno i/ili kompatibilno, može višekratno iskoristiti (u slučaju da se koristi na više mjesta, lakše je za izmijeniti logiku u odnosu na izmjenjivanje u svakoj metodi kontrolera zasebno).

4.7. Kreiranje i razvoj događaja i slušatelja događaja

Laravelov mehanizam događaja (eng. *Event*) i slušatelj događaja (eng. *Listener*) pruža jednostavnu implementaciju „Promatrač“ uzorka dizajna [14]. primjer promatrač uzorka bi bile dvije klase od koje jedna predstavlja klasu (nadalje subjekt) koju osluškuje druga klasa (nadalje slušatelj). Kako bi slušatelj osluškivao promjene subjekta, u subjektu se treba registrirati slušatelja(spremiti u npr. varijablu niza slušatelja). Kada se dogodi neka promjena stanja u subjektu, on putem određene metode obavještava sve slušatelje iz niza slušatelja o svojim promjenama tako da poziva metodu za obradu promjena svakog slušatelja te kao parametar prosljeđuje referencu na sebe (varijabla „$this“). Metoda za obradu promjena sadrži logiku koja se koristi za procesiranje promjena subjekta.

Laravelov mehanizam događaja i slušatelj događaja funkcionira na isti način, ali je pojednostavljen u korist programera.

Kako bi kreirali događaj i slušatelj događaja koriste se konzolne naredbe „make:event“ i „make:listener“, ali se preporučuje korištenje „EventServiceProvider“ klase za njihovo kreiranje.

Ukoliko se za kreiranje koristi „EventServiceProvider“ klasa, unutar „$listen“ niza definiraju se nizovi slušatelja događaja, gdje ključ svakog niza predstavlja događaj. U svrhu razvoja aplikacije potrebno je kreiranje „ReservationStatusHasChanged“ događaja i „NotifyUser“ slušatelj događaja (Sl. 4.12.).

```
protected $listen = [
    'App\Events\ReservationStatusHasChanged' => [
        'App\Listeners\NotifyUser',
    ],
];
```

Sl. 4.12. Definiranje „ReservationHasChanged“ događaja i „NotifyUser“ slušatelja događaja u „listen“ metodi „EventServiceProvider“ klase.

Nakon definiranja događaja i slušatelja događaja pokreće se konzolna naredba „php artisan event:generate“ koja stvara „app/Events“ i „app/Listeners“ direktorije (ukoliko već nisu kreirani) te u njih pohranjuje kreirane klase događaja i slušatelja događaja.

Novi kreirani događaj u sebi sadrži „__construct“ i „broadcastOn“ metode. Za primjer „ReservationStatusHasChanged“ događaja, dodaje se zaštićena „$reservation“ i „$status“ varijabla koje će vrijednost dobiti putem konstruktorske metode koja za parametre prima instancu „Reservation“ modela i logičku istinu ili laž (logička laž je zadana vrijednost). Metoda „broadcastOn“ se koristi za definiranje niza kanala (pr. Redis ključ vrijednost, Redis – ne relacijska baza podataka) u svrhu implementacije *publish/subscribe* modela komunikacije (paradigma komunikacije u stvarnom vremenu, pr. Laravel aplikacija emitira događaj na Redis ključ, a node.js server osluškuje taj ključ za novim promjenama te ih procesira). Za ovu aplikaciju neće se koristit „broadcastOn“ metoda.

Iz razloga što su „$reservation“ i „$status“ varijable zaštićene kreira se *getter* metoda (metode kojima druge klase pristupaju privatnim ili zaštićenim varijablama neke klase) „getEventParams“ koja vraća niz vrijednosti tih varijabli kao rezultat. Slika 4.13. predstavlja „ReservationStatusHasChanged“ klasu.

```php
<?php

namespace App\Events;

use App\Reservation;
use Illuminate\Queue\SerializesModels;

class ReservationStatusHasChanged
{
    use SerializesModels;

    protected $reservation;
    protected $status;

    public function __construct(Reservation $reservation, $status=false)
    {
        $this->reservation=$reservation;
        $this->status=$status;
    }

    public function getEventParams() {
        return [$this->reservation, $this->status];
    }

}
```

Sl. 4.13. „ReservationStatusHasChanged“ klasa.

Novi kreirani slušatelj događaja u sebi sadrži „__construct“ i „handle“ metodu. Za razvoj ove aplikacije konstruktorska metoda se neće koristiti. Metoda „handle“ predstavlja metodu koja reagira na događaj te obavlja određenu logiku. U ovom primjeru, za parametar prima instancu „ReservationStatusHasChanged“ događaja te putem „getEventParams“ metode instance dohvaća parametre događaja („$reservation“ i „$status“ varijable). Putem varijable „$status“ određuje se je li događaj bio odbijanje ili prihvaćanje rezervacije te se u skladu s time obavještava korisnik koji je kreirao rezervaciju. Obavještavanje korisnika odvija se putem Laravelovog „Notification“ mehanizma. Laravelove obavijesti biti će objašnjene kasnije u radu.

Također potrebno je naglasiti kako „NotifyUser“ slušatelj događaja implementira „ShouldQueue“ sučelje koje označava da se „handle“ metoda prije izvođenja stavlja u red čekanja (eng. *Queues*). Iz razloga što Laravel inačica 5.3. ne podržava, stavljanje slušatelja događaja na specifični red čekanja (omogućeno tek u inačici 5.4.) može se preopteretiti zadani red čekanja dodavanjem „queue“ metode u slušatelju događaja te promijeniti red čekanja na koji će se staviti. Ovo predstavlja zaobilaženje problema te se ne nalazi u službenoj dokumentaciji. Na slici 4.14. prikazan je „NotifyUser“ klasu.

```php
<?php

namespace App\Listeners;

use App\Events\ReservationStatusHasChanged;
use App\Notifications\ReservationApproved;
use App\Notifications\ReservationDeclined;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\QueueManager;

class NotifyUser implements ShouldQueue
{
    public function handle(ReservationStatusHasChanged $event)
    {
        $eventParams=$event->getEventParams();
        if($eventParams[1]){
            $eventParams[0]->user->notify(new ReservationApproved($eventParams[0]));
        }
        else{
            $eventParams[0]->user->notify(new ReservationDeclined($eventParams[0]));
        }

    }

    public function queue(QueueManager $handler, $method, $arguments) {
        $handler->push($method, $arguments, 'notifications');
    }
}
```

Sl. 4.14. „NotifyUser“ klasa.

4.8. Red čekanja

Laravel pruža ujedinjeni API za različite servise stavljanja u red čekanja kao što su Beanstalk, AmazonSQS, Redis i relacijske baze podataka [15]. Svrha reda čekanja je odgađanje izvođenja težih procesa za kasnije kako bi se ubrzala mogućnost odgovora web stranice ili aplikacije (asinkrono procesiranje) te postavljanje prioriteta izvođenja specifičnih procesa.

Web aplikacije za rezerviranje apartmana koristi Redis za implementaciju reda čekanja. Kako bi Laravel aplikacija koristila Redis potrebno je u „composer.json“ datoteku upisati, unutar „require“ objekta, "predis/predis": "^1.1" ("^1.1" predstavlja inačicu paketa koja se može izmijeniti) liniju koda (Sl. 4.15.) te izvršiti konzolnu naredbu „composer update“ putem koje se u aplikaciju instalira „predis“ biblioteku. Nakon toga potrebno je u „.env“ datoteci postaviti

varijablu „QUEUE_DRIVER“ na vrijednost „redis“. S ovim postupkom aplikaciji je omogućeno asinkrono izvođenje procesa.

```
"require": {
    "php": ">=5.6.4",
    "laravel/framework": "5.3.*",
    "intervention/image": "^2.3",
    "watson/active": "^2.0",
    "predis/predis": "^1.1",
    "laravel/cashier": "^7.0"
}
```

Sl. 4.15. Uključivanje „predis“ biblioteke u „require“ objekt „composer.json“ datoteke.

Laravel omogućava jednostavan način stavljanja poslova (eng. *Job*) i slušatelja događaja u red čekanja tako da programer jedino treba implementirati „ShouldQueue“ sučelje u klase poslova i slušatelja događaja. Razlika u mogućnostima stavljanja u red čekanja između poslova i slušatelja događaja je u tome što je poslovima omogućeno definiranje u koji red čekanja se treba stavit dok se svi slušatelji događaja stavljaju na „default“ red čekanja (moguće preopteretiti „queue“ metodom). Preporučuje se stavljanje poslova i slušatelja događaja u odvojene redove čekanja (poslovi i slušatelji događaja u aplikacijama većinom imaju različiti prioritet izvršavanja) .

Kako bi se pokrenuo slušatelj zadataka reda čekanja potrebno je pokrenuti konzolnu naredbu „php artisan queue:listen“ ili „php artisan queue:work“. Naredba „queue:listen“ pokreće proces izvršava jedan zadatak te uništava proces i tako za svaki zadatak u redu čekanja što rezultira većim opterećenjem procesora, ali zato oslobađa svu zauzetu memoriju. Naredba „queue:work“ pokreće proces i taj proces izvršava sve zadatke u redu čekanja. Pošto se „queue:work“ inicijalizirani proces ne uništava, opterećenje procesora je manje, ali zato, u slučaju izvođenja vrlo resursno-skupog zadatka, prije kraja izvršavanja zadatka treba osloboditi zauzetu memoriju. „queue:work“ proces se ne uništiva (stalno radi u pozadini) pa ukoliko dođe do promjena u kodu treba pokrenuti konzolnu naredbu „php artisan queue:restart“ koja završava trenutni zadatak, ukoliko ga ima, i restartira proces.

