Razvoj web-aplikacije pomoću Laravel radnog okvira

Sučec, Valentina

Master's thesis / Diplomski rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, University of Zagreb, Faculty of Humanities and Social Sciences / Sveučilište u Zagrebu, Filozofski fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:131:612278

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-07-17**

Sveučilište u Zagrebu
Filozofski fakultet
University of Zagreb
Faculty of Humanities
and Social Sciences

Repository / Repozitorij:

ODRAZ - open repository of the University of Zagreb
Faculty of Humanities and Social Sciences

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU
FILOZOFSKI FAKULTET
ODSJEK ZA INFORMACIJSKE I KOMUNIKACIJSKE ZNANOSTI
SMJER informacijske znanosti, istraživački
Ak. god. 2019./2020.

Valentina Sučec

Razvoj web-aplikacije pomoću Laravel radnog okvira

Diplomski rad

Mentor: dr. sc. Vedran Juričić, doc.

Zagreb, rujan 2020.

Izjava o akademskoj čestitosti

Izjavljujem i svojim potpisom potvrđujem da je ovaj rad rezultat mog vlastitog rada koji se temelji na istraživanjima te objavljenoj i citiranoj literaturi. Izjavljujem da nijedan dio rada nije napisan na nedozvoljen način, odnosno da je prepisan iz necitiranog rada, te da nijedan dio rada ne krši bilo čija autorska prava. Također izjavljujem da nijedan dio rada nije korišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

_____

(potpis)

Josipu, mojoj potpori u najljepšim i najtežim trenucima.

Lauri, mom razlogu.

Obitelji, bratu Miji koji su bili uz mene tijekom školovanja i poštivali moj put.

Sadržaj

1. Uvod

PHP programski jezik jedan je od najpopularnijih i najkorištenijih skriptnih jezika te su se zbog njegove široke upotrebe razvili mnogi radni okviri koji programerima olakšavaju razvoj aplikacija. Svrha radnih okvira je smanjivanje broja koraka pri izradi aplikacije, tj. preskakanje pisanja čestih i ponavljajućih dijelova koda pri izradi svake web-aplikacije. Jedan od radnih okvira čija popularnost raste otkad je razvijen je Laravel, besplatni radni okvir otvorenog koda.

U ovom radu je u prvom poglavlju pružen pregled osnovnih pojmova potrebnih za razumijevanje rada. Poglavlje objašnjava pojam radnog okvira, prednosti i nedostatke takvog alata, ali i informacije o PHP programskom jeziku te radnim okvirima za PHP. Drugo poglavlje daje informacije o web-aplikaciji te o model-prikaz-upravljač (engl. *Model View Controller; MVC)* arhitekturi na temelju koje Laravel radni okvir funkcionira. Cilj trećeg poglavlja je pružanje informacija o Laravelu, njegovoj povijesti, značajkama te o programima koji su potrebni za rad istog, ali i o funkcionalnostima koje nudi. Četvrto poglavlje sadrži usporedbu najpopularnijih PHP radnih okvira prema definiranim kriterijima, ali i usporedbu prema popularnosti temeljenu na aktualnim podacima. Sljedeća dva poglavlja daju pregled strukture direktorija novog Laravel projekta, a nakon toga i analizu komponenti pri stvaranju novog projekta. Poglavlje o analizi komponenti sadrži postupak izrade pojedinih komponenti prema uputama iz dokumentacije, ali i kroz primjere pri izradi aplikacije *Bebapapa* koja je razvijena u sklopu ovog rada. Posljednje poglavlje daje pregled cijele aplikacije te način razvoja iste.

2. Radni okvir

Kako bi se olakšao i ubrzao razvoj web-aplikacija, osim razvoja i pisanja novih programskih jezika, došlo je i do razvoja radnih okvira za određene programske jezike. U ovom poglavlju detaljnije je objašnjen pojam radnog okvira, prednosti i nedostaci istih te je definiran PHP programski jezik i PHP radni okviri.

2.1. Pojam

Radni okvir (engl. *framework)* podrazumijeva platformu koja se sastoji od različitih komponenti, tj. kolekcije različitih pomoćnih funkcija, klasa, biblioteka i aplikacijskih programskih sučelja (engl. *Application Programming Interface; API)*, a namjena mu je olakšavanje posla programerima i ubrzavanje procesa izrade aplikacije. Radni okvir ima neke značajke koje ga razlikuju od ostalih biblioteka, a to su (Christensson, 2013):

- zadano ponašanje koje podrazumijeva da se radni okvir prije prilagodbe ponaša na način specifičan za korisnikovo djelovanje
- inverzija kontrole podrazumijeva da je tok kontrole unutar okvira određen od strane samog okvira, a ne od strane korisnika
- proširivost omogućava korisniku proširenje okvira zamjenom zadanog koda vlastitim
- nepromjenjivost koda omogućava korisniku da proširi okvir, ali ne i da mijenja kod.

2.2. Prednosti i nedostaci

Neke od glavnih prednosti korištenja radnih okvira su ubrzavanje izrade prilagođenih (engl. *custom)* web-aplikacija gdje programeri pri izradi web-aplikacija moraju zadovoljiti kompleksne zahtjeve klijenata te moraju pružiti bogato korisničko iskustvo. Radni okviri pružaju svojstva i dijelove kodova koji ubrzavaju zadovoljenje takvih potreba. Pojednostavljeno je i održavanje aplikacija i to korištenjem model-pogled-upravljač arhitekture gdje programeri mogu podijeliti aplikaciju na te tri komponente i razdijeliti različite slojeve aplikacije. Pri korištenju radnih okvira nema potrebe za pisanjem dodatnog koda jer se u radnim okvirima nudi i mogućnost generiranja koda. Radni okviri omogućavaju efikasniji rad s bazama podataka, npr. neki radni okviri pružaju sustave za objektno-relacijsko mapiranje (engl. *object relational mapping, ORM*) koji izvršavaju operacije nad bazama podataka bez pisanja *sql* koda. Automatiziranje čestih i ponavljajućih zadataka kao što su autentikacija, *URL* mapiranje i sl.

skraćuju vrijeme izrade web-aplikacije i olakšavaju posao programera. Upotreba radnog okvira ne zahtijeva povećanje troškova pri izradi web-aplikacija s obzirom na to da su radni okviri najčešće otvorenog koda (engl. *open source*). Ugrađena sigurnosna svojstva i mehanizmi pojednostavljuju zaštitu web-stranica od postojećih sigurnosnih prijetnji. Korištenje radnih okvira omogućilo je brže i efikasnije testiranje jedinice (engl. *unit testing)* s već ugrađenim postavkama za testiranje (Smijulj & Meštrović, 2014).

Loše strane korištenja radnih okvira uključuju svakako razliku u kvaliteti različitih radnih okvira s obzirom na to da se radi o besplatnim, dostupnim alatima te se razlikuju po broju korisnika i kvaliteti podrške. Manjak mogućnosti izmjene nekih osnovnih struktura samih radnih okvira može otežati posao programera ili ga čak prisiliti na korištenje točno određenih alata. Važno je napomenuti i da će pri korištenju radnog okvira programeri morati naučiti sam radni okvir, a ne i programski jezik za koji je namijenjen (npr. PHP), tj. upotreba radnog okvira može dovesti do toga da korisnik uloži vrijeme i trud u učenje radnog okvira, a da zanemari učenje jezika jer nema potrebe za pisanjem dodatnog koda. Upotreba radnog okvira može utjecati i na brzinu i performanse web-stranica, npr. programer izrađuje manju i jednostavniju aplikaciju, a radni okvir sadrži neke dodatne značajke koje onda utječu na rad stranice (Smijulj & Meštrović, 2014).

2.3. PHP radni okviri

„**PHP** je jednostavan, ali moćan programski jezik kreiran za stvaranje HTML (engl. *HyperText Markup Language*) sadržaja.“ (Tatroe & MacIntyre, 2020, str. 1). PHP je skriptni programski jezik koji se može koristiti na dva osnovna načina (Tatroe & MacIntyre, 2020, str. 1-2):

1. skriptiranje na strani poslužitelja (engl. *server-side scripting*) – PHP je kreiran za kreiranje dinamičkih web-sadržaja te omogućava generiranje HTML-a za što je potreban PHP parser i web-server preko kojeg se šalju kodirani dokumenti
2. skriptiranje naredbenog retka (engl. *command-line scripting*) – PHP može slati skripte i preko naredbenog sučelja kao npr. Perl ili Unix; mogu se koristiti skripte naredbenog retka za kreiranje sigurnosnih kopija (engl. *backup*) ili za parsiranje logova.

Prema Tatroe i MacIntyre (2020), prvu verziju PHP-a napravio je Rasmus Lerdorf 1994. godine. Prva verzija je evoluirala kroz prve četiri verzije koje su dovele do razvoja složenog, ali i kvalitetnog jezika kakav je danas. Do danas je izdana verzija 7.4, a prema istraživanjima i dalje se najviše koristi verzija 5.0, a PHP koristi nekoliko desetaka milijuna domena te se

smatra najkorištenijim skriptnim jezikom u razvoju web-aplikacija zbog svoje fleksibilnosti te jednostavnosti. Kako razvoj web-aplikacija postaje sve kompliciraniji, pri razvoju se koriste poslovna logika, upiti baze podataka te prezentacijski dio aplikacije čime se dovodi u pitanje održavanje i skalabilnost aplikacije. Kako bi se riješili navedeni problemi, došlo je do razvoja PHP radnih okvira. **PHP radni okviri** imaju zajedničke značajke, a ujedno i prednosti (Laaziri, Benmoussa, Khoulji, & Larbi Kerkeb, 2019):

- Model-prikaz-upravljač (MVC) arhitektura koja osigurava brz razvoj aplikacije
- CRUD (engl. *Create, Read, Update and Delete*) operacije koje omogućavaju preuzimanje podataka iz baze bez pisanja složenih upita
- „ne ponavljaj se“ načelo (engl. *DRY – don't repeat yourself*) kojim je osiguran maksimalan utjecaj minimalne količine koda
- osiguravanje skalabilnosti sustava što omogućava rast i širenje radnog okvira
- povećanje sigurnosti web-aplikacije od različitih sigurnosnih prijetnji.

Neki od najpopularnijih PHP radnih okvira (osim Laravela) su Symphony, CodeIgniter i CakePHP te su detaljnije opisani u 4. poglavlju.

3. Web-aplikacija i Model-View-Controller (MVC) arhitektura

Pri razvoju web-aplikacije važno je održavanje i prilagodba iste, a to najviše ovisi o veličini same aplikacije i njenoj strukturi. Boljom strukturom postiže se lakše izmjenjivanje i nadopuna koda, bolji pregled aplikacije u svakom trenutku te je zbog toga važno poznavati arhitekturu na kojoj se temelji neki radni okvir u kojem se aplikacija razvija. U ovom poglavlju objašnjen je pojam web-aplikacije, ali i MVC arhitekture koju koristi Laravel radni okvir.

3.1. Web-aplikacija

„**Web-aplikacija** je računalni program koji povezuje web-preglednike i web-tehnologiju u svrhu izvršavanja nekog zadatka na internetu“ (Gibb, 2016). Web-aplikacije su obično napisane u programskim jezicima kao što su JavaScript i HTML jer se ti jezici „oslanjaju“ na preglednike kako bi se program izvršio. Neke web-aplikacije su dinamičke i zahtijevaju serversku obradu, dok su druge statičke i ne zahtijevaju obradu od strane servera (Gibb, 2016). Web-aplikacije funkcioniraju tako da (Gibb, 2016):

1. Korisnik šalje zahtjev web-serveru preko interneta (preko preglednika ili preko korisničkog sučelja aplikacije).
2. Web-server prosljeđuje zahtjev odgovarajućem serveru web-aplikacije.
3. Server web-aplikacije izvršava traženi zadatak (npr. pretraživanje baze podataka) i generira rezultate.
4. Server web-aplikacije šalje rezultat u obliku traženih podataka web-serveru.
5. Web-server šalje povratnu informaciju klijentu i prikazuje rezultat korisniku.

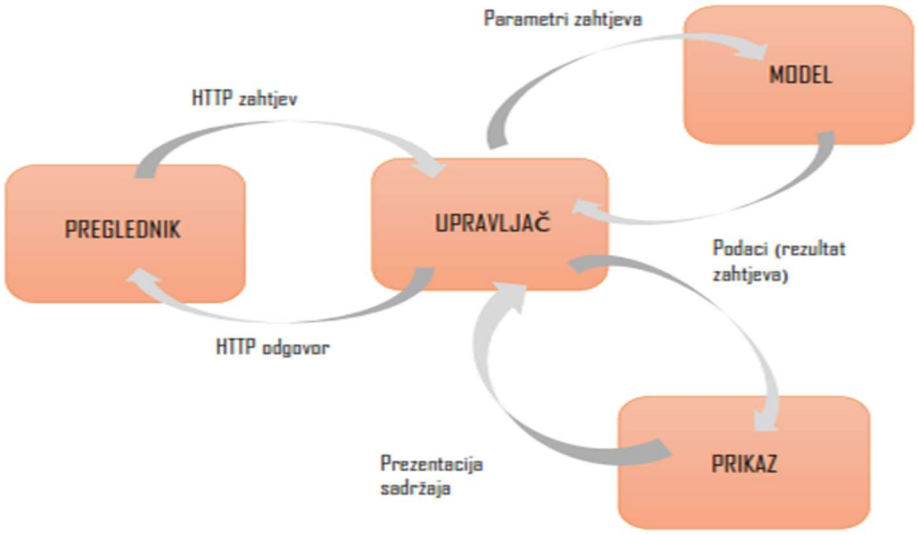
Prema Gibbu (2016), prednosti web-aplikacija su pristup istoj verziji svim korisnicima, ali i neovisnost o operacijskom sustavu, ne koriste prostor tvrdog diska (engl. *hard drive*), smanjuju troškove za poduzetnike i krajnje korisnike s obzirom na manje potrebe održavanja i pružanja podrške. Web-aplikacije su najčešće korištene za servise za slanje e-mail pošte, web-trgovina i sl., a neki od najpoznatijih primjera web-aplikacija su Gmail, Yahoo, Google Docs, Amazon i sl.