Uz navedene naredbe mogu se dodati i izborne opcije (zastavice, eng. *Flags*) kao „—tries“ (određuje koliko puta će se zadatak probati izvršiti u slučaju neuspjeha prije odustajanja i odbacivanja), „—memory“ (granica memorije u megabajtima na kojoj se proces restartira) i „—queue“ (koristi se za određivanje koji red čekanja se želi izvoditi ili, u slučaju potrebe izvođenja više redova, kojim prioritetom, pr. „--queue:high,low“ prvo će izvoditi zadatke iz „high“ reda čekanja, a tek onda iz „low“ reda čekanja).

Moguće je pokrenuti više slušatelja reda čekanja istovremeno. Tako pokrenuti slušatelji će izvršavati zadatke dok ih ima ili dok se konzola, u kojoj je naredba pokrenuta, ne zatvori. Zbog drugog slučaja postoje alati kao što je Supervisor (namijenjen prvenstveno za rad s redom čekanja) ili Cron zadatci (koristi se za zakazivanje izvođenja konzolnih naredbi).

4.9. Kreiranje i razvoj obavijesti

Laravel, od inačice 5.3., omogućuje slanje obavijesti putem E-pošte, SMS-a i Slack-a. Kako bi se kreirala klasa obavijesti potrebno je pokrenuti „make:notification“ konzolnu naredbu [16].

U svrhu razvoja web aplikacije za rezerviranje apartmana kreira se „ReservationApproved“ (obavijest o prihvaćanju rezervacije) i „ReservationDeclined“ (obavijest o odbijanju rezervacije) klase obavijesti. Za kreiranje „ReservationApproved“ klase pokreće se „php artisan make:notification“ konzolna naredba. Pokretanjem ove naredbe kreira se „app/Notifications“ direktorij, ukoliko već nije kreiran, i „ReservationApproved“ klasa unutar njega. Klasa „ReservationApproved“ u sebi sadrži „via“, „toMail“ i „toArray“ metode. Unutar „via“ metode definira se način slanja obavijesti (za potrebe ove aplikacije postavlja se „mail“ vrijednost), a unutar „toMail“ metode oblik obavijesti koja se šalje (oblik se generira putem Laravelove „MailMessage“ klase). Na „MailMessage“ klasi primjenjuju se metode „greeting“ (pozdrav na početku E-pošte) i „line“ (paragraf E-pošte) kojima se predaju informacije rezervacije dobivene putem konstruktora. Putem ovih metoda ispunjava se predložak koji se šalje putem E-pošte (predložak se nalazi u „vendor/laravel/framework/src/illuminate/Notifications/resources/views“ direktoriju te ga se tamo može prilagoditi potrebi). Slika 4.16. prikazuje „ReservationApproved“ klasu.

```php
<?php

namespace App\Notifications;

use Illuminate\Notifications\Notification;
use Illuminate\Notifications\Messages\MailMessage;

class ReservationApproved extends Notification
{
    use Queueable;

    protected $reservation;

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

    public function via($notifiable)
    {
        return ['mail'];
    }
    public function toMail($notifiable)
    {
        return (new MailMessage)
                    ->greeting('Hello '.$this->reservation->user->name.'!')
                    ->line('Reservation for '.$this->reservation->apartment->name.' from '.$this->reservation->check_in.' to '.$this->reservation->check_out.' has been approved. You can pay reservation bond on your reservation dashboard. If bond is not paid 7 days before reservation starts it will be removed from our schedule.')
                    ->line('We hope you have a pleasant stay.');
    }
}
```

Sl. 4.16. „ReservationApproved“ klasa.

Kako bi se omogućilo korištenje „notify“ metode za obavještavanje na modelu potrebno je tom modelu dodati „Notifiable“ svojstvo (eng. *Trait*, „use Notifiable“). U ovoj aplikaciji, „Notifiable“ svojstvo dodaje se „User“ modelu.

4.10. Kreiranje i razvoj poslova

Poslovi predstavljaju klase čija je svrha obavljanje kompleksne logike asinkrono (usko vezani uz red čekanja). Kako bi se kreirao posao pokreće se konzolna naredba „make:job“ [15].

Web aplikacija za rezerviranje apartmana sadrži klasu posla „SendMails“ koja se koristi za slanje korisničkih upita. Kako bi se kreirala „SendMails“ klasa pokreće se konzolna naredba

„php artisan make:job SendMails“. Pokretanjem ove naredbe kreira se „app/Jobs“ direktorij, ukoliko već nije kreiran, i „SendMails“ klasa unutar njega. Kada je klasa posla kreirana, u sebi sadrži „handle“ metodu u kojoj se piše logika posla. „SendMails“ klasa osim „handle“ metode sadrži i konstruktor kojim dobiva informacije o upitu koji se šalje. Unutar „handle“ metode, putem Laravelovog „Mail“ mehanizma šalje se E-pošta na adresu definiranu u „.env“ datoteci, a poruka se oblikuje putem kreirane „Mail“ klase („Mail“ klasa će biti objašnjena u idućem dijelu). Kao i slušatelj događaja, ovaj posao implementira „ShouldQueue“ sučelje kojim se omogućuje njegovo stavljanje u red čekanja i asinkrono izvođenje. Slika 4.17. prikazuje „SendMails“ klasu.

```php
<?php

namespace App\Jobs;

use App\Mail\Inquiry;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;
use Illuminate\Support\Facades\Mail;

class SendMails implements ShouldQueue
{
    use InteractsWithQueue, Queueable, SerializesModels;

    protected $inquiry;

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

    public function handle()
    {
        Mail::to(config('mail.username'))->send(new Inquiry($this->inquiry));
    }
}
```

Sl. 4.17. „SendMails“ klasa.

4.11. Kreiranje i razvoj pošte

Laravel pruža jednostavan API putem „SwiftMailer“ biblioteke za slanje pošte putem servisa za slanje E-pošte kao što su Mailgun, SparkPost, AmazonSES i drugih mehanizama ili putem SMTP (eng. *Simple Mail Transfer Protocol*, internet protokol za slanje E-pošte) [17]. Aplikacija za rezerviranje apartmana koristiti će SMTP za slanje E-pošte.

Kako bi se kreirala nova klasa pošte potrebno je pokrenuti konzolnu naredbu „make:mail“. Za potrebe aplikacije za rezerviranje apartmana kreirat će se „Inquiry“ klasa za slanje korisničkih upita putem „php artisan make:mail Inquiry“ konzolne naredbe. Pokretanjem ove naredbe kreira se „app/Mail“ direktorij, ukoliko već nije kreiran, i „Inquiry“ klasa unutar njega. Klasa „Inquiry“ u sebi sadrži „build“ metodu putem koje se kreira oblik E-pošte koja će se poslati i konstruktor putem koje se prosljeđuju informacije iz upita. Metoda „build“ kao rezultat vraća predložak HTML i čistog tekstualnog oblika poruke uz podatke korisničkog upita kojim će se ispuniti (HTML i čisti tekstualni predložak programer sam izrađuje). Predlošci za slanje upita ove aplikacije se nalaze u „resources/views/info“ direktoriju. Slika 4.18. prikazuje „Inquiry“ klasu.

```php
<?php

namespace App\Mail;

use Illuminate\Mail\Mailable;

class Inquiry extends Mailable
{

   protected $inquiry;

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

   public function build()
   {
      return $this->view('info.inquiry')
                  ->text('info.inquiry_plain')
                  ->with(['mail'=>$this->inquiry]);
   }
}
```

Sl. 4.18. „Inquiry“ klasa.

4.12. Kreiranje i razvoj Artisan konzolnih naredbi

Laravel omogućuje kreiranje klasa Artisan konzolnih naredbi putem „make:command“ konzolne naredbe [18]. Web aplikacija za rezerviranje apartmana sadrži Artisan konzolnu naredbu „reservations:clean“ koja koristi za brisanje neplaćenih rezervacija iz baze podataka. Također, ukoliko je administrator ili moderator aplikacije omogućio, briše iz baze podataka rezervacije starije od godinu dana.

Kako bi se kreirala klasa „ReservationsCleanUp“ pokreće se „php artisan make:command ReservationsCleanUp“ konzolna naredba. Ova naredba kreira „ReservationsCleanUp“ klasu unutar „app/Console/Commands“ direktorija. Kreirana klasa sadrži:

a) „$signature“ atribut - Naziv kojom se naredba pokreće.
b) „$description“ atribut - Opis naredbe.
c) „handle“ metodu - Sadrži logiku koja se izvodi prilikom pokretanja naredbe.

```php
<?php

namespace App\Console\Commands;

use App\Info;
use App\Reservation;
use Carbon\Carbon;
use Illuminate\Console\Command;

class ReservationsCleanUp extends Command
{

    protected $signature = 'reservations:clean';
    protected $description = 'Cleans old reservations from database';

    public function __construct()
    {
        parent::__construct();
    }

    public function handle()
    {
        $today=Carbon::today()->format('Y-m-d');

        $tA=explode('-',$today);
        $dateGap=Carbon::create((int)$tA[0],(int)$tA[1],(int)$tA[2]+7)->format('Y-m-d');

        Reservation::where('status','!=','2')->where('check_in','<=',$dateGap)->delete();

        $cleanup=(Info::select('cleanup')->first())->cleanup;

        if($cleanup==1){
            $dateGap=Carbon::create((int)$tA[0]-1,(int)$tA[1],(int)$tA[2])->format('Y-m-d');
            Reservation::where('check_out','<',$dateGap)->delete();
        }

        $this->info('Reservations have been cleaned.');
    }

}
```

Sl. 4.19. „ReservationsCleanUp“ klasa.

Nakon što je logika „ReservationsCleanUp“ Artisan naredbe razvijena (Sl. 4.19), potrebno je naredbu registrirati u „Kernel.php“ datoteci koja se nalazi u „app/Console“ direktoriju. Naredba se registrira tako da u „$commands“ niz „Kernel“ klase unesemo putanju naredbe (Sl. 4.20).

```
protected $commands = [
    'App\Console\Commands\ReservationsCleanUp'
];
```

Sl. 4.20. Registriranje „ReservationsCleanUp“ Artisan naredbe u „$commands“ niz „Kernel.php“ datoteke

Nakon ovoga naredba se iz konzole može pokrenuti putem „php artisan reservations:clean“ konzolne naredbe.

4.13. Zakazivanje zadataka

Kako bi se zakazalo izvođenje nekih zadataka na serveru koristi se Cron (Linux alat koji služi za zakazivanje izvođenja određenih naredbi ili skripti). Iz razloga što se za svaki zadatak mora pisati novi Cron upis Laravel je razvio mehanizam zakazivanja zadataka iz same aplikacije koji omogućuje jednostavno i intuitivno pisanje koda za zakazivanje zadataka [19].

Zakazivanje naredbi izvršava se u „Kernel.php“ datoteci, unutar „app\Console“ direktorija, putem „schedule“ metode koja prima „$schedule“ parametar Laravelove klase „Schedule“. Nad „$schedule“ parametrom u „schedule“ metodi mogu se izvršiti „call“ (pokretanje *closure* ili anonimne funkcije) i „command“ (pokretanje konzolne naredbe) metode. U svrhu razvoja web aplikacije za rezerviranje apartmana, zakazuje se „reservations:clean“ Artisan naredba (Sl. 4.21.). Nakon „call“ ili „command“ metode dalje se mogu vezati neke od Laravelovih metoda kao što su [19]:

a) „daily()“ - Pokretanje zadatka svaki dan u 00:00 sati.
b) „hourly()“ - Pokretanje zadatka svaki sat.
c) „weekly()“ - Pokretanje zadatka svaki tjedan.
d) „monthly()“ - Pokretanje zadatka svaki mjesec.
e) „yearly()“ - Pokretanje zadatka svake godine.
f) „cron('******')“ - Zakazivanje pokretanja Cron oblikom.
g) „dailyAt('<vrijeme_u_danu>')“ - Pokretanje zadatka svaki dan u određenom vremenu (aplikacija koristi ovu metodu).

```
protected function schedule(Schedule $schedule)
{
    $schedule->command('reservations:clean')->dailyAt('20:00');
}
```

Sl. 4.21. Zakazivanje „reservations:clean“ Artisan naredbe u „schedule“ metodi „Kernel.php“ klase.

Također, postoji još dosta metoda zakazivanja koje se mogu naći na Laravelovoj službenoj dokumentaciji.

Ovaj način zakazivanja olakšava posao programeru te u Cron treba unijeti samo „php putanja_do_artisan_datoteke/artisan schedule:run >dev/null 2>&1“ liniju („>dev/null 2>&1“ dio linije označava da se ne vraća nikakav izlaz niti greška te nije nužan ukoliko se želi prikupljati izlaz izvedenih zadataka).

4.14. Pogledi i Blade predlošci

Laravel je temeljen na MVC arhitekturalnom uzorku te iz tog razloga pogledi predstavljaju jednu od temeljnih komponenti aplikacije. Pogledi sadrže HTML koji se prikazuje korisniku aplikacije te služe kako bi se odvojila prezentacijska od aplikacijske logike. Svi pogledi kreirani u Laravel aplikaciji se nalaze u „resources/views“ direktoriju.

Iz razloga što su pogledi aplikacija dinamički i ovise o aplikacijskoj logici, potrebne parametre za oblikovanje mogu im se predati putem:

a) „view“ globalne metode
b) „share“ metode

Metoda „view“ najčešće se koristi u kontrolerima kako bi se korisniku nakon izvršene akcije vratio odgovor u obliku pogleda s parametrima koji će ga ispuniti. Jednostavan primjer bi bila „edit“ metoda „InfoController“ kontrolera koja nakon izvršavanja aplikacijske logike vraća korisniku pogled „info.info“ (Laravel „info.info“ prevodi u „resources/views/info/info.blade.php“ te na taj način zna koji pogled treba prikazati korisniku) s „info“ i „photos“ parametrima kojima će se pogled ispuniti (Sl. 4.22.).

```
public function edit()
{
    $info=Info::where('id',1)->select('general','location','apartments','map_mark')->first();
    $photos=Carousel::all();
    return view('info.info',compact('info','photos'));
}
```

Sl. 4.22. „edit“ metoda „InfoController“ kontrolera.

Metoda „share“ koristi se za dijeljenje nekog parametra sa svim pogledima aplikacije te se vezanje parametara tom metodom izvršava u „boot“ metodi „AppServiceProviders“ klase unutar „app/Providers“ direktorija [20]. Za potrebe razvoja web aplikacije za rezerviranje apartmana, potrebno je svakom pogledu proslijediti kontakt informacije koje se koriste za

oblikovanje podnožja (eng. *Footer*) aplikacije. Nakon što se obavi aplikacijska logika za prikupljanje potrebnih parametara, poziva se „share“ metoda Laravelove „View“ klase putem koje se ti parametri vežu uz poglede (Sl. 4.23.).

```
public function boot()
{
    $fInfo=Cache::tags('infos')->rememberForever('fInfo', function(){
        $r=Info::where('id',1)->select('country','city','address','email','phone')->first();
        $rValue=(object)([
            'country'=>'undefined',
            'city'=>'undefined',
            'address'=>'undefined',
            'email'=>'undefined',
            'phone'=>'undefined'
        ]);
        if($r){
            $rValue=(object)$r;
        }
        return (object)$rValue;
    });
    View::share(compact('fInfo'));
}
```

Sl. 4.23. „boot“ metoda „AppServiceProvider“ klase.

Blade predlošci predstavljaju datoteke s „.blade.php“ ekstenzijom. U suštini, Blade predlošci su obične datoteke s „.php“ ekstenzijom u kojima se zamjenjuje neuredna PHP sintaksa za dinamično popunjavanje HTML predloška s Blade sintaksom („.blade.php“ datoteke se kompajliraju u „.php“ datoteke). Osnovne funkcionalnosti koje pruža Blade su [21]:

a) Definiranje okvirnog predloška i njegovo proširivanje.
b) Prikaz podataka.
c) Logičke petlje.

a) Definiranje okvirnog predloška podrazumijeva izrada predloška koji će koristit skoro svaki pogled aplikacije te će ga proširiti drugim predloškom za vlastite potrebe. Na primjeru web aplikacije za izradu apartmana definira se okvirni predložak „layout.blade.php“ u „resources/views“ direktoriju koji se koristi za sve poglede aplikacije. Ovaj pogled u sebi sadrži „@yield“ linije koda s kojima se definira mjesta i nazivi odjeljaka koje će okvirni predložak ubaciti iz predloška kojeg proširuje. Nakon kreiranja okvirnog predloška potrebno je izraditi predloške kojima će se okvirni proširiti. Za prikaz pogleda početne stranice web aplikacije za rezerviranje apartmana kreira se „home.blade.php“ predložak u istom direktoriju kao i okvirni predložak. Kako bi Blade znao da ovaj predložak treba proširiti okvirni potrebno je na početku

predloška to definirati „@extends“ linijom koda. Nakon što je definiran okvirni predložak potrebno je izraditi odjeljke (eng. *Sections*). Svaki odjeljak započinje s „@section“ linijom koda koja sadrži naziv tog odjeljka i završava s „@stop“ linijom koda. Svaki odjeljak predloška zamjenjuje „@yield“ liniju koda okvirnog predloška ukoliko se njihovi nazivi podudaraju.

b) Prikaz parametara koji su preneseni za korištenje u Blade predlošku obavlja se tako da se parametri zatvore unutar dvostrukih vitičastih zagrada (pr. „{{$name}}“). Ukoliko se koriste neki Javascript okviri, poput Angular.js, koji također obrađuju parametre unutar dvostrukih vitičastih zagrada te je određeni parametar namijenjen njima, Blade može zanemariti tako da se ispred parametra doda „@“ simbol(pr. „@{{$name}}“). Također putem „@verbatim“ (početak odjeljka) i „@endverbatim“ (kraj odjeljka) linije koda mogu se zanemariti cijeli odjeljci HTML-a.

c) Logičke petlje koje pruža Blade su „@if“, „@for“, „@foreach“ i mnogo drugih. Označavanje kraja petlje obavlja se dodavanjem linije koda koja odgovara petlji i sadrži „end“ dio između „@“ simbola i naziva petlje (pr. „@if()“ označuje početak *if* petlje, a „@endif“ kraj). Parametri koji se koriste unutar zagrada petlji ne trebaju biti unutar dvostrukih vitičastih zagrada (vitičaste zagrade „{{}}“ predstavljaju „<?php ?>“ što je već implementirano putem Blade petlji).