3.2. Model-View-Controller (MVC) arhitektura

MVC (skraćeno od engl. ***M**odel **V**iew **C**ontroller*) je standardni obrazac dizajna koji se koristi u web-dizajnu, odnosno u dizajnu web-aplikacija. Prednosti korištenja ovog obrasca su različite mogućnosti poput proširivosti, održavanja i ponovne upotrebe (Jadhav & Sodawala, 2015). Prijašnje inačice takve arhitekture sadržavale su sve komponente na jednom mjestu što je rezultiralo otežanim čitanjem koda, pronalaženjem i ispravljanjem grešaka te manjom funkcionalnošću i fleksibilnošću (Smijulj & Meštrović, 2014).

MVC inačica se sastoji od komponenti gdje je svaka zadužena za obavljanje specifičnih funkcija, a to su (Jadhav & Sodawala, 2015):

- **Model** (engl. *Model*) – ova razina aplikacije je odgovorna za radnu logiku aplikacije, poput pristupa i obrade baze podataka te pruža biblioteku klasa koje se koriste više puta u aplikaciji. Tipični primjeri su dijelovi aplikacije koji uređuju baze podataka, upravljaju dostavom e-pošte ili služe za provjeru valjanosti i autentičnosti.
- **Prikaz** (engl. *View*) – razina prikaza obično se smatra web-dizajnom odnosno predloškom web-aplikacije ili web-stranice. Zaslužna je za izgled i dojam web-stranice ili aplikacije, a omogućuje prikupljanje informacija od korisnika kao i pružanje informacija. Tehnologije koje se koriste u prikaznom sloju su obično HTML, CSS (engl. *Cascading Style Sheets*) i JavaScript.
- **Upravljač** (engl. *Controller*) – upravljač je svojevrsni „handler“ koji upravlja funkcionalnošću modela i spaja ga s izgledom prikaza te prosljeđuje informacije između dva sloja ovisno prema zahtjevima web-aplikacije odnosno web-stranice. Upravljač mora uzeti podatke iz modela i preoblikovati ih prikladan oblik koji će se prikazati u sloju prikaza kako bi se informacija mogla prenijeti do krajnjeg korisnika. Osim što je odgovoran za protok informacija, također je bitan za eventualne pogreške tijekom izvršavanja programa.



Slika 1. Grafički prikaz rada MVC arhitekture (Smijulj & Meštrović, 2014)

Slika 1 prikazuje proces koji počinje zahtjevom korisnika (upisivanje web-adrese u pregledniku) kojeg prvo analizira upravljač (koji sadrži funkcije koje je potrebno izvršiti) i nakon toga upravljač prosljeđuje modelu i prikazu potrebne parametre, ali i kontrolira (upravlja) daljnjim slijedom procesa. Model radi s podacima, kao što je ranije navedeno, komunicira s bazom podataka, dodatno obrađuje primljene podatke te vraća rezultate upravljaču. U nekim slučajevima upravljač dobivene podatke šalje drugim procesima te će oni biti naknadno obrađivani ili, najčešće, upravljač završava proces pozivajući prikaz koji onda generira zadani predložak po kojem će se podaci prikazati. Takav predložak (HTML kod) upravljač šalje prema korisniku, tj. pregledniku (Smijulj & Meštrović, 2014).

4. Laravel radni okvir

Laravel je besplatni radni okvir za razvoj web-aplikacija temeljen na PHP-u, a cijeli izvorni kod Laravela dostupan je na GitHub-u pod MIT (engl. *Massachusetts Institute of Technology*) licencom. Laravel je prema mnogim web-stranicama jedan od najpopularnijih PHP radnih okvira. Laravel je popularnost stekao zahvaljujući jednostavnoj sintaksi, jednostavnosti i detaljnoj dokumentaciji. Laravel je napisao Taylor Otwell (Stauffer, 2019, str. 3).

4.1. Povijest Laravela

Prva verzija Laravela objavljena je u lipnju 2011. godine te je sadržavala prilagođeni *Eloquent* (sustav za objektno-relacijsko mapiranje), mehanizam za kontrolu toka zahtjeva u aplikaciji (engl. *routing*), modele, podršku za autentikaciju korisnika i ostale mehanizme. Nakon prve verzije, već u studenom 2011. godine objavljena je druga, a u veljači 2012. godine i treća verzija. One su uključivale upravljače (engl. *controllers*), jedinično testiranje (engl. *unit testing*), sučelje naredbenog retka, inverziju kontrole te migracije i *Eloquent* relacije (Stauffer, 2019, str. 2-5).

Laravel 4 (svibanj, 2013) označava svojevrstan preokret u razvoju Laravela kada je Otwell potpuno preradio Laravelov kod. Razlog ponovnog pisanja koda je Composer koji je do tada postao industrijski standard te je Laravel „prerađen“ u skup paketa kojeg Composer (alat za upravljanje paketima) pretvara u organizirani radni okvir. Nova verzija donijela je i opciju *seedinga* baza podataka (popunjavanje baze podataka testnim podacima). Peta verzija – Laravel 5 objavljen je u veljači 2015. godine te donosi novu strukturu mapa te *Socialite* za autentikaciju korisnika društvenih mreža, nove prikaze i druge komponente. Laravel 6 objavljen je u rujnu 2019. godine i najznačajnija promjena je tzv. semantičko verzioniranje (engl. *semantic versioning*). Posljednja verzija koja je objavljena je Laravel 7 (ožujak, 2020) te uključuje poboljšanja ranijih verzija Laravela, (Stauffer, 2019, str. 2-5).

4.2. Značajke Laravela

Bez obzira na to koristi li se samo PHP lokalno na računalu, Vagrant na virtualnom računalu (engl. *virtual machine*) ili lokalni serveri kao što su MAMP/WAMP/XAMPP, Laravel zahtijeva neke nužne postavke kao što su PHP verzija 7.2.5 (ili veća) te različite PHP ekstenzije (JSON, Ctype, OpenSSL, XML, Tokenizer i dr.). Instaliranje svih traženih komponenti može

se izbjeći instaliranjem Laravel Homestead virtualnog računala. Analiza značajki Laravela u ovom će radu biti prikazana kroz objašnjenja nekih osnovnih alata potrebnih za kreiranje novog Laravel projekta i analize novostvorenih direktorija i značajki stvorenih pomoću Composera i PHP Artisana.

4.2.1. Laravel Homestead (Windows) – VirtualBox i Vagrant

Laravel Homestead je alat koji se koristi za postavljanje lokalne razvojne okoline (engl. *local development environment*) te se može koristiti na Windows, Linux i Mac operacijskim sustavima (Stauffer, 2019, str. 12-13). Homestead je službeni, unaprijed konfiguriran Vagrant paket koji korisniku pruža gotovu razvojnu okolinu sa svim potrebnim alatima za razvoj web-aplikacije. Da bi se Homestead mogao pokrenuti, potrebno je instalirati virtualno računalo, npr. VirtualBox te ranije spomenuti Vagrant, softverski alat za korištenje virtualnih okolina. Vagrant je nužno instalirati jer je Homestead kreiran za Vagrant (Laravel Homestead). Homestead se dodaje Vagrant instalaciji kodom:

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

Nakon dodavanja je potrebno „klonirati“ službeni Laravel repozitorij na računalo, nakon čega se kreira Homestead.yaml datoteka koju je potrebno dodatno podesiti. Homestead.yaml datoteka sastoji se od niže navedenih redaka koji su pobliže objašnjeni:

```
---
ip: "192.168.10.21"
memory: 1024
cpus: 1
provider: virtualbox
```

IP (engl. *Internet Protocol*) polje označava koju će IP adresu Homestead poslužitelj pratiti, a u ovom slučaju to je 192.168.10.21, zatim je navedena maksimalna količina memorije koju će moći koristiti (1024), koliko će procesora (CPUs) koristiti (1) te pružatelja usluge što je virtualbox.

```
authorize: c:/Users/Valentina/.ssh/id_rsa.pub

keys:
    - c:/Users/Valentina/.ssh/id_rsa
```

U ovim se redcima podešava *ssh* (engl. *Secure Shell*) ključ za Homestead, tj. podešava se putanja do datoteke u kojoj se nalazi ključ. Javni ključ postavlja se pod *authorize*, a privatni ključ pod *keys* opcijom.

```
folders:
    - map: c:/Homestead_Projects
      to: /home/vagrant/Code
```

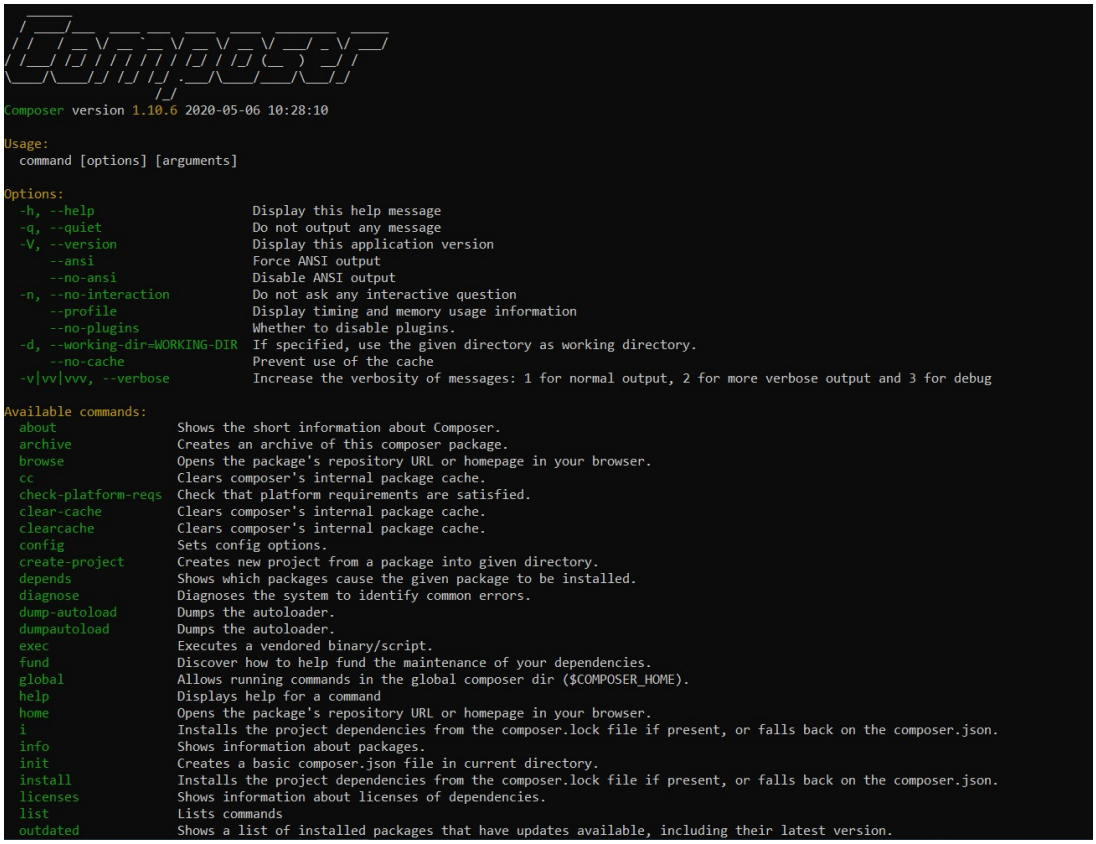
Sljedeći je korak povezivanje mapa koje će koristiti i lokalno računalo i Vagrant; postavlja se direktorij na lokalnom računalu (*c:/Homestead_Projects*) i u sljedećem retku je mapa na virtualnom serveru. Prethodni korak osigurava da se pri mijenjaju bilo kakve datoteke unutar tih direktorija sve promjene sinkroniziraju i na lokalnom računalu i na Homesteadu. U tim mapama bit će spremljeni Laravel projekti.

```
sites:
    - map: bebapapa.test
      to: /home/vagrant/Code/Bebapapa/public
```

Na kraju se još podešava mapiranje domene u direktorij u Homestead okolini čime se osigurava kreiranje baze podataka u Vagrantu s nazivom „*homestead*“. Nakon izmjene *.yaml* datoteke, potrebno je domenu koja je postavljena u *.yaml* datoteci dodati i u *hosts* datoteku na lokalnom računalu koja će preusmjeriti zahtjev za Homestead web-stranicu u Homestead okolinu. U postojeću *hosts* datoteku potrebno je dodati samo IP adresu koja je podešena u *.yaml* datoteci. Nakon navedenih izmjena, instalacija Homesteada je gotova i potrebno je samo još „podići“ virtualnu okolinu. To će se izvršiti naredbom **`vagrant up`** i nakon toga **`vagrant ssh`**, naredbom koja služi za autentikaciju.

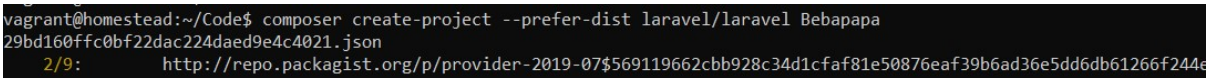
4.2.2. Composer

Composer je alat koji je osnova modernog PHP razvoja. To je alat za upravljanje paketima (engl. *package manager*), a omogućava instalaciju i ažuriranje svih paketa potrebnih pri razvoju aplikacije. Composer je prije instalacije potrebno skinuti sa službene web-stranice *https://getcomposer.org/.* Composer je potreban za instaliranje i ažuriranje Laravela (Stauffer, 2019, str. 12-15). Nakon preuzimanja programa, potrebno ga je instalirati izvršavajući naredbu **`composer install`**, a na slici 2 vidljiv je rezultat instalacije s popisom dostupnih naredbi.