Sve Blade komponente koje rade s PHP parametrima ({{}} i petlje) se kompajliraju u PHP kod. To znači da nije nužno koristit Blade komponente, ali čine kod urednijim i čitljivijim.

4.15. Kreiranje i razvoj prilagođenih pogleda greški.

Sve Laravel aplikacije, ukoliko dođe do greške, prikazuju u pretraživaču pogled s detaljnim opisom greške (detaljan opis može se isključiti postavljanje „APP_DEBUG“ parametra unutar „.env“ datoteke na logičku laž), ali omogućuju programeru jednostavnu zamjenu prilagođenim pogledom. Prilagođeni pogledi grešaka se kreiraju u „resources/views/errors“ direktoriju a trebaju imati naziv oblika „<kod_HTTP_greške>.blade.php“ (pr. „404.blade.php“) [22]. Ukoliko Laravelov upravitelj greškama za određeni kod greške pronađe pogled koji mu odgovara, on će umjesto zadanog prikazati prilagođeni pogled (upravitelj greškama automatski povezuje poglede grešaka s kodom uhvaćene greške ako postoji pogled za njih).

Za neke greške Laravel nije omogućio prilagođen pogled (pr. „500“ HTTP greška), ali se može primijeniti zaobilazno rješenje tako da u „render“ metodi „Handler“ klase unutar

„app/Exceptions“ direktorija unesemo blok logike za hvatanje te greške i preusmjeravanje na prilagođeni pogled (Sl. 4.24.).

```
public function render($request, Exception $e)
{
    if($e instanceof FatalErrorException) {
       return response()->view('errors.500');
    }
    return parent::render($request, $e);
}
```

Sl. 4.24. Prikaz prilagođenog pogleda za „500“ HTTP grešku u „render“ metodi „Handler.php“ klase.

4.16. Kreiranje i razvoj kontrolera

Kontroleri služe za interakciju korisnika s web aplikacijom ili stranicom. Svaki korisnički HTTP zahtjev predaje se određenoj kontrolerovoj metodi u ovisnosti na rutu kojoj je taj zahtjev predan.

U Laravelu, kako bi se kreirao kontroler koristi se konzolna naredba „php artisan make:controller <naziv_kontrolera>“. Konvencija za imenovanje kontrolera je naziv resursa koji se kontrolira (najčešće je to model) te se na njega dodaje riječ „Controller“ (koristi se „CamelCase“ metoda s velikim prvim početnim slovom).

Ukoliko se želi kreirati resursni kontroler na kraj konzolne naredbe dodaje se „—resource“. Tako kreiran kontroler zove se resursni kontroler te je namijenjen za CRUD manipuliranje određenim resursom. Kada je kreiran resursni kontroler on u sebi sadrži [23]:

a) „indeks“ metodu - Metoda za dohvaćanje svih jedinica resursa.
b) „create“ metodu - Metoda za prikaz forme stvaranje jedinice resursa.
c) „store“ metodu - Metoda za spremanje jedinice resursa u bazu podataka.
d) „show“ metodu - Metoda za prikaz jedinice resursa.
e) „edit“ metodu - Metoda za prikaz forme izmjene jedinice resursa.
f) „update“ metodu - Metoda za spremanje promjena jedinice resursa u bazu podataka.
g) „delete“ metodu - Metoda za brisanje jedinice resursa.

Za primjer, kreira se resursni kontroler „ReservationsController“.

a) Kada korisnik pošalje HTTP GET zahtjev na „http://domena/reservations“ pokreće se „indeks“ metoda koja, ako je korisnik administrator ili moderator, izvršava četiri upita na bazu.

To su upiti za nadolazeće, aktivne i arhivirane rezervacije putem „Reservation“ modela (na temelju trenutnog datuma) te informaciju, je li omogućeno automatsko brisanje zastarjelih rezervacija iz baze podataka putem „Info“ modela. Ako korisnik nije administrator ili moderator, dohvaćaju se samo rezervacije koje pripadaju njemu (Sl. 4.25.).

```
public function index()
{
    if (Auth::user()->hasRole()) {
        $today=Carbon::today()->format('Y-m-d');
        $old=Reservation::where('check_out','<',$today)->with('apartment')->get();
        $current=Reservation::where('check_in','<=',$today)
        ->where('check_out','>=',$today)->with('apartment')->get();
        $pending=Reservation::where('check_in','>',$today)->with('apartment')->get();
        $cleanup=(Info::select('cleanup')->first())->cleanup;
        return view('reservations.show', compact('old','current','pending','cleanup'));
    }
    else{
        $reservations=Reservation::where('user_id',Auth::id())->with('apartment')->get();
        return view('reservations.show', compact('reservations'));
    }
}
```

Sl. 4.25. „indeks“ metoda „ReservationsController“ kontrolera.

Kako se iz koda može vidjeti, koristi se metoda „nestrpljivog/željnog“ učitavanja (eng. *Eager Loading*). To znači da se pri učitavanju svake rezervacije automatski učitava i apartman vezan za nju. Jednostavan primjer bi bio: „Reservation::with('apartment')->get()“.

Ova metoda omogućuje izbjegavanje "N+1" problema pri upitima, gdje "N+1" problem označava izvođenje jednog upita za dohvaćanje svih rezervacija (1) ali kad se putem relacije (metoda „apartment“ u „Reservation“ modelu) zatraži apartman, za svaku rezervaciju će se izvesti još jedan upit (N). Korištenjem „nestrpljivog/željnog“ učitavanja, broj upita se smanjuje na dva (jedan za dohvaćanje svih rezervacija i jedan za dohvaćanje apartmana svih rezervacija).

Metoda „indeks“ završava se tako da sva četiri parametra preda definiranom pogledu za prikaz rezervacija putem „view“ metode. Metoda „view“ za prvi parametar prima putanju do definiranog pogleda („reservations.show“ Laravel prevodi u „resources/views/reservations/show.blade.php“), a za drugi „compact“ metodu pomoću koje se predaju potrebni parametri pogledu (unutar „compact“ metode imena varijabli su ista kao i imena u PHP varijabli ali bez znaka „$“).

b) Kada korisnik pošalje HTTP GET zahtjev na „http://domena/reservations/create/{id}“ pokreće se „create“ metoda kontrolera koja dohvaća apartman sa svim njegovim rezervacijama putem

proslijeđenog „id“ parametra. Kada se apartman dohvati prosljeđuje se dalje pogledu za kreiranje rezervacije (Sl. 4.26.).

```
public function create($id)
{
    $apartment=Apartment::where('id',$id)->with('reservations')->first();
    return view('reservations.create', compact('apartment'));
}
```

Sl. 4.26. „create“ metoda „ReservationsController“ kontrolera.

c) HTTP POST zahtjev na „http://domena/reservations/store/{id}“ pokreće „store“ metodu kontrolera koja dohvaća apartman putem proslijeđenog „id“ parametra, kreira novu instancu „Reservation“ modela, dodjeljuje instanci parametre iz zahtjeva, povezuje rezervaciju s dohvaćenim apartmanom i korisnikom koji izrađuje rezervaciju te je sprema u bazu i vraća JSON odgovor (Sl. 4.27.).

```
public function store($id, AddReservationRequest $request)
{
    $apartment=Apartment::locatedAt($id);
    $reservation=new Reservation;
    $reservation->info=$request->info;
    $dateRange= explode('/', $request->date_range);
    $reservation->check_in=$dateRange[0];
    $reservation->check_out=$dateRange[1];
    $reservation->status=0;
    $reservation->apartment()->associate($apartment);
    $reservation=Auth::user()->addReservation($reservation);
    echo json_encode(array('Success','Reservation has been submited.','success'));
}
```

Sl. 4.27. „store“ metoda „ReservationsController“ kontrolera.

d) HTTP GET zahtjev na „http://domena/reservations/{id}“ pokreće „show“ metodu kontrolera koja dohvaća rezervaciju putem proslijeđenog „id“ parametra te apartman, i sve njegove druge rezervacije, koji je vezan uz dohvaćenu rezervaciju. U ovoj metodi prikazan je primjer „nestrpljivog/željnog“ učitavanja s ugniježđenim vezama. Unutar „with“ metode, drugi parametar „apartment.reservations“ označava da se želi dohvatiti rezervacija s pripadajućim apartmanom, ali se i želi dohvatiti ostale rezervacije tog apartmana (Sl. 4.28.).

```
public function show($id)
{
    $reservation=Reservation::where('id',$id)->with('user', 'apartment.reservations')->first();
    return view('reservations.info',compact('reservation'));
}
```

Sl. 4.28. „show“ metoda „ReservationsController“ kontrolera.

e) HTTP GET zahtjev na „http://domena/reservations/{id}/edit“ pokreće „edit“ metodu kontrolera koja dohvaća rezervaciju putem proslijeđenog „id“ parametra te ju prosljeđuje pogledu koji priprema formu za izmjenu rezervacije (Sl. 4.29.).

```
public function edit($id)
{
    $reservation=Reservation::where('id',$id)->with('apartment.reservations')->first();
    if($reservation->status!=2){
       return view('reservations.edit',compact('reservation'));
    }
    else{
       return back();
    }
}
```