Slika 2. Composer

Kako bi se kreirao novi projekt, potrebno je u naredbeni redak unijeti **`composer -prefer-dist create-project laravel/laravel nazivProjekta`** gdje laravel/laravel označava ***ime razvijatelja (engl. developera)/naziv paketa***. Nakon izvršavanja ove naredbe kreirat će se poddirektorij u direktoriju *nazivProjekta* koji sadržava kostur Laravel projekta kojeg je potrebno dalje razviti. Prema zadanim značajkama, stvaranjem novog projekta stvara se datoteka *composer.json* u kojoj su u obliku JavaScript Object Notation (JSON) teksta navedeni svi instalirani paketi aplikacije, a instalacijom novih paketa ta se datoteka ažurira.



Slika 3. Kreiranje Laravel projekta

Na slici 3 vidljivo je pokretanje naredbe za kreiranje projekta pod nazivom *Bebapapa*. Nakon instalacije projekta, Composer ažurira i mapu *vendor* koja sadrži sve pakete nužne za rad aplikacije te datoteku *composer.lock* koja sadrži sve informacije o verziji web-aplikacije.

4.2.3. Artisan

Artisan je Laravelovo sučelje naredbenog retka i njegove naredbe pomažu bržem radu na web-aplikaciji. Artisan se koristi za pokretanje migracija, kreiranje raznih podataka u bazama podataka te za mnoge druge zadatke koji se izvode samo jednom pri kreiranju aplikacije (Stauffer, 2019, str. 42). Artisanu se pristupa iz direktorija u kojem je kreiran Laravel projekt. S obzirom na to da se lista naredbi mijenja ovisno o instaliranim paketima ili o kodu aplikacije, dobro je provjeriti koje su naredbe dostupne (Stauffer, 2019, str. 201). Kako bi se provjerila lista naredbi, potrebno je u naredbeni redak unijeti naredbu **`php artisan list`** ili samo **`php artisan`**, a rezultat je vidljiv na slici 4.

```
vagrant@homestead:~/Code/Bebapapa$ php artisan list
Laravel Framework 7.12.0

Usage:
  command [options] [arguments]

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

Available commands:
  clear-compiled       Remove the compiled class file
  down                 Put the application into maintenance mode
  env                  Display the current framework environment
  help                 Displays help for a command
  inspire              Display an inspiring quote
  list                 Lists commands
  migrate              Run the database migrations
  optimize             Cache the framework bootstrap files
  serve                Serve the application on the PHP development server
  test                 Run the application tests
  tinker               Interact with your application
  ui                   Swap the front-end scaffolding for the application
  up                   Bring the application out of maintenance mode
 auth
  auth:clear-resets    Flush expired password reset tokens
 cache
  cache:clear          Flush the application cache
  cache:forget         Remove an item from the cache
  cache:table          Create a migration for the cache database table
 config
  config:cache         Create a cache file for faster configuration loading
  config:clear         Remove the configuration cache file
 db
  db:seed              Seed the database with records
  db:wipe              Drop all tables, views, and types
 event
  event:cache          Discover and cache the application's events and listeners
  event:clear          Clear all cached events and listeners
  event:generate       Generate the missing events and listeners based on registration
  event:list           List the application's events and listeners
 key
  key:generate         Set the application key
 make
```

Slika 4. Php Artisan List

Kada je potrebno objašnjenje neke naredbe, može se pozvati i **`help`** naredba. Potrebno je samo upisati **`php artisan help (naziv naredbe)`** za koju je potrebno objašnjenje npr. **`php artisan help tinker`**. Na slici 5 vidljiv je rezultat ove naredbe.

```
vagrant@homestead:~/Code/Bebapapa$ php artisan help tinker                                    Description:
  Interact with your application

Usage:
  tinker [options] [--] [<include>...]

Arguments:
  include                  Include file(s) before starting tinker

Options:
      --execute[=EXECUTE]  Execute the given code using Tinker
  -h, --help               Display this help message
  -q, --quiet              Do not output any message
  -V, --version            Display this application version
      --ansi               Force ANSI output
      --no-ansi            Disable ANSI output
  -n, --no-interaction     Do not ask any interactive question
      --env[=ENV]          The environment the command should run under
  -v|vv|vvv, --verbose     Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
vagrant@homestead:~/Code/Bebapapa$
```

Slika 5. Php artisan help

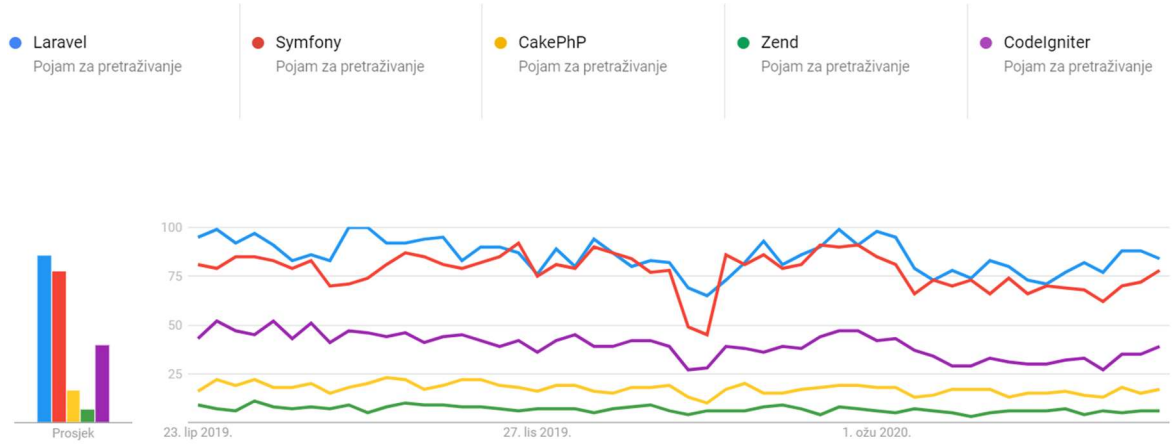
Tinker je tzv. REPL (engl. *Read Evaluate Print Loop*), tj. petlja koja prima unos od strane korisnika, evaluira ga, isprinta odgovor i ponavlja taj postupak. Tinker služi za pokretanje testnih interakcija s modelima, eventima i dr. unutar aplikacije. Tinker se poziva naredbom **`php artisan tinker`** te se zatim otvara polje za unos testnih naredbi što olakšava interakciju s bazom podataka s ciljem isprobavanja novih ideja ili dijelova koda koje ne treba spremati u aplikaciju prije testiranja.

5. Usporedba Laravela s drugim radnim okvirima

PHP radni okvir koji bira velik broj programera je Laravel te je zbog svoje popularnosti tema ovog rada. U ovom poglavlju Laravel je uspoređen s drugim poznatijim PHP radnim okvirima prema određenim kriterijima te su navedene njihove prednosti i nedostaci.

5.1. Usporedba prema popularnosti

Korištenjem radnih okvira povećava se kvaliteta krajnjeg proizvoda i standardizacija web-aplikacije. Danas je na tržištu velik broj radnih okvira koji nude različite opcije, značajke i mogućnosti koje olakšavaju stvaranje novih, prilagođenih web-aplikacija. Svaki od ponuđenih radnih okvira ima neke prednosti i nedostatke te je važno istražiti i odlučiti se za onaj koji najbolje odgovara zahtjevima aplikacije koja će se razvijati. Ranijim istraživanjima određeni su neki kriteriji koji se smatraju najvažnijima pri odabiru odgovarajućeg radnog okvira. Prema Variscou (2016), najvažniji kriteriji za odabir odgovarajućeg radnog okvira su vrijeme potrebno za učenje, tehničke karakteristike, angažman zajednice i posljedice nekih izbora. Za učenje rada svakog radnog okvira potrebno je neko vrijeme te je važno odlučiti koliko vremena treba uložiti, koje predznanje korisnik ima i sl. Najvažnije tehničke karakteristike koje treba provjeriti su mogućnosti, kompatibilnost i je li to odgovarajući radni okvir za strukturu nove aplikacije. Angažman zajednice je važan, tj. što više programera koristi neki radni okvir, veći je broj izvora za učenje i povratnih informacija o dobrim i lošim stranama nekog radnog okvira (Varisco, 2016). Najpopularniji PHP radni okviri u 2020. godini su redom Laravel, Symfony, CodeIgniter, CakePHP i Zend (Google Trends, Slika 6).



Slika 6. Google trends usporedba PHP radnih okvira

Laravel je vodeći u 2019. godini i prema Stack Overflow trendovima, a njegova je popularnost počela rasti 2014. godine. Istovremeno, s rastom popularnosti Laravela, ostali radni okviri doživjeli su pad popularnosti. Drugi po popularnosti na web-stranici Stack Overflow je Symfony, a slijede ga CodeIgniter, CakePHP i Zend. Na GitHub-u se razlikuju po broju zvjezdica koje imaju, vodeći je opet Laravel s 59,7 tisuća zvjezdica, a slijede ga Symfony s 23,5 tisuća, CodeIgniter broji 18 tisuća, CakePHP 8,2 tisuće i Zend s 5,7 tisuća. Tablica 1 prikazuje usporedbu poretka s ova tri izvora.

Tablica 1. Popularnost PHP radnih okvira

	Google Trends	*Stack Overflow*	*GitHub*
1.	Laravel	Laravel	Laravel
2.	Symfony	Symfony	Symfony
3.	CodeIgniter	CodeIgniter	CodeIgniter
4.	CakePHP	CakePHP	CakePHP
5.	Zend	Zend	Zend

Vidljivo je kako je popularnost PHP radnih okvira jednaka prema svim izvorima, čak i u pregledu popularnosti unutar dvije godine (2019. i 2020. godina). U ovom će radu biti prikazana usporedba ovih radnih okvira prema nekim njihovim značajkama, ali i prednostima i nedostacima. Prije svega, neke opće informacije o navedenim radnim okvirima prikazane su u tablici 2.

Tablica 2. Opće informacije o PHP radnim okvirima

Radni okvir	*Laravel*	*Symfony*	*CodeIgniter*	*CakePHP*	*Zend*
Prva verzija	9. 6. 2011.	4. 1. 2010.	25. 8. 2006.	16. 5. 2005.	28. 4. 2009.
GitHub	DA	DA	DA	DA	DA
Trenutna verzija	7.1	5.1.2	3.1.11	4	3

5.2. Usporedba prema odabranim kriterijima

Kriteriji po kojima je napravljena usporedba radnih okvira preuzeti su od Kaluža et al. (2019).

1. Generator koda koristi se za dijelove koda ili za cijele aplikacije, a može se koristiti u različitim programskim jezicima (Kaluža, 2019 prema Tomassetti, 2018).

Laravel koristi inteligentni generator koda koji se koristi za generiranje prikaza, upravljača, toka usmjeravanja (engl. *routes*), migracije, jezike i obrasce zahtjeva. Generator koda se u Laravelu instalira pomoću composera naredbom **`composer require crestapps/laravel-code-generator –dev.`**

Symfony ima generator koda naziva *SymfonyMakerBundle* koji se koristi za kreiranje praznih naredbi, upravljača, testova i dostupan je za Symfony verziju 3.4 i više. CodeIgniter i Cake PHP također koriste generator koda koji generiraju upravljače, modele i prikaze. Zend koristi generator koda koja pruža mogućnost generiranja proizvoljnog koda koristeći objektno-orijentirano sučelje, ali i mogućnost ažuriranja postojećeg koda.

2. Tzv. zdravlje projekta (engl. *project health*) je prema Variscou (2016) broj ljudi na internetu koji pričaju o nekoj tehnologiji i koriste nju i njezinu službenu dokumentaciju.

Laravel ima opširnu i potpunu službenu dokumentaciju dostupnu na službenim web-stranicama. Laravel ima i svoju zajednicu na Laracast stranici koja pruža i različite edukativne video vodiče (engl. *tutorials*) te diskusije vezane za rad Laravela. Upute za rad s Laravelom mogu se naći i na YouTubeu, ali i na Stack Overflowu postoji gotovo 150 tisuća oznaka Laravela.

Na službenim stranicama Symfonyja može se pronaći opširna službena dokumentacija, ali i vodiči i video vodiči za učenje različitih komponenti ovog radnog okvira. Symfony također nudi opciju službene obuke od strane stvaratelja Symfonyja, ali i stjecanje certifikata iz navedenog. Symfony je na Stack Overflowu označen 66 tisuća puta.

CodeIgniter je na Stack Overflowu označen više od 67 tisuća puta. Njegova dokumentacija sastoji se od korisničkog priručnika koji sadrži uvod, vodič te veliku količinu uputa kako nešto napraviti (engl. *„how to“ guide*). Dostupan je priručnik za sve verzije ovog radnog okvira.

CakePHP ima 30 tisuća oznaka na Stack Overflowu, a svoju dokumentaciju nudi *online*, na službenim web-stranicama, u *.pdf*, *.epub*, i video obliku. Radi se opširnoj dokumentaciji koja je otvorena zajednici za izmjenu, brisanje ili ispravljanje.

Zend nudi svoju dokumentaciju posebno za svaki od paketa koji sadrži, a sastoji se od 60+ paketa. Sva dokumentacija dostupna je preko GitHub-a ili se može instalirati pomoću Composera. Na Stack Overflowu označen je 20 tisuća puta.

3. Zadovoljstvo programera – ovaj kriterij izmjeren je prema rangiranju na *hotframeworks.com* gdje su rangirani po rezultatima s GitHub-a i Stack Overflowa.

Svi navedeni radni okviri odabrani su za usporedbu upravo zbog svoje popularnosti među programerima i rangiranju na ovim stranicama što je vidljivo ranije u poglavlju.

4. Broj programera koji rade u određenom radnom okviru – prema Kaluži (2019, prema Raibleu, 2010) ovaj se kriterij ocjenjuje prema broju programera kojima je neki od navedenih radnih okvira naveden kao vještina na LinkedIn profilu.

Laravel prema ovom kriteriju broji 289,000 korisnika, Symfony 141,000 korisnika, CodeIgniter 177,000 korisnika, CakePHP 57,000 korisnika, a Zend 85,000 korisnika.

5. Poslovni trendovi – ovdje će kriterij uključivati broj ponuđenih poslova na Indeed.com i LinkedIn-u za svaki od navedenih radnih okvira. Usporedba je napravljena i između ponude poslova diljem svijeta i u Hrvatskoj te je vidljiva u tablici 3.

Tablica 3. Poslovni trendovi

	Laravel	*Symfony*	*CodeIgniter*	*CakePHP*	*Zend*
LinkedIn – svijet	**3,739**	**11,008**	**5,631**	**3,701**	**2,451**
LinkedIn – Hrvatska	**0**	**2**	**1**	**3**	**0**
Indeed.com – svijet ili remote	**452 / 48**	**155 / 18**	**60 / 6**	**41 / 2**	**122 / 6**
MojPosao.net – Hrvatska ili remote	**2**	**2**	**0**	**0**	**0**

6. Sustav za predloške (engl. *template system*) za određeni radni okvir.

Laravel koristi *Blade* sustav za predloške, a Symfony *Twig*. Ono što im je zajedničko su nasljeđivanje predloška (engl. *template inheritance*) i blokovi ili sekcije (engl. *blocks or sections*) (Blade Templates, n.d.).

CodeIgniter ne nudi ugrađen sustav za predloške, već je potrebno integrirati neki drugi radni okvir s vlastitim sustavom za predloške (CodeIngiter4 User Guide, 2020) (Symfony Documentation, n.d.).

CakePHP koristi predloške koji omogućavaju poboljšavanje izgleda kroz prilagođene predloške, ali ne pružaju previše mogućnosti kao što je to npr. u slučaju Laravela (CakePHP Documentation, n.d.).

Zend pruža službeni sustav za predloške, Zend_View koji pruža mogućnost raspoređivanja objekata te mogućnost pružanja prilagođenih proširenja za pomoć (Getting Started with Zend Framework, n.d.).

7. Podrška za internacionalizaciju i lokalizaciju; „internacionalizacija podrazumijeva dizajniranje i razvoj proizvoda koji omogućava jednostavnu lokalizaciju za određeno tržište (kulturu, jezik i područje). Lokalizacija se odnosi na prilagođavanje proizvoda ili sadržaja (valuta, mjernih jedinica, fraza i sl.) tržištu na kojem će se prodavati (Beatty, 2011 prema W3C)".

Laravel pruža mogućnost lokalizacije tako da dohvaća nizove (engl. *strings*) u različitim jezicima omogućavajući tako jednostavnu višejezičnu podršku unutar aplikacije (Kaluža, 2019). U direktoriju *resources/language* postoji poddirektorij za svaki jezik koji aplikacija podržava (Localization, n.d.).

Symfony ima opciju konfiguracije prevoditelja koji podržava različite prevoditeljske formate kao što su, npr. PHP, JSON, CSV i sl. te podržava *gettext*. *GNU gettext* je paket koji omogućava programerima i prevoditeljima alate za stvaranje višejezičnih poruka (Introduction to gettext, n.d.).

CodeIgniter sadrži funkcije koje dohvaćaju jezične datoteke i linije teksta u svrhu internacionalizacije, a te se datoteke nalaze unutar direktorija pod nazivom jezika npr. *english*.

CakePHP za internacionalizaciju koristi vlastitu implementaciju *gettexta*. Zend također podržava *gettext,* ali i sadrži Zend_translate biblioteku koja služi za prevođenje (Getting Started with Zend Framework, n.d.).

8. Testiranje; ovaj kriterij odnosi se na podršku za testiranje unutar radnog okvira (Kaluža 2019, prema Raibleu 2010).

Laravel ima ugrađenu podršku za testiranje koja dolazi s PHPUnit radnim okvirom (za testiranje) i s postavljenom testnom datotekom za aplikaciju. CodeIgniter, Symfony, CakePHP i Zend također koriste PHPUnit.

9. Provjera (engl. *validation*)

Prema Kaluži (2019), Laravel pruža različite mogućnosti provjere dolaznih podataka. Osnovna značajka Laravelovog upravljača je *ValidatesRequest* koja uključuje različita pravila provjere valjanosti i kao takva predstavlja dobru metodu provjere dolaznog HTTP zahtjeva.

U Symfonyju se pruža mogućnost provjere samo modela, ne i ostalih objekata.

CodeIgniter ima ugrađenu biblioteku za provjeru (engl. *validation library*), a ona uključuje funkcije za provjeru određenih polja, za provjeru vrste i duljine podataka, za uklanjanje zlonamjernog koda i sl.

CakePHP pruža set pravila za provjeru koji uključuju pravila za e-mail adrese, poštanske brojeve, kreditne kartice i sl., a postoji mogućnost pisanja vlastitih pravila za provjeru. Zend također ima ugrađen *Zend/Validator* koja sadrži set najčešće korištenih funkcija za provjeru.

10. Licenca; svaki računalni program/tehnologija dolazi s nekom licencom koja korisniku daje prava korištenja istog, a proizvođaču osigurava zaštitu softvera od kopiranja i upotrebe.

S obzirom na to da se ovdje radi o tehnologijama otvorenog koda, Zend ima novu BSD (engl. *Berkeley Software Distribution*) licencu, a svi ostali radni okviri MIT licencu.

11. Mogućnost integracije rješenja treće strane (engl. *third-party solutions*)

Laravel ima mogućnost integracije rješenja treće strane koristeći aplikacijsko programsko sučelje (API) i pretvaranja dobivenog odgovora u podatkovnu strukturu. CodeIgniter i CakePHP također koriste API za integraciju rješenja treće strane. Symfony i Zend omogućavaju korištenje postojećih rješenja treće strane i instalaciju preko *Composera.*

12. Dodaci/priključci (engl. *add-ons/plugins*)

Svi navedeni radni okviri imaju mogućnost postavljanja dodataka (engl. *plugins*). U Laravelu se ta mogućnost instalira koristeći naredbeni redak unutar postojećeg, određenog projekta. U Symfonyju se mogućnost treba uključiti u *config/bundles.php* datoteci. CodeIgniter ima

dostupne dodatke koji se aktiviraju pozivanjem naredbe **`cake bake plugin nazivPlugina`**. U Zendu se dodaci pozivaju naredbom **`getPlugins()`**.

13. Prilagođenost izradi velikih aplikacija

Prema mišljenju kreatora Laravela, Taylora Otwella, Laravel može biti korišten za kompleksne aplikacije (i već je bio), te se pokazao kao dobra alternativa drugim PHP radnim okvirima u takvom zadatku (Otwell, 2017).

Symfony je stvoren za velike aplikacije i kao takav se koristi od samih početaka. Najbolji primjeri koji koriste Symfony su velike platforme poput Drupala i Magenta.

CodeIgniter je radni okvir male veličine (2 MB) te je namijenjen izradi malih aplikacija koje se pokreću na jednostavnijim serverima. Smatra se dobrim izborom za početnike s obzirom na opširnu dokumentaciju i jednostavnost korištenja, ali ne pruža dovoljno sigurnosnih značajki potrebnih za izradu većih aplikacija (Bhartia, 2020).

Na službenim stranicama Zenda stoji kako je u Zendu sve u obliku objekata zbog čega se kod može ponovno koristiti i nasljeđivati što omogućava Zendu korištenje velikog broja „ovisnosti“ (engl. *dependencies*) bez smanjenja performansi. CakePHP se također prema Stack Overflowu smatra dobrim izborom za izradu velikih aplikacija.

14. Prilagođenost početnicima

Prema ocjenama programera na web-stranici StackShare.io, Laravel je jednostavan za korištenje, ali „prenatrpan“ i može biti zbunjujuć za početnike, dok CodeIgniter smatraju jednostavnim i prilagođenim početnicima. CakePHP nema oznaku jednostavnog ili prilagođenog početnicima. Zend se smatra jednostavnim, ali ne i prilagođenom početnicima. Symfony nosi oznaku profesionalnog i teškog za svladati.

15. Porast u trendovima

Kao što je ranije navedeno, Laravelu popularnost raste, dok je drugima s rastom popularnosti Laravela vlastita pala (Google Trends).

6. Analiza strukture novostvorenog Laravel projekta

Nakon kreiranja novog Laravel projekta kreirat će se mapa s određenim direktorijima i datotekama koji čine kostur nove aplikacije. Direktoriji i datoteke koje se tamo nalaze su:

- *app* direktorij baza je aplikacije i sadrži modele, upravljače, naredbe te će se tamo spremati svi navedeni novi dijelovi aplikacije
- *bootstrap* direktorij sadrži *app.php* datoteku koja služi za „podizanje“ (engl. *boot*) radnog okvira te *cache* direktorij koji sadrži podatke predmemorije
- *config* direktorij sadrži sve konfiguracijske datoteke aplikacije
- *database* direktorij sadrži migracije baze podatka, *seed*-ove (koji služe za popunjavanje baze s testnim podacima) i tvornice (engl. *factory*) u kojima se definira oblik testnih podataka
- *public* direktorij je direktorij na koji se server usmjerava pri pokretanju web-stranice, a sadrži *index.php* datoteku koja je u ulozi početne točke aplikacije; važnost *public* direktorija leži u njegovoj korištenosti. Naime, on sadrži *CSS* i *JS* datoteke koje služe za uređivanje dizajna web-stranice; u novostvorenom Laravel projektu se po zadanim postavkama nalazi *css i javascript* datoteka, ali korisnik može stvarati i vlastite
- *resources* direktorij sadrži sve prikaze, jezične datoteke i potrebne sirove, nekompilirane CSS i JavaScript datoteke
- *routes* direktorij sadrži sve potrebne rute (tokove usmjeravanja) potrebne za rad aplikacije kao što su npr. HTTP (engl. *Hyper Text Transfer Protocol*) rute, konzolne rute (engl. *console routes*)
- *storage* direktorij čuva sve podatke predmemorije, logove, ali i služi kao lokalno spremište podataka
- *tests* direktorij sadrži testne skripte, npr. PHPUnit
- *vendor* direktorij sadrži sve datoteke potrebne za rad Composera. (Directory Structure, n.d.).

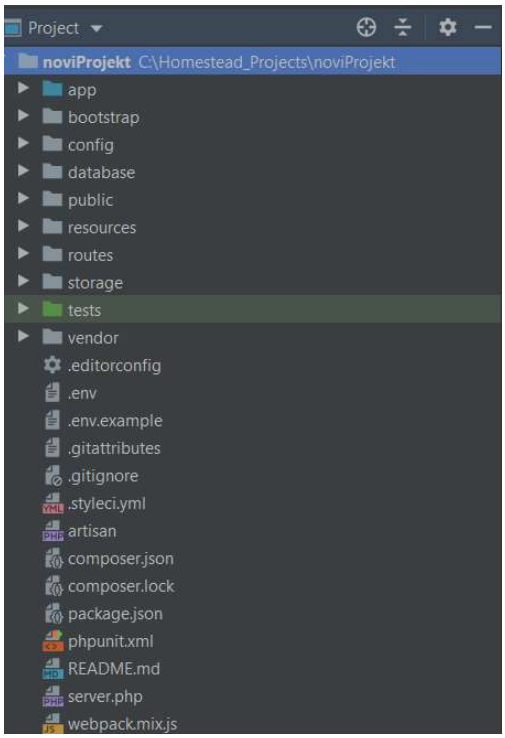
Datoteke koje koristi također imaju svoj značaj u radu aplikacije i pobliže su objašnjene (Stauffer, 2019, str. 41-43):

- *.editorconfig;* datoteka koja daje korisnikovom uređivaču teksta (engl. *text editor*) upute o Laravelovom standardu kodiranja
- *.env i env.example*; *.env* služi kao konfiguracijska datoteka nove web-aplikacije i svaki programer ju je u mogućnosti mijenjati, a *.env.example* je predložak koji svaka okolina

treba kopirati kako bi se kreirala vlastita *.env* datoteka, a ta je datoteka *git-ignored*, tj. namjerno zanemarena datoteka unutar programa

- *.gitignore i .gitattributes* – konfiguracijske Git datoteke
- *.styleci.yml* – konfiguracijska datoteka za postavljanje StyleCI alata za ispravljanje koda za uređivanje stilova
- *artisan* – datoteka koja omogućava pokretanje *Artisan* naredbi iz naredbenog sučelja
- *composer.json i composer.lock* – konfiguracijske datoteke za rad *Composera, .json* datoteku korisnik može uređivati, a *.lock* ne može
- *package.json* – datoteka koja sadrži upute za NPM (engl. *Node Packet Manager*)
- *phpunit.xml* – konfiguracijska datoteka za PHPUnit
- *readme.md* – datoteka koja sadrži osnovni uvod u Laravel
- *server.php* – sigurnosna kopija servera koja omogućava manje sposobnim serverima pregled Laravel aplikacije
- *webpack.mix.js* – neobvezna konfiguracijska datoteka za Mix; daje upute kako kompilirati i obrađivati sučelje aplikacije.

Na slici 7 prikazani su direktoriji i datoteke novog Laravel projekta.