Sl. 4.29. „edit“ metoda „ReservationsController“ kontrolera.

f) HTTP PATCH zahtjev na „http://domena/reservations/{id}“ pokreće „update“ metodu kontrolera koja dohvaća rezervaciju putem proslijeđenog „id“ parametra te joj dodjeljuje parametre iz zahtjeva. Nakon izmjene, osvježava se zapis te rezervacije u bazi i vraća JSON odgovor (Sl. 4.30.).

```
public function update($id, AddReservationRequest $request)
{
    $reservation=Reservation::where('id',$id)->first();
    $reservation->status=0;
    $reservation->info=$request->info;
    $dateRange= explode('/', $request->date_range);
    $reservation->check_in=$dateRange[0];
    $reservation->check_out=$dateRange[1];
    $reservation->update();
    echo json_encode(array('Success','Reservation has been updated.','success'));
 }
```

Sl. 4.30. „update“ metoda „ReservationsController“ kontrolera.

g) HTTP DELETE zahtjev na „http://domena/reservations/{id}“ pokreće „destroy“ metodu kontrolera koja dohvaća rezervaciju putem proslijeđenog „id“ parametra te je briše iz baze i vraća JSON odgovor (Sl. 4.31.).

```
public function destroy($id)
{
    Reservation::findOrFail($id)->delete();
    echo json_encode(array('Success','Reservation has been deleted.','success'));
}
```

Sl. 4.31. „destroy“ metoda „ReservationsController“ kontrolera.

U svakom kontroleru može se putem konstruktora („__construct“ metoda, metoda kreiranja instance objekta putem koje se u objekt injektiraju potrebne ovisnosti) definirati koji *middleware* želimo primijeniti na kontroler. Za svrhu ovog primjera, dodaje se dva *middleware*-a, a to su „auth“ *middleware*, koji se primjenjuje na cijelom kontroleru, i „has_role“ *middleware* koji se primjenjuje samo na „confirmReservation“, „show“, „declineReservation“ i „reservationsCleanup“ metodama kontrolera (Sl. 4.32).

```
public function __construct()
{
    $this->middleware('auth');
    $this->middleware('has_role', ['only' =>
['reservationsCleanup','confirmReservation','show','declineReservation']]);
}
```

Sl. 4.32. „__construct“ metoda „ReservationsController“ kontrolera.

Osim navedenih metoda, „ReservationController“ sadrži još četiri metode a to su:

a) „confirmReservation“
b) „payReservation“
c) „declineReservation“
d) „reservationsCleanup“

a) HTTP PATCH zahtjev na „http://domena/confirm/{id}“ pokreće „confirmReservation“ metodu koja dohvaća rezervaciju putem proslijeđenog „id“ parametra te joj mijenja „status“ parametar na „1“ što označava potvrđenu rezervaciju, pokreće događaj „ReservationStatusHasChanged“ (*event* klasa) s rezervacijom i logičkom istinom (*true* je opcionalni parametar koji preopterećuje zadanu vrijednost *false*, označuje događaj prihvaćanja, a ne odbijanja rezervacije.) kao parametrima te sprema promjene rezervacije u bazu i vraća JSON odgovor za prikaz korisniku (Sl. 4.33.).

```
public function confirmReservation($id) {
    $reservation=Reservation::where('id',$id)->with('user','apartment')->first();
    $reservation->status=1;
    event(new ReservationStatusHasChanged($reservation, true));
    $reservation->update();
    echo json_encode(array('Success','Reservation has been confirmed.','success',$reservation-
>check_in,$reservation->check_out));
}
```

Sl. 4.33. „confirmReservation“ metoda „ReservationsController“ kontrolera.

b) HTTP POST zahtjev na „http://domena/checkout/{id}“ pokreće „payReservation“ metodu koja putem parametara zahtjeva (Stripe žeton - apstraktni prikaz platne kartice) i „id“ parametra

kreira uplatu, na Stripe posrednik za plaćanje (eng. *Gateway*, kartični podatci nikada ne dolaze na server, pa se vlasnika web aplikacije ili stranice ne može držati pravno odgovornim), za dohvaćenu rezervaciju putem „id“ parametra. Ukoliko transakcija uspije, mijenja se status rezervacije na „2“ (označava plaćenu rezervaciju), sprema se izmjena rezervacije u bazu i vraća se JSON odgovor o uspjehu. U suprotnom, hvata se greška *(try/catch* mehanizam) tipa „StripeErrorInvalidRequest“ i vraća se JSON odgovor o neuspjehu (Sl. 4.34.).

```
public function payReservation(Request $request, $id) {
      $token=$request->stripeToken;
      $bond=(int) $request->bond;
      $reservation=Reservation::where('id',$id)->with('user')->first();
      $reservation->status=2;
      try{
         StripeCharge::create([
            'amount' => $bond*100,
            'currency' => 'usd',
            'card' => $token,
            'description' => 'Bond payment fo reservation under ID: '.$id.'. Payed by: '.$reservation-
>user->name.' (email: '.$reservation->user->email.').'
         ],
         [
            'api_key' => config('keys.STRIPE_SECRET')
         ]);
         $reservation=Reservation::where('id',$id)->first();
         $reservation->status=2;
         $reservation->update();
         echo json_encode(array('Success','Payment has been received.','success',$id));
      } catch (StripeErrorInvalidRequest $e) {
         echo json_encode(array('Error','Insufficient funds.','error'));
      }
}
```

Sl. 4.34. „payReservation“ metoda „ReservationsController“ kontrolera.

c) HTTP DELETE zahtjev na „http://domena/decline/{id}“ pokreće „declineReservation“ metodu koja dohvaća rezervaciju putem proslijeđenog „id“ parametra pokreće događaj „ReservationStatusHasChanged“ (*event* klasa) s rezervacijom kao parametrom te briše tu rezervaciju iz baze (Sl. 4.35.).

```
      public function declineReservation($id){
            $reservation=Reservation::where('id',$id)->with('user','apartment')->first();
            event(new ReservationStatusHasChanged($reservation));
            $reservation->delete();
            return redirect('reservations')->with('status', 'reservation declined');
      }
```

Sl. 4.35. „declineReservation“ metoda „ReservationsController“ kontrolera.

d) HTTP POST zahtjev na „http://domena/cleanup“ pokreće „reservationsCleanup“ metodu koja u bazu sprema informaciju je li automatsko čišćenje baze uključeno ili isključeno ovisno o parametru zahtjeva (Sl. 4.36.).

```
public function reservationsCleanup(Request $request) {
    $info=Info::where('id',1)->first();
    if(! is_null($request->cleanup)){
       $info->cleanup=1;
       $status='Database auto cleanup has been enabled';
    }
    else{
       $info->cleanup=0;
       $status='Database auto cleanup has been disabled';
    }
    $info->update();
    echo json_encode(array('Success',$status,'success'));
}
```

Sl. 4.36. „reservationsCleanup“ metoda „ReservationsController“ kontrolera.

Kreiranje i razvoj strukture kontrolera je na ovaj način ubrzan i razumljiv te omogućuje programeru da se više posveti razvoju njegove logike.

4.17. Predmemoriranje

Laravel pruža podršku za predmemoriranje putem spremanja u datoteku ili bazu podataka (pr. mysql relacijska baza podataka te Memcached ili Redis ključ-vrijednost ne relacijske baze podataka). Kako bi se koristio Redis za predmemoriranje, potrebno je putem Composer instaliralti „predis“ biblioteku (podpoglavlje 4.8.) i u „.env“ datoteci postaviti „CACHE_DRIVER“ na „redis“.

Predmemoriranje obavlja se putem Laravelove „Cache“ klase koja sadrži [24]:

a) „get(<ključ>)“ metodu - Dohvaćanje vrijednosti za definirani ključ. Također kao drugi parametar možemo postaviti zadanu vrijednost ukoliko ključ ne postoji ili anonimnu funkciju čija će povratna vrijednost predstavljati rezultat.
b) „store(<driver>)“ metodu - Ova metoda omogućuje izmjenu mehanizma za memoriranje (pr. ukoliko je zadan Redis, može se za određeno spremanje postaviti „store('memcached')").
c) „has(<ključ>)“ metodu - Provjera postoji li određeni ključ spremljen.

d) „increment(<ključ>)“ metodu - Metoda kojom se inkrementira cjelobrojna vrijednost određenog ključa. Kao drugi parametar može se postaviti količina kojom se vrijednost ključa inkrementira.
e) „decrement(<ključ>)“ metodu - Metoda kojom se dekrementira cjelobrojna vrijednost određenog ključa. Kao drugi parametar može se postaviti količina kojom se vrijednost ključa dekrementira.
f) „remeber(<ključ>,<trajanje>,<anonimna_funkcija>)“ metodu - Metoda koja dohvaća vrijednost ključa ako postoji, a u suprotnom sprema povratnu vrijednost anonimne funkcije pod tim ključem u definiranom vremenu trajanja (trajanje se definira u minutama).
g) „pull(<ključ>)“ metodu - Dohvaća vrijednost ključa te briše ključ.
h) „put(<ključ>,<vrijednost>,<trajanje>)“ metodu - Spremanje vrijednosti pod definiranim ključem u definiranom vremenu trajanja.
i) „add(<ključ>,<vrijednost>,<trajanje>)“ metodu - Spremanje vrijednosti pod definiranim ključem u definiranom vremenu trajanja samo ako taj ključ ne postoji.
j) „forever(<ključ>,<vrijednost>)“ metodu - Spremanje ključa i njegove vrijednosti zauvijek.
k) „rememberForever(<ključ>,<anonimna_funkcija>)“ metodu - Kombinacija „remember“ i „forever“ metoda. Dohvaća vrijednost ključa ako postoji, a u suprotnom sprema povratnu vrijednost anonimne funkcije pod tim ključem zauvijek.
l) „forget(<ključ>)“ metodu - Brisanje ključa.
m) „flush()“ metodu - Brisanje svih ključeva.
n) „tags()“ metodu - Dodavanje oznaka ključevima (jednu oznaku ili niz oznaka). Koristi se ukoliko postoji ovisnost između skupa ključeva (pr. „Cache::tags('infos')->flush()“ - briše sve ključeve s „infos“ oznakom). Ukoliko se dohvaća ključ s oznakom, metoda dohvaćanja se treba vezati na „tags()“ metodu (pr. „Cache::tags('infos')->get('info')“ - dohvaća "info" ključ s oznakom „infos“).