Slika 7. Pregled direktorija i datoteka u novom Laravel projektu

6.1. *.env* datoteka

.env datoteka je konfiguracijska datoteka za novu web-aplikaciju i obično se podešava na početku rada na aplikaciji. Neke od stvari koje se mogu izmijeniti su:

`APP_KEY` – ključ za kriptiranje aplikacije; u slučaju da je polje prazno, javljat će se greška; „Nije određen ključ za kriptiranje aplikacije.“, a to će se javiti u slučaju da projekt nije stvoren pomoću Composera. U tom slučaju je potrebno pokrenuti naredbu **`php artisan key:generate`** i Laravel će ga izgenerirati (Stauffer, 2019, str. 20-21).

`APP_DEBUG` – Booleov operator koji određuje hoće li korisnici vidjeti greške za ispravljanje (engl. *debug errors*) ili ne (Stauffer, 2019, str. 20-21).

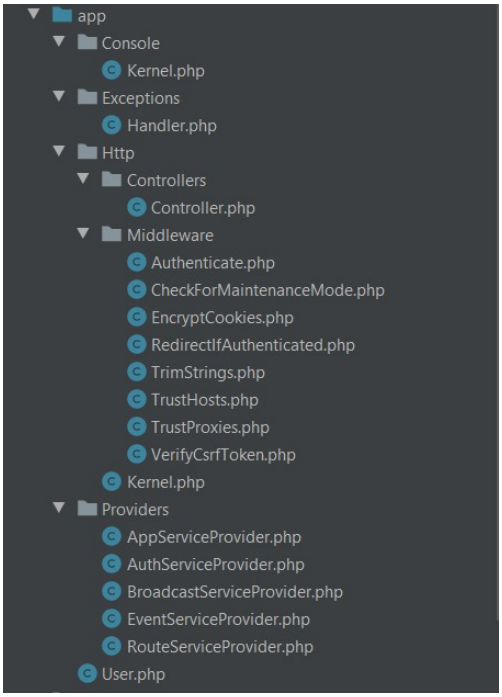
```
DB_DATABASE=laravel
DB_USERNAME=root

DB_PASSWORD=
```

Polja za postavljanje naziva baze podatka s kojom je povezana aplikacija te korisničkog imena i lozinke. U novom Laravel projektu polja će biti ispunjena kao u gore navedenom kodu.

6.2. *app* direktorij (Directory Structure, n.d.)

App direktorij sadrži poddirektorije *Console*, *Exceptions*, *Http*, *Providers* i model *User.php* (dio MVC arhitekture). *Console* sadrži *Kernel.php* datoteku koja sadrži sve *Artisan* naredbe. *Exceptions* sadrži datoteku *Handler.php* koja služi kao upravljač iznimkama (engl. *exception*) u aplikaciji. *Http* datoteka sadrži poddirektorij *Controllers* u kojem su spremljeni upravljači i drugi *Middleware* koji služi za sigurnost web-aplikacije, tj. za kontrolu korisničkih zahtjeva. Ovaj poddirektorij sadrži gotovo cijeli logički dio aplikacije. Poddirektorij *Providers* sadrži sve davatelje usluga aplikacije koji omogućuju pripremu aplikacije za buduće zahtjeve.

Prilikom izvršavanja **`make`** *Artisan* naredbi generirat će se drugi direktoriji unutar *app* direktorija. Npr., ako se izvrši naredba **`make:Mail`** kreirat će se direktorij app/Mail. Slika 8 prikazuje *app* direktorij.



Slika 8. App direktorij

6.3. node_modules direktorij

Ovaj direktorij neće biti kreiran nakon kreiranja samog projekta, već ga je potrebno instalirati. Ovaj direktorij stvara se naredbom **`npm install`** unutar naredbenog sučelja, a u mapi će se stvoriti skupina modula za korištenje. Te module moguće je koristiti ako postoji *Node.js* instaliran na računalu, a u primjeni pri izradi web-aplikacije npm je korišten za *frontend* dio aplikacije.

6.4. *database* direktorij

Kao što je ranije navedeno, unutar *database* direktorija nalaze se poddirektoriji *seeds, factories* i *migrations.* Prva dva poddirektorija koriste se za testiranje kako aplikacija radi s više zapisa unutar baze podataka. Npr. *factory* datoteke se koriste za generiranje testnih objekata u aplikaciji, a *seed* datoteke ispunjavaju tablice u bazi podataka. U *seed* datoteci se definira broj testnih zapisa koji će se generirati ili odnos između testnih objekata.

7. Analiza komponenti novog Laravel projekta

Ranije su objašnjeni osnovni pojmovi o komponentama, u ovom poglavlju bit će prikazano kreiranje komponenti i njihova uloga unutar Laravel projekta. Neki od primjera prikazani su na primjeru izrade projekta *Bebapapa.*

7.1. Modeli

Prema Staufferu (2019, str. 118-125) modeli služe za interakciju sa svakom tablicom unutar baze podataka, a pomoću njih je omogućeno umetanje novih zapisa ili postavljanje upita za podatke u tablicama. Laravel ima implementiran *Eloquent ORM* što je sustav za objektno-relacijsko mapiranje koji omogućuje stvaranje virtualne objektne baze podataka koja je namijenjena korištenju objektnog programskog jezika. Svrha Eloquenta je olakšavanje i ubrzavanje posla izbjegavanjem pisanja dugih *sql* upita te omogućava povezivanje modela s jednom tablicom u bazi podataka. Način na koji se omogućava je Laravelova konvencija naziva gdje se preporučuje naziv modela u jednini, a tablice u množini, npr. model će se zvati *Post.php*, a tablica *create_table_posts* i na osnovu naziva Eloquent povezuje model s tablicom.

Model se kreira pomoću *Artisan* naredbe **`php artisan make:model Post`**. Ovom naredbom dobit će se *app/Post.php* vidljiv na slici 9*:*

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
//
}
```

Slika 9. Post.php

Za rad sa zapisima iz baza podataka umjesto SQL upita koriste se *Eloquent* metode, a neke od njih su (Stauffer, 2019, str. 117-120):

`all` – služi za dohvaćanje svih zapisa iz tablice npr. svih korisnika društvene mreže, a bit će napisana u sljedećem obliku **`$users = User::all();`**

`get` – služi za dohvaćanje nekog podatka uz određeni uvjet npr. dohvaćanje korisnika koji ima objavljen post, **`$users = User::where('post', true)->get();`**

`first` – ova metoda dohvaća samo prvi zapis kao rezultat, npr. **`$users = User::first();`**.

U projektu Bebapapa izrađeni su modeli *Like.php*, *User.php* i *Post.php.* Model *User.php* sadrži model podataka za svakog registriranog korisnika ove društvene mreže.

```php
namespace App;

use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Overtrue\LaravelFollow\Traits\Followable;
use Illuminate\Database\Eloquent\Model;


class User extends Authenticatable
{
    use Notifiable;

    public function posts(){
        return $this->hasMany( related: 'App\Post');
    }

    public function likes()
    {
        return $this->hasMany( related: 'App\Like');
    }

    public function comments()
    {
        return $this->hasMany( related: 'App\Comment');
    }
     public function follows()
     {
         return $this->hasMany( related: Follow::class);
     }
```

Slika 10. Isječak koda datoteke User.php

Na slici 10 vidljivo je da su na početku samog modela definirane datoteke čije će se metode koristiti unutar modela pa su tako vidljive datoteke *Auth, Notifiable* i *Followable*.

Auth sadrži *MustVerifyEmail.php* i *Authenticatable.php* koji sadrže sve metode za zahtijevanje potvrde korisnikove e-mail adrese i za autentikaciju korisnika. *Followable.php* sadrži metode za određivanje pravila za praćenje na društvenoj mreži.

U modelu su nakon toga postavljene **relacije**. Povezanost tablica u bazama podataka često je podešena koristeći relacije koje podržava Eloquent (Eloquent: Relationships, n.d.):

- *jedan prema jedan*
- *jedan prema više*
- *više prema više*
- *jedan prema više kroz atribut*
- *polimorfne relacije*
- *više prema više polimorfnih relacija.*

Dalje će u radu biti definirane osnovne relacije. Relacije se definiraju kao funkcije unutar modela te se najčešće nazivaju prema tablici s kojom je model povezan. U projektu *Bebapapa* kreirano je nekoliko relacija, a na slici je vidljiva relacija jedan prema više koja se definira ako korisnik ima više objava u *User* modelu vidljivo na slici 11.

```
class User extends Authenticatable
{
    public function posts(){
        return $this->hasMany( related: 'App\Post');
    }
```

Slika 11. Relacija jedan prema više

Relacija jedan prema više tražit će definiranje inverzne relacije pozivajući metodu *belongsTo* u *Post* modelu vidljivo na slici 12.

```
class Post extends Model
{
    public function user()
    {
        return $this->belongsTo( related: 'App\User');
    }
```

Slika 12. belongsTo metoda

Relacija jedan prema jedan tražit će iste uvjete kao i metoda jadan prema više, a definirat će npr. da jedan korisnik ima samo jedan broj mobitela.

Sljedeća relacija koja je omogućena je više prema više i ona definira npr. da jedan korisnik ima više uloga unutar aplikacije, npr. više korisnika može imati ulogu „Admin“. Za takvu relaciju potrebno je kreirati tri tablice u bazi podataka. Baza će morati sadržavati tablicu *users, roles* i *role_user* koja će sadržavati *role_id* i *user_id*, tj. povezivat će prve dvije tablice pomoću primarnog ključa tih tablica. Takva relacije definirat će se unutar *User* i *Role* modela pomoću *belongsToMany* metode, npr. unutar *User* modela (Eloquent: Relationships, n.d.):

```
public function roles()
{
    return $this->belongsToMany('App\Role');
}
```

Nadalje, u modelu su definirana svojstva **`$fillable, $hidden, $casts.`** Svojstvo **`$fillable`** definira stupce u tablici *users* u koje je korisniku dozvoljeno upisivanje vrijednosti, **`$hidden`** definira stupce u tablici koji su skrivenog tipa, a **`$casts`** označava vrstu podatka u koju će se spremati informacije iz određenog stupca.

7.2. Prikazi (Views, n.d.)

Prikazi (engl. *views*) nalaze se u direktoriju *resources/views*. Moguće ih je pronaći u različitim poddirektorijima, a to su ovisno o namjeni, prema zadanim postavkama – *auth, includes* ili *layouts.* Također, postoje i neki koji nisu svrstani u ova tri poddirektorija. *Auth* poddirektorij sadrži prikaze generirane pri stvaranju novog projekta, a služe za autentikaciju korisnika. *Includes* poddirektorij sadrži pomoćne dijelove stranice koji se mogu uključiti u glavne predloške koji se nalaze u *layouts* poddirektoriju. Svaki od prikaza ima dodataka *.blade*, npr. *home.blade.php.* To označava *Blade* sistem za predloške koji se smatra velikom prednošću Laravela pri kreiranju novih web-aplikacija. Mogućnosti koje *Blade* pruža su izrada glavnog predloška što je prema zadanim postavkama *app.blade.php*. Moguće je taj predložak proširiti u slučaju kreiranja novih prikaza te *Blade* omogućava prikaz podataka generiranih upravljačem.

Uloga glavnih predložaka unutar Laravel projekta je kreiranje jednog predloška koji će biti isti za sve prikaze kako bi se izbjeglo ponavljanje istog koda pri kreiranju svakog prikaza. Zbog toga je omogućeno proširivanje određenog dijela koda glavnog predloška, tj. *@yield* oznakom moguće je označiti dio koda koji će se proširiti u nekom novom prikazu. Isto tako *@include* oznakom dodaje se kod od nekog drugog, pomoćnog predloška. Npr. u ovom projektu unutar

glavnog predloška nalazi se dio koda koji je niže napisan, a upućuje na *'content'* odjeljak unutar drugog pogleda gdje je proširen. Na slici 13 vidljiv je kod u glavnom predlošku.

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

Slika 13. Kod unutar glavnog predloška

Slika 14 prikazuje kod unutar drugog pogleda gdje se pod oznakom *@section* povezuje proširuje *'content'* koji je naveden u glavnom predlošku.

```
@section('content')
    @include('includes.message-block')
    <link href="{{ asset('css/dash.css') }}" rel="stylesheet">
    <section class="row new-post">
        <div class="col-md-6 offset-md-3">
                <header><h3>Podijeli objavu</h3></header>
            <form action="{{ route('post.create') }}" method="POST">
                <div class="form-group">
                    <textarea class="form-control" name="body" id="new-post" rows="5" pl
                </div>
                <button type="submit" class="btn btn-primary">Podijeli objavu</button>
                <input type="hidden" value="{{Session::token() }}" name="_token">
            </form>
            </div>
    </section>
```

Slika 14. Dashboard.blade.php

U ovom je kodu vidljiva i *@include* opcija koja uključuje pomoćni prikaz pod nazivom *message-block.blade.php,* a u ovom slučaju služi za prikaz eventualnih grešaka pri unosu podataka ili izvršavanju neke naredbe.

Na slici 14 vidljiv je dio koda kreiranog prikaza *dashboard.blade.php* (zbog njegove duljine nije prikazan cijeli kod). Na njemu će biti objašnjeno proširivanje i stvaranje novog prikaza. Za kreiranje novog prikaza potrebno je unutar poddirektorija *views* kreirati novu PHP datoteku s *.blade.php* ekstenzijom. Nakon toga se dodaje oznaka *@extends* kojom se proširuje glavni

predložak koji se nalazi u poddirektoriju *layouts* pod nazivom *app.* Nakon toga slijedi ©*section* koji se povezuje na oznaku *@yield* u glavnom predlošku, a zatvara se oznakom *@endsection* na kraju koda. Unutar ovog odjeljka pozvan je i pomoćni predložak za pokazivanje poruka o greškama. Nakon toga se poziva i ruta koja je kreirana u *web.php* datoteci unutar *routes* direktorija. Ta ruta nam omogućava povezivanje s upravljačem *PostController* koji poziva funkciju za kreiranje nove objave. Rute su detaljnije objašnjene u nastavku rada. Unutar prikaza pozvana je i *$posts* varijabla koja je definirana unutar upravljača *PostController* te su dodani uvjeti po kojima će se prikazivati post (ID posta).

7.3. Upravljači (Controllers, n.d.)

Unutar MVC arhitekture upravljači su klase koje organiziraju logički dio aplikacije, tj. prema Staufferu (2019, str. 42), glavna zadaća upravljača je upravljanje HTTP zahtjevom i prosljeđivanje istog ostatku aplikacije. Upravljači se unutar Laravel projekta nalaze u *http* poddirektoriju unutar *app* direktorija. Kod kreiranja novog Laravel projekta u tom direktoriju naći će se poddirektorij *Controllers* i unutar njega *Auth.* U *Auth* direktoriju nalaze se upravljači odgovorni za upravljanje postupkom autentikacije – od procesa registracije korisnika do resetiranja lozinke, prijave korisnika i sl. Svi novostvoreni upravljači pojavit će se unutar direktorija *Controllers.* Novi upravljač (Controllers, n.d.) se kreira naredbom **`php artisan make:controller PostController`** čime će se kreirati *PostController.php.* Datoteka koja će se generirati izgleda kao na slici 15.

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

use Illuminate\Http\Request;

class PostController extends Controller
{
//
}
```

Slika 15. PostController.php

Prema Laravelovoj konvenciji nazivlja, upravljači će se imenovati prema modelu kojim upravljaju, npr. *PostController* upravlja modelom *Post,* a *UserController* modelom *User.*

Na slici 16 prikazane su funkcije iz upravljača PostController.php kreiranog u projektu *Bebapapa*.

```
class PostController extends Controller
{
  public function getDashboard()
  {
    $posts = Post::orderBy('created_at', 'desc')->get();
    return view('dashboard', ['posts' => $posts]);
  }

  public function postCreatePost(Request $request)
  {
    $this->validate($request, [
      'body' => 'required|max:1000'
    ]);
    $post = new Post();
    $post->body = $request['body'];
    $message = 'Dogodila se pogreška';
    if ($request->user()->posts()->save($post)) {
      $message = 'Post je uspješno kreiran!';
    }
    return redirect()->route('dashboard')->with(['message' => $message]);
  }
```

Slika 16. PostController.php u novom projektu

Na primjeru su vidljive dvije funkcije od kojih funkcija *getDashboard()* koja definira varijablu *$posts* dohvaća postove korisnika i treba ih poredati po datumu kreiranja, silaznim redoslijedom u prikazu *dashboard.* Funkcija *postCreatePost* kao argument prima trenutni korisnikov unos i provjerava je li unesen u *body* polje koje je zadano s određenom maksimalnom duljinom unosa. Taj unos sprema kao novu objavu i ako je uspješno unesen vraća poruku da je post uspješno unesen i prikazuje prikaz *dashboard* s porukom o unosu.

7.4. Tok usmjeravanja – rute (engl. *routing*) (Basic Routing, n.d.)

Nakon što su postavljeni modeli, prikazi i upravljači, sve te komponente potrebno je povezati i za to služi tok usmjeravanja. Naime, u Laravelu je to omogućeno korištenjem klase *Route.* Unutar mape *routes* nalaze se datoteke koje Laravel sam učitava, a to su *api.php, channels.php, console.php* i *web.php.* Pri izradi jednostavnijih aplikacija, najčešće se uređuje samo *web.php* datoteka. Rute koje se definiraju u toj datoteci dostupne su unosom definiranog URL-a u preglednik. Prvoj ruti na slici 17 pristupa se preko adrese http://bebapapa.test/account.

```
Route::get('/account', [
   'uses'=>'UserController@getAccount',
   'as' => 'account'
]);

Route::post('/updateaccount', [
   'uses' => 'UserController@postSaveAccount',
   'as' => 'account.save'
]);
```

Slika 17. Primjer rute

Kako bi se definirala ruta, potrebno je prvo pozvati klasu *Route,* a nakon toga vrsta zahtjeva što je u primjeru na slici *get* ili *post.* Nakon toga se u zagradi definira ime rute, a to je */acoount* ili */updateaccount*. Nakon toga se otvara nova zagrada u kojoj se definira koja funkcija unutar kojeg upravljača će koristiti navedenu rutu. U tom slučaju se prvo piše ime upravljača, npr. *UserController* i s funkcijom se povezuje znakom @. Na kraju je moguće dodati i *alias* za određenu rutu koji će se koristiti pri svakom sljedećem korištenju rute u ostatku aplikacije, a to se označava s 'as' i nakon toga se piše željeni *alias,* u gornjem primjeru *account* i *account.save.*

Osnovne rute primaju URI (*Uniform Resource Identifier)* i anonimne funkcije koje se u PHP-u nazivaju *closures.* Metode koje je moguće registrirati odgovaraju bilo kojoj HTTP metodi, a to su:

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

Osim ovih, moguće je i registrirati rute koje odgovaraju većem broju HTTP metoda. To je moguće napraviti koristeći metodu **match** ili **any**, rutu koja odgovara svim HTTP metodama.

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

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

Vrste tokova usmjeravanja koje su dostupne su rute preusmjeravanja i rute prikaza. Rutom preusmjeravanja (engl. *redirect routes*) moguće je preusmjeriti jedan URI na drugi, a to se može napraviti koristeći **Route::redirect** npr. **Route::redirect('/ovdje', '/tamo').** U slučaju da je potrebno samo vratiti neki prikaz, može se koristiti **Route::view** metoda i kao i ranija metoda ne zahtijeva dodatne informacije kao što je puna ruta ili upravljač koji će koristiti. Ruta kao prvi argument prima URI i naziv prikaza kao drugi. Piše se u obliku: **Route::view('/welcome', 'welcome');**

Parametri koje rute primaju mogu biti nužni ili neobvezni. Korisnik može definirati nužne parametre ako želi dohvatiti neke informacije iz URL-, što je u sljedećem primjeru na slici 18 ID posta koji će biti obrisan.

```
Route::get( uri: '/deletepost/{post_id}', [
    'uses' => 'PostController@getDeletePost',
    'as' =>'post.delete',
```

Slika 18. Parametri rute

Parametri se uvijek pišu unutar zatvorenih vitičastih zagrada {}, a trebaju se sastojati od abecednih znakova te ne smiju sadržavati znak minus -, ali smiju donju crticu _. Ne postoji pravilo za broj parametara koji će se definirati, npr. kod je mogao sadržavati i ID korisnika čiji je post obrisan. U slučaju definiranja neobveznih parametara, potrebno je samo dodati znak upitnika nakon naziva samog parametra te je potrebno dodati zadanu vrijednost odgovarajućeg parametra. Primjer takvog koda je:

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

Postoji mogućnost kreiranja grupe ruta što omogućava dodjeljivanje atributa kao što je *middleware* ili *namespace* velikom broju ruta bez potrebe da se ti atributi definiraju na svakoj od tih ruta. Kako bi se to definiralo, potrebno je koristiti metodu **`Route::group`**.
Na slici 19 vidljivo je kako je grupa postavljena u ovom projektu.

```
Route::group(['middleware' => 'auth'], function () {
    Route::get( uri: '/dashboard', [
        'uses' => 'PostController@getDashboard',
        'as' => 'dashboard'

    ]);

    Route::post( uri: '/createpost', [
        'uses' =>'PostController@postCreatePost',
        'as' =>'post.create'

    ]);

    Route::get( uri: '/deletepost/{post_id}', [
        'uses' => 'PostController@getDeletePost',
        'as' =>'post.delete'

    ]);
```

Slika 19. Grupe ruta

7.5. Baza podataka (Databases, n.d.)

Laravel omogućava jednostavno upravljanje i interakciju s bazama podataka koristeći neobrađene *sql* upite, graditelj upita i Eloquent ORM koji je objašnjen u poglavlju 7.1. Trenutna verzija Laravela (7.x) podržava četiri baze podataka:

- *MySQL 5.6+*
- *PostgreSQL 9.4+*
- *SQLite 3.8.8+*
- *SQL Server 2017+*

7.5.1. Konfiguracija baze podataka

Konfiguracijske datoteke baze podataka nalaze se u *config/database.php* datoteci, gdje je moguće definirati sve veze baze podataka te odrediti zadane veze. Laravelov primjer konfiguracije spreman je za korištenje nakon instalacije Laravel Homestead virtualnog računala te korisnik tu konfiguraciju može mijenjati prema potrebi. Baza podataka za projekt Bebapapa kreirana je pri stvaranju novog projekta. U *.env* datoteci definirani su parametri potrebni za povezivanje s bazom podataka.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=homestead
DB_USERNAME=homestead
DB_PASSWORD=secret
```

Vidljivo je kako je korištena *mysql* baza podataka s nazivom *homestead* i passwordom *secret.* Nakon konfiguracije veze s bazom podataka, moguće je pokretati *sql* upite. U Laravelu je moguće pokretati neobrađene *sql* zapise i pokretati graditelje upita (engl. *query builder*).

Pokretanje neobrađenih *sql* upita omogućeno je koristeći DB fasadu (engl. *facade*). Fasade u Laravelu omogućavaju tzv. statično sučelje za klase koje su pohranjene u spremniku servisa i one omogućuju pristup mnogim Laravelovim značajkama. Fasade omogućavaju korisniku da koristi značajke bez korištenja dugih imena klasa kojima je potrebna ručna konfiguracija, a definirane su u *Illuminate\Support\Facades* prostoru i mogu se pozvati iz bilo kojeg dijela aplikacije. DB fasada poziva se sljedećom naredbom:

```
Use Illuminate\Support\Facades\DB
```

Nakon pozivanja fasade omogućene su različite naredbe DB fasade, a to su: *select*, *update*, *insert*, *delete* i *statement*. Primjer select naredbe unutar upravljača, gdje je prvi argument proslijeđen *select* naredbi je neobrađeni (sirovi) *sql* upit, dok je drugi parametar bilo koji parametar koji treba vezati za upit, a tu se obično nalaze vrijednosti uvjeta *where,* te izgleda ovako:

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

Sljedeći primjer, primjer je iz projekta Bebapapa. Naredba *insert* služi za umetanje u bazu podataka te isto kao i *select* metoda prima neobrađeni *sql* upit i parametar koji treba vezati za upit. Naredba na slici 20 umeće u stupac ID vrijednost 1 i u stupac *name* vrijednost 'vale'.

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

Slika 20. Insert naredba DB fasade

Naredbom *update* ažuriraju se postojeći zapisi u bazi podataka te vraća broj redaka na koje upit utječe. Primjer za takvu naredbu je:

```
$updated=DB::update('update users set likes = 100 where name = ?',
['user1']);
```

Naredba *delete* koristi se za brisanje zapisa iz baze podataka, a vraća broj promijenjenih redaka. Naredba *delete* piše se ovako:

```
$deleted=DB::delete('delete from posts')
```

Naredba *statement* ne vraća nikakvu vrijednost, a piše se ovako:

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

Graditelj upita drugi je način pokretanja upita za baze podataka. Predstavlja tečno sučelje namijenjeno interakciji s nekoliko različitih tipova baze podataka s jednim jasnim API-jem (Stauffer, 2019, str. 105). U radu su prikazane neke od osnovnih metoda graditelja upita.

Pomoću graditelja upita moguće je dohvatiti sve retke iz tablice u ovom obliku:

```
$users = DB::table('users')->get();
```

Koristi se table() metoda DB fasade koja dohvaća sve retke iz tablice users naredbom get(). *Get* naredba vraća rezultate u obliku instance PHP *stdClass* objekta te je moguće dohvatiti vrijednost svakog stupca tako da se dohvati stupac kao svojstvo objekta.

```
foreach ($users as $user) {
echo $user->name;
}
```

Također je moguće dohvatiti samo jedan redak ili samo jedan stupac iz tablice pomoću ovih metoda i to tako da se postave dodatni uvjeti na metodu table. Takav upit izgledat će ovako:

```
$user = DB::table('users')->where('name', 'John')->first();
echo $user->name;
```

Moguće je koristiti i neke kompleksnije metode koje kombiniraju više uvjeta kao što su –

orWhere(),

whereBetween(colName, [low, high]) – prima ime stupca koja je između dvije vrijednosti

whereBewhereIn(colName, [1, 2, 3] – prima ime stupca i listu opcija

whereNotIn(),

whereIn(colName, [1, 2, 3] – vraća samo retke gdje je vrijednost stupca u traženoj listi opcija,

whereNull(colName) i whereNotNull(colName) – omogućavaju odabir redaka gdje je vrijednost traženog stupca *NULL* ili nije *NULL*,

whereRaw() – omogućava prosljeđivanje sirovog stringa *WHERE* uvjetu npr.:

```
$data = DB::table('users')->whereRaw('id = 1234')->get()
```

whereExists() – omogućava odabir redaka koji će vratiti barem jedan redak pri prosljeđivanju u podupit, npr. ako je potrebno dodati samo korisnike koji su objavili barem jedan post:

```
$posted = DB::table('users)
 ->whereExists(function ($query) {
       $query->select('id')
            ->from('posts')
            ->whereRaw('posts.user_id = user.id');
})
->get();
```

distinct() – omogućava odabir samo onih redaka gdje su podaci jedinstveni u usporedbi s ostalim redcima i obično se koristi uz *select()* naredbu:

```
$names= DB::table('users')->select('city')->distinct()->get();
```

7.5.2. Migracije

Usko povezane s modelima su migracije koje mogu biti kreirane prilikom kreiranja samog modela tako da se naredbi za kreiranje modela doda -m:

```
php artisan make:model Test –m
```

Migracije definiraju željene izmjene kada se pokrene *up* migracija ili *down* migracija. Pri stvaranju migracije pokreće se **`up()`** metoda koja kreira migraciju, ali sustav dopušta i vraćanje na staru verziju migracije, tj. na prošlo stanje migracije i to metodom **`down()`** (Stauffer, 2019, str. 97). Na slici 21 prikazana je migracija *create_users_table* kreirana u Laravel projektu Bebapapa na kojem se radilo pri izradi ovog rada.

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

class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name')->unique();
            $table->string('email')->unique();
            $table->string('password');
            $table->string('gender')->nullable();
            $table->string('age');
            $table->rememberToken();
            $table->timestamps();

        });
    }

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

Slika 21. Kod migracije create_users_table

U kodu je vidljivo kako prema zadanim postavkama unutar migracije postoji **`up()`** metoda koja definira kreiranje nove tablice s određenim poljima i **`down()`** metoda koja definira brisanje tablice. (Stauffer, 2019, str. 90). Također, varijablom $table definiran je tip podatka i naziv svakog stupca u tablici koja se kreira. Laravel pri stvaranju migracije kreira prvi stupac koji je u ulozi primarnog ključa. U retku **`$table->increments('id');`** definira se automatsko povećavanje vrijednosti za 1 za stupac *user_id* u slučaju da mu se ne dodijeli vrijednost. Varijabla **`$table->timestamps();`** kreirat će dva nova stupca u tablici, a to su *created_at i updated_at* i oba sadrže vremensku oznaku kada je zapis stvoren ili kada je izvršena promjena na nekom retku tablice.

Migracija se pokreće novom naredbom, a to je **`php artisan migrate`** i svaki put pri pozivanju ove naredbe Laravel će provjeriti koje migracije su već pozvale metodu **`up()`**, a koje ne i pozvati ih ovisno o tome. Ovu je naredbu potrebno pozvati nakon svakog definiranja parametara migracije (Stauffer, 2019, str. 97).

7.5.3. Popunjavanje baze podataka testnim podacima (engl. *seeding*)

Laravel omogućuje jednostavnu metodu popunjavanja baze podataka testnim podacima koja se naziva *seeding*, a za to koristi *Seeder* klasu. Klasa se nalazi unutar direktorija *database/seeds*. Prema konvenciji se nazivaju prema nazivu tablice, npr. u projektu je korišten *UsersTableSeeder* koji služi za popunjavanje *users* tablice testnim podacima, a prema zadanim postavkama Laravel sadrži *DatabaseSeeder* koji kontrolira redoslijed popunjavanja baze.

Za kreiranje *Seeder-a* koji će biti spremljen u *database/seeds* koristi se *Artisan* naredba:

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

Nakon kreiranja vidljivo je kako Seeder koristi samo metodu *run()* koja će biti pokrenuta izvršavanjem naredbe **`php artisan db:seed.`** Dio kojim korisnik definira koje će podatke unijeti u bazu podataka unosi se unutar *run()* metode. U primjeru iz projekta *Bebapapa* vidljivo je kako je definirano popunjavanje tablice *users* sa svim podacima o korisniku koristeći naredbu *insert(),* prikazano na slici 22.

```php
class UsersTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        DB::table('users')->insert([
            'name'=>'vale',
            'email' => 'user4@gmail.com',
            'password'=>bcrypt( value: 'password'),
            'gender'=>'musko',
            'age'=>'six',
        ]);
```

Slika 22. UsersTableSeeder primjer

Da bi se omogućilo pokretanje te naredbe, potrebno je istu definirati u *Database Seeder* datoteci kao na slici 23, a ta opcija omogućava i pozivanje više klasa, npr. moguće je dodati još *PostTableSeeder::class* i sl.:

```php
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     *
     * @return void
     */
    public function run()
    {
        $this->call([
            UsersTableSeeder::class,

        ]);
    }
```

Slika 23. Database Seeder

Osim ručnog kreiranja podataka pomoću *Seedera* moguće je koristiti i tvornice (engl. *factories*). Razlikuju se od stvaranja modela po tome što generiraju veliku količinu podataka u bazu podataka. Korisnik unutar tvornice može definirati broj zapisa koji će kreirati npr. 50 korisnika. U primjeru je vidljiva naredba za kreiranje 50 korisnika te se svakom korisniku definira relacija:

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

Nakon što je korisnik kreirao *Seeder* potrebno ga je pokrenuti. Prije svega naredbom *dump-autoload* poziva se *Composerovo* automatsko učitavanje:

```
composer dump-autoload
```

Nakon toga poziva se Artisan naredba *db:seed* koja pokreće *Database Seeder*, a ako je definirano više klasa, a potrebno je pozvati samo jednu, može se pozvati i samo jedna klasa.

```
php artisan db:seed,
```
ili

```
php artisan db:seed –class=UserSeeder
```

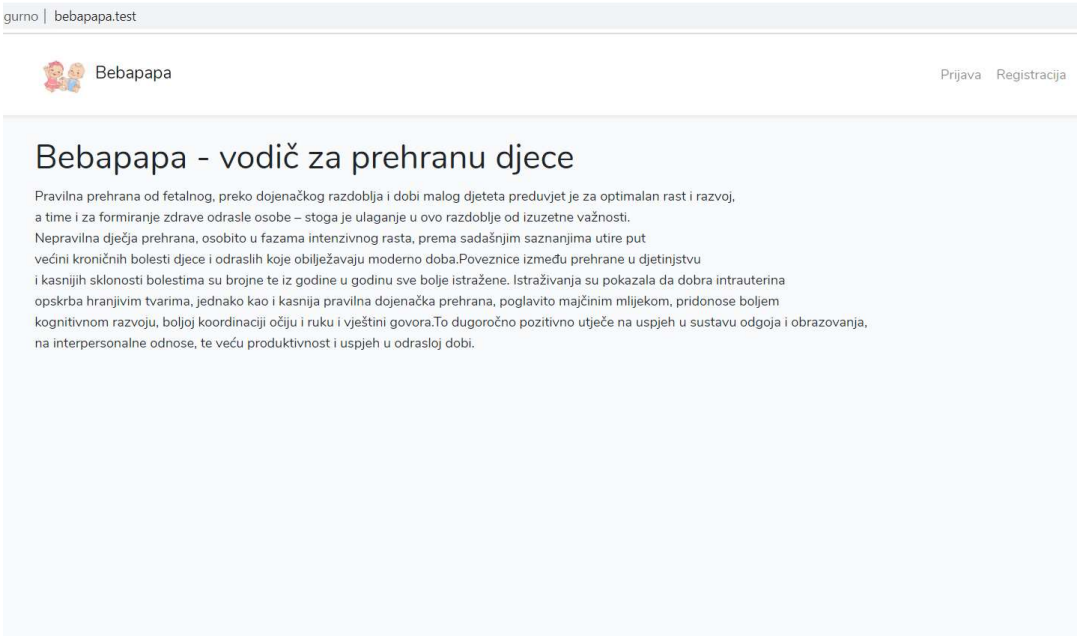
8. Izrada Laravel projekta – primjena

U prethodnim poglavljima objašnjen je postupak izrade web-aplikacije unutar Laravel radnog okvira te sve njegove mogućnosti, ali i prednosti i nedostaci te su vidljivi dijelovi koda razvijene web-aplikacije. U ovom će poglavlju detaljnije biti prikazana izrada projekta na konkretnom primjeru te njegove mogućnosti.

Pri izradi Laravel web-aplikacije korišteni su sljedeći programi:

- *Oracle VM Box* za pokretanje *Laravel Homestead* programa i *Vagranta*
- *Composer*
- *HeidiSQL* (za manipulaciju s bazom podataka)

Ova aplikacija, *Bebapapa,* zamišljena je kao svojevrsna društvena mreža koja pruža informacije o prehrani djece do godine dana, tj. o uvođenju dohrane kod djece, ali i na kojoj korisnici mogu dijeliti svoje objave, mijenjati ih ili komentirati. Omogućeno je i praćenje drugih korisnika te se korisnicima nudi preporuka sadržaja ovisno o dobi djeteta koju navode pri registraciji na društvenu mrežu. Naslovnica aplikacije vidljiva je na slici 24.



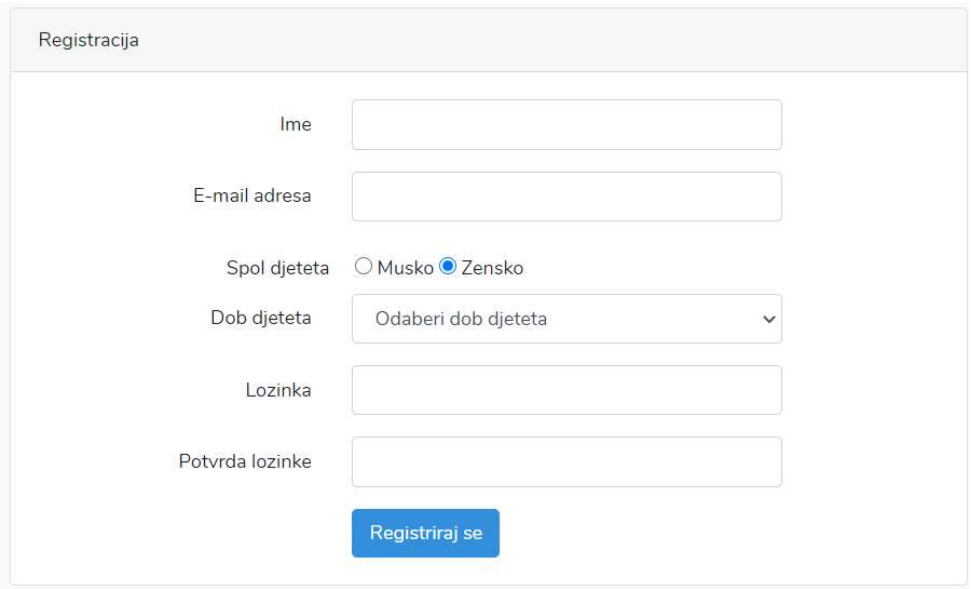
Slika 24. Naslovnica aplikacije

8.1. Autentikacija korisnika

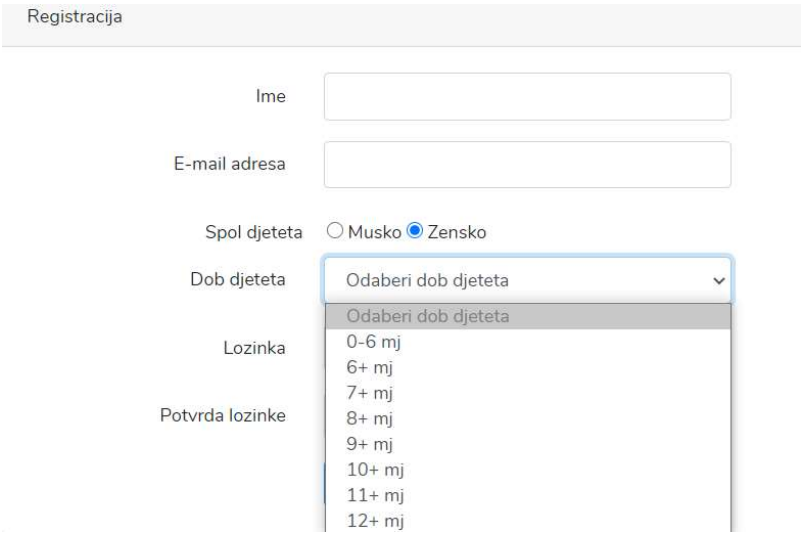
Laravel programerima znatno olakšava proces implementacije autentikacije, tj. gotovo sve opcije su unaprijed konfigurirane i spremljene u konfiguracijsku datoteku *config/auth.php.* Kako bi se implementirala opcija autentikacije, prije svega je potrebno instalirati *Composer* paket *laravel/ui* koji nudi izdvojene dijelove korisničkog sučelja (engl. *user interface*). Zatim je potrebno pokrenuti naredbu **`php artisan ui vue –auth`**. Naredba kreira potrebne tokove usmjeravanja i generira *HomeController* u kojem su definirane metode koje se pokreću nakon prijave korisnika. U *Controllers* direktoriju generirat će još nekoliko upravljača koji upravljaju registracijom novog korisnika (*RegisterController*), autentikacijom (*LoginController*), slanjem poveznica za resetiranje lozinke (*ForgotPasswordController*) i upravljač koji sadrži logiku za resetiranje lozinke (*ResetPasswordController*). *Laravel/ui* paket kreira i sve potrebne prikaze te ih sprema u *resources/views/auth* direktorij. Ovom naredbom se kreiraju i glavni predlošci za aplikaciju koji koriste Bootstrap CSS radni okvir, ali je svakog od njih moguće prilagoditi.

8.1.1. Registracija

Pri izradi *Bebapapa* aplikacije registracijska forma je izmijenjena kako bi se dodale neke nove informacije o korisniku važne za rad aplikacije. Tako je osim generiranih polja dodano polje za spol i dob djeteta korisnika koje se registrira, vidljivo na slici 25.



Slika 25. Forma za registraciju novih korisnika



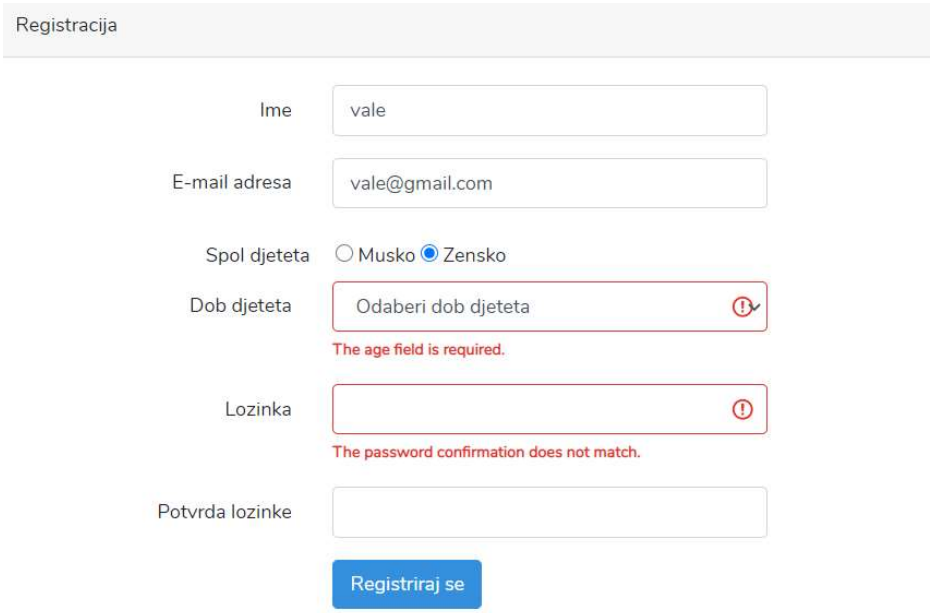
Slika 26. Dodatna polja za registraciju korisnika

Za odabir spola djeteta u prikaz za registraciju je dodan *radio button*, a za dob djeteta padajući izbornik koji omogućava aplikaciji preporuku sadržaja na temelju dobi djeteta (Slika 26). Unutar *RegisterController* datoteke unesena su dodatna polja *gender* i *age,* koja definiraju da su ta polja pri kreiranju korisnika obvezna te da vraćaju podatke koje korisnik unese u ta polja unutar funkcije za validaciju korisnika i za kreiranje korisnika (slika 27).