U web aplikaciji za rezerviranje apartmana, predmemoriranje se koristi za spremanje općenitih informacija, apartmana i slika koje se koriste na početnoj stranici i stranici s popisom apartmana.

Jednostavan primjer predmemoriranja je „index“ metoda „ApartmentsController“ kontrolera. Unutar „indeks“ metode, za formatiranje pogleda koji se vraća korisniku potrebno je dohvatiti listu svih apartmana iz baze. Kako će ta stranica u stvarnost bit najviše posjećivana i relativno

nepromjenjiva, rezultati upita biti će spremljeni u Redis pod „apartments“ ključem putem „rememberForever“ metode (Sl. 4.37.). Kada je određeni apartman izmijenjen, obrisan ili je dodan novi (metode „update“, „destroy“ i „store“) pokreće se „flush“ metoda te se briše „apartments“ ključ.

```
public function index()
{
    $apartments=Cache::rememberForever('apartments', function(){
       return Apartment::with('photos')->get();
    });

    return view('apartments.show', compact('apartments'));
}
```

Sl. 4.37. Predmemoriranje postojećih apartmana u „index“ metodi „ApartmentsController“ kontrolera.

Predmemoriranje se koristi i u „InfoController“ i „PhotoController“ kontrolerima te u „AppServiceProvider“ klasi.

4.18. Testiranje

Laravel u sebi sadrži podršku za PHPUnit (testni okvir za PHP) testiranje odmah nakon instaliranja aplikacije. Kako bi se kreirao testni slučaj pokreće se „make:test“ Artisan naredba [25]. U svrhu demonstracije, kreiran je testni slučaj „AddApartmentTest“ putem konzolne naredbe „php artisan make:test AddApartmentTest“. Nakon izvođenja naredbe kreira se „AddApartmentTest“ klasa unutar „tests“ direktorija s „testExample“ metodom kao primjerom. Ovaj test služit će za provjeru što će se dogoditi ukoliko korisnik s ovlastima i bez njih pokuša kreirati novi apartman.

Metoda „testHasRoleAdding“ dohvaća prvog korisnika s ovlastima te se postavlja da se buduće akcije odvijaju u njegovo ime. Posjećuje će se ruta „http://domena/apartments“, provjerava se postajanje elementa „Add Apartment“ na stranici te ako postoji klikne na njega. Ukoliko je test prošao dalje provjerava se je li trenutna ruta „http://domena/apartments/create“ te ako je ispuniti polja za unos, predati formu, posjetiti „http://domena/apartments“ rutu i provjeriti postoji li „New Apartment“ element (novo dodani apartman).

Metoda „testNoRoleAdding“ dohvaća prvog korisnika bez ovlasti te se postavlja da se buduće akcije odvijaju u njegovo ime. Posjećuje će se ruta „http://domena/apartments“, provjerava se postajanje elementa „Add Apartment“ na stranici te ako ne postoji posjećuje se

ruta „http://domena/apartments/create“ i provjerava nalazi li se korisnik na toj ruti ili je preusmjeren na „http://domena“ rutu. Slika 4.38. prikazuje izgled gotove testne klase „AddApartmentTest“.

```php
<?php

use Illuminate\Foundation\Testing\WithoutMiddleware;
use Illuminate\Foundation\Testing\DatabaseMigrations;
use Illuminate\Foundation\Testing\DatabaseTransactions;

class AddApartmentTest extends TestCase
{
    public function testHasRoleAdding()
    {
        $user=App\User::whereIn('role',['1','2'])->first();
      $this->actingAs($user)
        ->visit('apartments')
        ->see('Add Apartment')
        ->click('Add Apartment')
        ->seePageIs('/apartments/create')
        ->type('New Apartment', 'name')
        ->type('New Apartment Description', 'description')
        ->type('100', 'size')
        ->type('3', 'rooms')
        ->type('100', 'price')
        ->press('Create Apartment')
        ->visit('apartments')
        ->see('New Apartment');
    }

    public function testNoRoleAdding()
    {
        $user=App\User::where('role','0')->first();
      $this->actingAs($user)
        ->visit('apartments')
        ->dontSee('Add Apartment')
        ->visit('apartments/create')
        ->seePageIs('/');
    }
}
```

Sl. 4.38. „AddApartmentTest“ testna klasa.

Metode za kreiranje PHPUnit testova mogu se pronaći u službenoj dokumentaciji Laravel API pod „Illuminate/Foundation/Testing“ imenskim prostorom unutar „TestCase“ klase..

Kako bi se napisani test pokrenuo, potrebno je iz terminala ući pokrenuti konzolnu naredbu „phpunit tests/<naziv_testa>“. Pokretanje konzolne naredbe „phpunit tests/AddApartmentTest“ rezultira uspješnim izvršavanjem testa što je prikazano na slici 4.39.

```
domagoj@domagoj-K53SM:/var/www/html/apartmentsBooking$ phpunit tests/AddApartmen
tTest
PHPUnit 5.6.3 by Sebastian Bergmann and contributors.

.["Success","Apartment has been created.","success"].
                                                  2 / 2 (100%)

Time: 1.51 seconds, Memory: 14.00MB

OK (2 tests, 14 assertions)
domagoj@domagoj-K53SM:/var/www/html/apartmentsBooking$
```

Sl. 4.39. Rezultat pokretanja „AddApartmentTest“ testa.

4.19. Optimizacija Laravel aplikacija

Nakon što je aplikacija razvijena i postavljena na web poslužitelj, moguće je provesti nekoliko koraka kako bi se poboljšale performanse aplikacije. Optimizacija aplikacije postiže se pokretanjem Artisan konzolnih naredba a one su [26]:

a) „route:cache“ - Kreira se kompajlirana „bootstrap/cache/routes.php“ datoteka koja sadrži sve rute (registrira ih) iz „routes/api.php“, „routes/console.php“ i „routes/web.php“ datoteka čime se omogućuje brže učitavanje.

b) „config:cache“ - Kreira se kompajlirana „bootstrap/cache/config.php“ datoteka koja predstavlja sve konfiguracijske datoteke čime se ubrzava učitavanje. Važno je za naglasiti, iz razloga što se nakon izvršavanja ove naredbe ignorira „.env“ datoteka, da je prije potrebno svaki „env()“, „getenv()“ i „$_ENV[]“ (dohvaćanje konstanti okoline) poziv unutar aplikacije zamijeniti s „config()“ pozivom. Kako bi se putem „config“ metode dohvatila konstanta okoline potrebno ju je registrirati u konfiguracijsku datoteku koja će biti kompajlirana nakon pokretanja „config:cache“ naredbe (pr. web aplikacija za rezerviranje apartmana dohvaća Stripe i Google API ključeve, pa se kreira „config/keys.php“ datoteka; Sl. 4.40.).

```php
<?php

    return [
        'GMAP_KEY' => env('GMAP_KEY'),
        'STRIPE_KEY' => env('STRIPE_KEY'),
        'STRIPE_SECRET' => env('STRIPE_SECRET')
    ];
```

Sl. 4.40. „keys.php“ konfiguracijska klasa.

Nakon pokretanja naredbe i kreiranja „bootstrap/cache/config.php“ varijablama se može pristupiti putem „config(<naziv_konfiguracijske_datoteke>.<naziv_ključa>)“.

c) „optimize“ - Kreira se kompajlirana datoteka „bootstrap/cache/compiled.php“ koja u sebi sadrži sve klase koje se najčešće koriste u aplikaciji. Time se smanjuje broj datoteka koje Laravel treba uključiti/povezati što rezultira bržim učitavanjem. Ukoliko je naredba pokrenuta u fazi razvoja („APP_ENV=local“ u „.env“ datoteci ili kopajliranoj „config.php“ datoteci), „bootstrap/cache/compiled.php" neće se kreirati (može se prisliliti s dodavanjem „—force“ zastavice u naredbu, ili redovno generirati ako je „APP_ENV=production“). Naredba se u pravilu pokreće tek nakon „config:cache“ naredbe.

d) „view:clear“ - Svaki pogled web aplikacije se kompajlira i sprema unutar „storage/framework/views“ direktorija kada se učita prvi put u svrhu bržeg učitavanja aplikacije. Isti pogledi se spremaju ako se izmjene, ali stari pogledi ostaju u direktoriju. Ova naredba se pokreće kako bi se obrisale starije inačice pogleda.

5. RAZVOJ WEB APLIKACIJE KORIŠTENJEM LARAVELA U ODNOSU NA RAZVOJ BEZ ILI S DRUGIM PROGRAMSKIM OKVIROM

Ovo poglavlje opisuje prednosti i nedostatke pri korištenju Laravel programskog okvira u odnosu na prednosti i nedostatke ne korištenja programskih okvira ili korištenja nekih drugih kao što su Yii i Symfony prilikom razvoja web rješenja.

5.1. Razvoj web aplikacija u Laravelu u odnosu na razvoj bez programskog okvira

Laravel, kao i drugi programski okviri, je skup alata i funkcionalnosti koji programerima olakšavaju i ubrzavaju proces razvoja aplikacije. Postoje mnogi argumenti za korištenje Laravela, ali i za njegovo izbjegavanje (također i drugih programskih okvira). Najveći argumenti za korištenje Laravela su njegove „Facade“ klase (uzorak dizajna, klasa sa skupom statičnih metoda koja pruža programeru jednostavno sučelje prema njima - pr. „Request“ klasa), migracije koje omogućuje jednostavno definiranje sheme baze podataka i njenu izmjenu, prisiljavanje modularnog pristupa razvoja aplikacija (MVC arhitektura), mehanizmi za rješavanje sigurnosnih problema, upravljanje varijablama sesije, Eloquent objektno-relacijsko mapiranje, jednostavna implementacija asinkronog procesiranje (redovi čekanja), intuitivnost koda Laravela, stabilnost, stalno održavanje, razvoj i integracija novih alata u okvir, opširna i jasna dokumentacija.

Svi navedeni argumenti, ali i mnogi drugih, podupiru osnovne ideje razvoja aplikacija putem programskog okvira kao što su:

a) „radi više s manje“ (eng. *do more with less*) - Najbolji primjer ovoga bi bile implementacija autentikacije (koja se putem Laravela može provesti za otprilike jednu minutu), Artisan CLI (konzolno sučelje, eng. Command Line Interface) generator, usmjeravanje zahtjeva, razvoj SQL upita, implementacija *observer* uzorka dizajna.

b) Održivost, modularnost i skalabilnost - Praćenje Laravelove najbolje prakse rezultira čistim, modularnim i skalabilnim kodom koji se lako održava i izmjenjuje kako bi se prilagodio novim specifikacijama i/ili zahtjevima.
c) Olakšavanje timskog razvoja - Korištenje Laravela zahtjeva praćenje i korištenje određenih postupaka i načina razvoja što olakšava suradnju među programerima u slučaju potrebe za nadogradnjom tuđih aplikacija i/ili modula te njihovog recenziranja ili implementiranja u kompleksniji sustav.

Jedini konkretni nedostatak razvoja web aplikacija u Laravelu (također i drugim programskim okvirima) u odnosu na razvoj bez programskog okvira je brzina aplikacije i njena veličina. Normalno je da PHP kod, s isključivo osnovnim kodom, ostvaruje bolje performanse u brzinama. Ta dodatna brzina „čistog" PHP koda dolazi od toga što ne postoji veliki broj komponenti (paketa) kao u programskom okviru koji se moraju povezati i pokrenuti (eng. *Bootstraping*).

Unatoč ovom nedostatku, u prosječnim aplikacijama radi se o razlici od oko 140 milisekundi koji nemaju preveliki utjecaj na performanse aplikacije.

Jedini slučaj, kada se treba razvijati aplikacija bez programskog okvira je kada se razvijaju manji servisi te aplikacije i/ili veliki sustavi čije potrebe ne može ispuniti niti jedan okvir (što nije česti slučaj). Potrebno je naglasiti, ako se započne razvoj aplikacije bez programskog okvira treba se paziti da ta aplikacija zapravo sadrži takve performanse da se isplati ne koristit programski okvir, jer razvoj aplikacije bez okvira ne mora nužno rezultirati boljim performansama, te paziti na rješavanje sigurnosnih pitanja aplikacije.

5.2. Razvoj web aplikacija u Laravelu u odnosu na druge programske okvire

Zbog velike popularnosti PHP-a u razvoju web aplikacija i stranica razvili su se mnogi programski okviri poput Laravel, Symfony i Yii, koji su trenutno najpopularniji PHP programski okviri. Prije nego što se krene s usporedbom potrebno je objasniti svojstva tih programskih okvira.

Symfony predstavlja set višekratno upotrebljivih PHP komponenti, koje omogućuju razvoj skalabilnih aplikacija visokih performansi. Podržava MVC razvoj aplikacija. Sadrži API-e koji pružaju jednostavnu integraciju s drugim aplikacijama te se može koristiti s popularnim *frontend* programskim okvirima (programski okviri za razvoj klijentske strane) poput Angular.js.

Pruža najbolju modularnost jer se temelji na razvoju putem komponenti. Mnoge njegove komponente koriste drugi programski okviri poput Laravela, ali i projekti kao što su Drupal (platforma za izradu CMS aplikacija) i phpBB (platforma za izradu foruma).

Yii predstavlja programski okvir visokih performansi za razvoj web aplikacija i/ili stranica. Podržava MVC razvoj aplikacija. Važan dio je i jQuery (Javascript okvir) integracija koja omogućuje razvojnim programerima klijentske strane lakšu prilagodbu te koristi. Kao i Symfony koristi komponente za omogućavanje brzog razvoja aplikacija, ali koristi i skele (eng. *Scaffolding*, pr. izuzetno brz način generiranja CRUD metoda za manipulaciju resursima) za generiranje koda. Laravel nije toliko modularan poput Symfony okvira, ali je sličan Yii-u. Ukoliko je projekt zahtjeva visoko modularni programski okvir, najbolje je korištenje Symfony okvira, a u suprotnom Laravel ili Yii.

Usporedba ova tri programska okvira može se najbolje provesti putem [27]:

a) Mehanizama za generiranje i oblikovanje predložaka.
b) Postupka instalacije
c) Brzina razvoja aplikacija
d) Performanse
e) Podrška baza podataka
f) Proširivost
g) Zajednica i resursi.

a) Generiranje i oblikovanje predložaka pruža korisne funkcionalnosti pri razvoju klijentske strane kao što su preskakanje HTML oznaka i filtriranje. Symfony zadano koristi Twig, a Laravel zadano koristi Blade. Osnovna razlika između Twig i Blade je u tome što Twig resursno skuplji od Blade za kompajliranje u čisti PHP, ali omogućuje *sandbox* (koncept rada u testnom okruženju) način proširivanja predložaka (ukoliko korisnik ima mogućnost promjene predloška Twig ga može procijeniti) i prilagođene oznake i filtere. Yii nema zadani mehanizam za generiranje i oblikovanje predložaka ali se većinom koristi Twig.

b) Sva tri programska okvira podržavaju instaliranje novih projekata putem Composer upravitelja paketa, ali i drugih načina te pružaju pokazni (eng. *Demo*) primjer aplikacije nakon instalacije.

c) Brzina razvoja je bitna jer svaka organizacija ili individualni programer ima za cilj lansiranje kvalitetne aplikacije što brže. Sva tri okvira podržavaju brzi razvoj korištenjem velikog broja

dostupnih komponenti. Symfony i Laravel također imaju veliki broj online vodiča koji pomažu pri savladavanju poteškoća u razvoju aplikacija.

d) U domeni performansi Yii daleko premašuje i Laravel i Symfony zbog svoje Lazy Loading metode. Dok Laravel i Symfony uključuju sve klase i funkcionalnosti koje se možda u tom dijelu neće iskoristiti, Yii ne uključuje klasu dok se ona ne zatraži, to jest proba kreirati njena nova instanca. Laravel ima najlošije performanse, ali se one mogu poboljšati različitim paketima (pr. „laravel-responsecache“ paket, paket za spremanje uspješnih GET zahtjeva u predmemoriju na tjedan dana čime se znatno ubrzavaju performanse).

e) Symfony podržava najveći broj baza podataka (13 baza podataka), dok Yii i Laravel podržava jednak broj uz razliku podrške za Redis (Laravel) i Oracle (Yii). Tablica 5.1. prikazuje listu podržanih baza za pojedini programski okvir.

Programski Okvir	**Laravel**	**Yii**	**Symfony**
Baze Podataka	Microsoft BI MongoDB MySQL PostgreSQL Redis SQLite	Microsoft BI MongoDB MySQL Oracle PostgreSQL SQLite	Apache Jackrabbit CouchDB DynamoDB GemFire GraphDB MemBase MemCacheDB Microsoft BI MongoDB MySQL NoSQL Oracle PostgreSQL

Tab. 5.1. Podržane baze podataka

f) Programski okviri su strukture koje imaju mogućnost proširivanja postojećih funkcionalnosti i dodavanje novih putem paketa. Packalyst (direktorij Laravelovih paketa) sadrži preko 9000 paketa dok za Symfony i Yii postoji oko 2800 paketa i skupova paketa (eng. *Bundles*).

g) Korištenje Laravela za razvoj web aplikacija je postao sve popularnija opcija te mu zbog toga raste zajednica koja utječe na broj paketa koji se za njega razvijaju kao i razvoj samog okvira. Symfony ima najveću zajednicu i podršku zbog njegove zrelosti kao programski okvir, a

najmanju ima Yii koja i dalje raste zbog njegove mogućnosti doprinosa razvoju aplikacija visokih performansi.

Unatoč razlika ovih programskih okvira, sva tri su dobre alternative jer pružaju okolinu za *full-stack* razvoj web aplikacija (razvoj serverske i klijentske strane), otvorenog su koda što programerima omogućuje bolje razumijevanje načina njihovog rada, podržavaju ORM i MVC arhitekturu, robusni su i sigurni te za pojedine inačice pružaju dugoročnu podršku (LTS inačice). Symfony je najstariji programski okvir i stoga najstabilniji s velikom zajednicom koja ga podržava, Yii omogućava razvoj aplikacija visokih performansi, a Laravel je mladi i stabilan programski okvir koji omogućuje razvoj aplikacija velikog raspona svrha te nema visoku krivulju učenja. Najbolji programski okvir ne postoji, ali je moguće izabrati najbolju alternativu ovisno o zahtjevima pojedinog projekt ili općeg rad neke IT kompanije.

6. ZAKLJUČAK

Razvoj web rješenja postaje sve kompleksniji pa u skladu s time dolazi do potrebe razvoja alata koji će pomoći u ispunjenju tih zahtjeva, gdje programski okviri predstavljaju vrlo važan alat. Značaj programskih okvira je što potiču programere na poštivanje određenih konvencija razvoja te daju pristup različitim paketima alata kojima se postižu bolje performanse timskog rada kao i performanse samog web rješenja.

Ovaj rad daje uvid u koncepte programskih okvira te prednosti i nedostatke Laravel programskog okvira i PHP programskog jezika koji čitatelju mogu poslužiti kao pomoć u nedoumici njihovog korištenja ili kao referentna točka istraživanja. Nadalje, daju se osnovni koncepti Laravela te postupci instalacije, konfiguriranja, razvoja i testiranja Laravel web rješenja putem razvoja web aplikacije za rezerviranje apartmana. Ta web aplikacija primjenjuje veliki postotak temeljnih alata i tehnika te nekolicinu naprednih koje čitatelju može pomoći u laganom i kvalitetnom započinjanju razvoja aplikacije, kao i pripomoć u savladavanju kompleksnijih problema ili daljnjih istraživanja okvira u cilju razvoja novih i/ili boljih web rješenja. Osim koncepta razvoja web rješenja i drugih značajki Laravel programskog okvira, čitatelj može dobiti uvid u značajnost korištenja programskog okvira, kao i razlike između drugih popularnih PHP programskih okvira koje mogu pripomoć kao referentna točka u svrhu daljnjeg istraživanja ili kao pomagalo u izboru odgovarajućeg programskog okvira za razvoj web rješenja specifičnih zahtjeva.

Glavni cilj ovog rada bio je dati uvid čitatelju u mogućnosti Laravel programskog okvira kao i postaviti dovoljno činjenica kojima bi se osporile tvrdnje neefikasnosti PHP-a kao programskog jezika, što je postignuto uspješnim razvojem web aplikacije za rezerviranje apartmana te njenom mogućnošću lagane nadogradnje, izmjene i testiranja kao i vrlo dobrim performansama.

LITERATURA

[1] What Is Software Framework -Its Uses And Types, http://www.moweble.com/what-is-software-framework-its-uses-and-types.html, pristupljeno 02.06.2017.

[2] Laravel, https://github.com/laravel/laravel, pristupljeno 02.06.2017.

[3] PHP Advantages and Disadvantages, http://www.techstrikers.com/PHP/php-advantages-disadvantages.php, pristupljeno 02.06.2017.

[4] Installation, https://laravel.com/docs/5.3/installation, pristupljeno 02.06.2017.

[5] Directory Structure, https://laravel.com/docs/5.3/structure, pristupljeno 02.06.2017.

[6] Model View Controller(MVC) in PHP, http://php-html.net/tutorials/model-view-controller-in-php/, pristupljeno 02.06.2017.

[7] Eloquent: Getting Started, https://laravel.com/docs/5.3/eloquent, pristupljeno 03.06.2017.

[8] 5 best security tips for a Laravel application, https://www.dunebook.com/5-best-security-tips-for-a-laravel-application, pristupljeno 03.06.2017.

[9] Authentication, https://laravel.com/docs/5.3/authentication, pristupljeno 03.06.2017.

[10] Database: Migrations, https://laravel.com/docs/5.3/migrations, pristupljeno 03.06.2017.

[11] Routing, https://laravel.com/docs/5.3/routing, pristupljeno 04.06.2017.

[12] Middleware, https://laravel.com/docs/5.3/middleware, pristupljeno 04.06.2017.

[13] Validation, https://laravel.com/docs/5.3/validation, pristupljeno 04.06.2017.

[14] Events, https://laravel.com/docs/5.3/events, pristupljeno 05.06.2017.

[15] Queues, https://laravel.com/docs/5.3/queues, pristupljeno 06.06.2017.

[16] Notifications, https://laravel.com/docs/5.3/notifications, pristupljeno 06.06.2017.

[17] Mail, https://laravel.com/docs/5.3/mail, pristupljeno 06.06.2017.

[18] Console Commands, https://laravel.com/docs/5.3/artisan, pristupljeno 07.06.2017.

[19] Task Scheduling, https://laravel.com/docs/5.3/scheduling, pristupljeno 07.06.2017.

[20] Views, https://laravel.com/docs/5.3/views, pristupljeno 08.06.2017.

[21] Blade Templates, https://laravel.com/docs/5.3/blade, pristupljeno 08.06.2017.

[22] Errors & Logging, https://laravel.com/docs/5.3/errors, pristupljeno 08.06.2017.

[23] Controllers, https://laravel.com/docs/5.3/controllers, pristupljeno 09.06.2017.

[24] Cache, https://laravel.com/docs/5.3/cache, pristupljeno 09.06.2017.

[25] Testing, https://laravel.com/docs/5.3/testing, pristupljeno 11.06.2017.

[26] Laravel 5 optimization commands, https://sentinelstand.com/article/laravel-5-optimization-commands, pristupljeno 12.06.2017.

[27] How to choose a PHP framework, https://opensource.com/business/16/6/which-php-framework-right-you, pristupljeno 14.06.2017.

SAŽETAK

Rad se bavi opisivanjem koncepta programskih okvira, prednosti i nedostataka Laravela i PHP-a u razvoju web rješenja. Teorijski dio sadrži objašnjenja osnovnih koncepta poput MVC arhitekture, objektno-relacijskog mapiranja i tijeka razvoja web rješenja korištenjem Laravel programskog okvira. Praktični dio sadrži korake razvoja web aplikacije za rezerviranje apartmana koja sadrži implementaciju autentikacije migracija i modela, usmjeravanja i filtriranja zahtjeva, kontrolera, pogleda, testiranja i drugih komponenti. Kraj rada opisuje prednosti korištenja Laravela za razvoj web rješenja u odnosu na razvoj bez programskih okvira ili s drugim sličnim programskim okvirima.

Ključne riječi: Laravel 5.3., MVC, ORM, Eloquent, programski okviri, web rješenja

ABSTRACT

Title: Advantages of Laravel framework in developing web solution

The thesis deals with describing the concepts of the software frameworks, advantages and disadvatages of Laravel and PHP in development of the web solutions. The theoretical part contains explanations of the basic concepts such as the MVC architecture, object-relational mapping and development course of the web solutions using Laravel software framework. The practical part contains the steps of apartments booking web application development, which includes the implementation of authentication, migrations and models, requests routing and filtering, controllers, views, testing and other components. The end of the thesis describes the advantages of using Laravel for web solutions development in relation to development without software frameworks or with other similar software frameworks.
Key words: Laravel 5.3., MVC, ORM, Eloquent, software frameworks, web solutions

ŽIVOTOPIS

Domagoj Šimić rođen je 24. studenog 1993 u Osijeku. Od rođenja živi u Osijeku, gdje stječe osnovnoškolsko obrazovanje u OŠ Ivana Filipovića od 2000. do 2008 godine. 2008.godine upisuje smjer prirodoslovna gimnazija u „Tehničkoj školi i prirodoslovnu gimnaziju Ruđera Boškovića“ u Osijeku te većinu razreda prolazi s vrlo dobrim uspjehom. Nakon završetka srednjoškolskog obrazovanja 2012. godine upisuje „Elektrotehnički fakultet“, preddiplomski studij računarstva u Osijeku kojeg završava 2015. godine. 2016. godine upisuje diplomski studij smjer računarstvo, modul programsko inženjerstvo na istom fakultetu, promijenjenog naziva u „Fakultet elektrotehnike, računarstva i informacijskih tehnologija“, kojeg trenutno pohađa.

Vlastoručni potpis: _____

PRILOZI

Na CD-u:

a) Diplomski rad „Prednosti izrade web rješenja korištenjem Laravel programskog okvira“ u „.docx“ formatu.
b) Diplomski rad „Prednosti izrade web rješenja korištenjem Laravel programskog okvira“ u „.pdf“ formatu.
c) Izvorni kod web aplikacije za rezerviranje apartmana.

Internet:

a) Laravel službena dokumentacija: https://laravel.com/docs/5.3/