```php
protected function validator(array $data)
{
    return Validator::make($data, [
        'name' => ['required', 'string', 'max:255'],
        'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
        'password' => ['required', 'string', 'min:8', 'confirmed'],
        'gender'=> ['required'],
        'age'=>['required']
    ]);
}

/**
 * Create a new user instance after a valid registration.
 *
 * @param  array  $data
 * @return \App\User
 */
protected function create(array $data)
{
    return User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => Hash::make($data['password']),
        'gender'=>$data['gender'],
        'age'=>$data['age']
    ]);
```

Slika 27. RegisterController

U slučaju neispunjavanja nekih polja ili neispravnog ispunjavanja, javit će se greške kao na slici 28.



Slika 28. Neuspješna registracija

Što se događa nakon registracije definirano je u *UserController* datoteci unutar funkcije *postRegister* koja prima podatke prema unosu korisnika i sprema ih kao novog korisnika te nakon spremanja vraća prikaz *home* (slika 29).

```
class UserController extends Controller
{
    public function postRegister(Request $request)
    {
        $user=new User();
        $user->email=$request['email'];
        $user->name=$request['name'];
        $user->password=$request['password'];
        $user->gender=$request['gender'];
        $user->age=$request['age'];
        $user->save();

        return view( view: 'home');

    }
```

Slika 29. postRegister funkcija

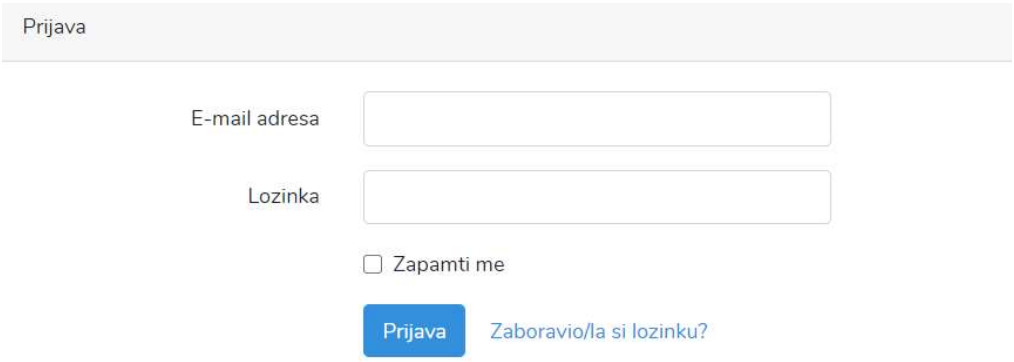
Na slici 30 vidljivo je kako nakon registracije upravljač usmjerava korisnika na *bebapapa.test/home* te javlja informaciju o uspješnoj prijavi.

Slika 30. Uspješna registracija

Slika 30 također prikazuje i kako se nakon registracije pojavio novi izbornik koji nudi opcije *Prehrana, Preporuka, Prati* te se s desne strane pojavljuje korisničko ime prijavljenog korisnika te padajući izbornik s mogućnošću odabira *Računa* ili *Odjave.*

8.1.2. Prijava

Pri prijavi korisnika koristi se generirana forma koja traži unos e-maila korisnika kojim se registrirao te lozinke. Postoji i mogućnost „*Zapamti me*“ i opcija resetiranja lozinke kojoj se pristupa klikom na *„Zaboravio/la si lozinku?“*. Slika 31 prikazuje izgled forme za prijavu, a slika 32 isječak koda iste forme.

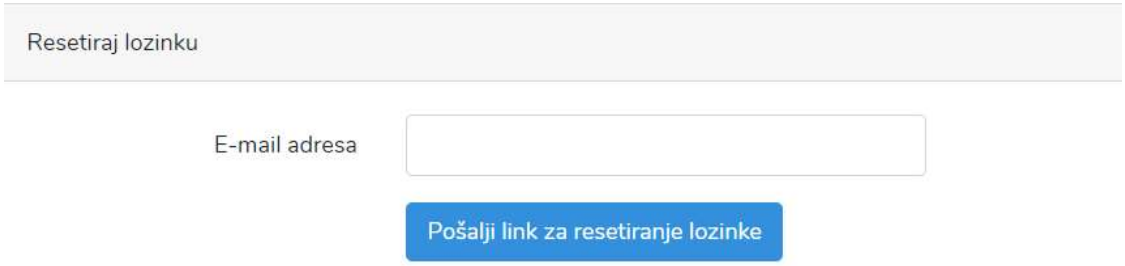


Slika 31. Prijava korisnika

```
<form method="POST" action="{{ route('login') }}">
    @csrf

    <div class="form-group row">
        <label for="email" class="col-md-4 col-form-label text-md-right">
            {{ __('E-mail adresa') }}</label>

        <div class="col-md-6">
            <input id="email" type="email" class="form-control @error('email')is-invalid @enderror"
                   name="email" value="{{ old('email') }}" required autocomplete="email" autofocus>

            @error('email')
                <span class="invalid-feedback" role="alert">
                    <strong>{{ $message }}</strong>
                </span>
            @enderror
        </div>
    </div>

    <div class="form-group row">
        <label for="password" class="col-md-4 col-form-label text-md-right">
            {{ __('Lozinka') }}</label>

        <div class="col-md-6">
            <input id="password" type="password" class="form-control @error('password')
                is-invalid @enderror" name="password" required autocomplete="current-password">

            @error('password')
                <span class="invalid-feedback" role="alert">
                    <strong>{{ $message }}</strong>
                </span>
            @enderror
```

Slika 32. Isječak koda forme za prijavu

Ako korisnik odabere opciju *„Zaboravio/la si lozinku?“*, otvorit će se prikaz koji nudi slanje poveznice za resetiranje lozinke na e-mail adresu, prikaz je vidljiv na slici 33.



Slika 33. Resetiranje lozinke

Nakon uspješne prijave korisnika ponovno se otvara prikaz kao i nakon registracije korisnika te se korisniku prikazuje cijeli izbornik. Da bi se izbornik pojavio nakon prijave, a ne prije nje, koristi se metoda *Auth* koja je dio *laravel/ui* paketa te kontrolira sadržaj koji se može prikazati samo autenticiranim korisnicima ili tokove usmjeravanja koji su omogućeni samo za autenticirane korisnike. Tako je pri glavnom predlošku u kod dodana metoda *Auth* kojom se provjerava je li korisnik autenticiran te mu se prikazuje sadržaj ovisno o tome. Metoda je prikazana na slici 34.

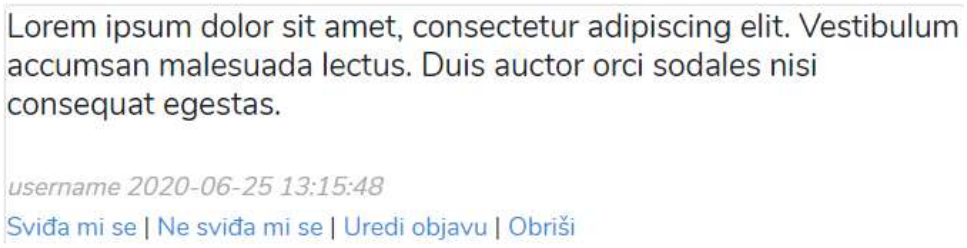
```
<div class="collapse navbar-collapse" id="navbarSupportedContent">
    <!-- Left Side Of Navbar -->
    <ul class="navbar-nav mr-auto">
        @if(Auth::check())
        <div class="navbar-nav">
            <a href="/prehrana" class="nav-item nav-link">Prehrana</a>
        </div>
        <div class="navbar-nav">
        <a href="/preporuka" class="nav-item nav-link">Preporuka</a>
        </div>
        <div class="navbar-nav">
            <a href="/users" class="nav-item nav-link">Prati</a>
        </div>
            @endif
    </ul>
```

Slika 34. Korištenje Auth metode

8.2. Objave korisnika

Nakon prijave korisnika, otvara se prikaz s linkom na naslovnicu koji vodi na *bebapapa.test/dashboard* na kojem je moguće podijeliti objave koje su vidljive svim prijavljenim korisnicima.

Objava korisnika sadrži samu objavu, korisničko ime korisnika koji je objavu podijelio te vrijeme i datum dijeljenja objave. Vidljive su i opcije za dodavanje oznake *„sviđa mi se“* i *„ne sviđa mi se“* te za uređivanje i brisanje objave. Opcija uređivanja i brisanja dostupna je samo za objave koje korisnik kreirao, a za objave drugih korisnika vidljive su samo prve dvije opcije. Slika 35 prikazuje izgled objave korisnika.



Slika 35. Izgled objave korisnika

Pri izradi mogućnosti objave korisnika prvo je izrađen model *Post.php* te je pokrenuta i migracija za kreiranje *posts* tablice unutar baze podataka. Za rad ove funkcije potreban je još i upravljač te je generiran *PostController* koji sadrži sve funkcije potrebne za dijeljenje, uređivanje i brisanje objave.

Na slici 36 vidljiv je kod modela *Post.php* koji uređuje veze s modelima *User, Like* i *Comment.* Definira inverznu relaciju *belongsTo* za *User* model što znači da svaka objava pripada nekom korisniku, relaciju *hasMany* koja definira da jedna objava ima više *„sviđa mi se“* oznaka te da svaka objava ima više komentara.

```php
namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{

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

    public function likes()
    {
        return $this->hasMany( related: 'App\Like');
    }

    public function comments()
    {
        return $this->hasMany( related: Comment::class)->whereNull( columns: 'parent_id');
    }
```

Slika 36. Post.php model

Kreiranje objave, ali i njen prikaz definirani su u upravljaču *PostController.* Na slici 37 vidljiv je isječak koda *PostController* datoteke.

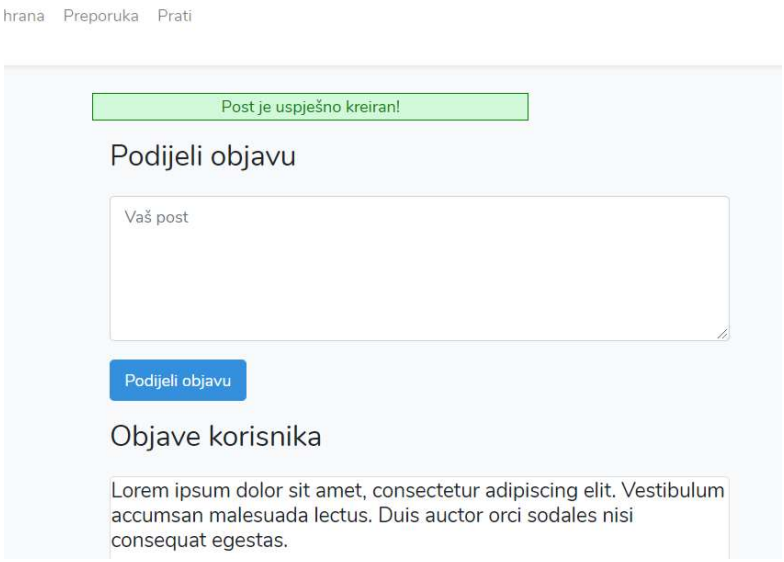
```php
<?php
namespace App\Http\Controllers;

use App\Like;
use App\Post;
use App\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class PostController extends Controller
{
    public function getDashboard()
    {
        $posts = Post::orderBy('created_at', 'desc')->get();
        return view( view: 'dashboard', ['posts' => $posts]);
    }

    public function postCreatePost(Request $request)
    {
        $this->validate($request, [
            'body' => 'required|max:1000'
        ]);
        $post = new Post();
        $post->body = $request['body'];
        $message = 'Dogodila se pogreška';
        if ($request->user()->posts()->save($post)) {
            $message = 'Post je uspješno kreiran!';
        }
        return redirect()->route( route: 'dashboard')->with(['message' => $message]);
    }
```

Slika 37. Isječak koda PostController datoteke

U kodu je vidljivo kako su definirane funkcije *getDashboard()* koja u varijablu *$posts* sprema postove poredane po datumu stvaranja, silaznim putem, i vraća ih u prikazu *dashboard.*

Funkcija *postCreatePost* prije svega Laravelovom uključenom funkcijom za validaciju formi provjerava odgovara li unesen tekst duljini definiranoj unutar te funkcije. Nakon toga se u varijablu *$post* sprema novi post koji sadrži tekst unesen od strane korisnika. U slučaju da unesen sadržaj ne odgovara zahtjevima, javit će se greška, a ako je sve u redu, post će biti

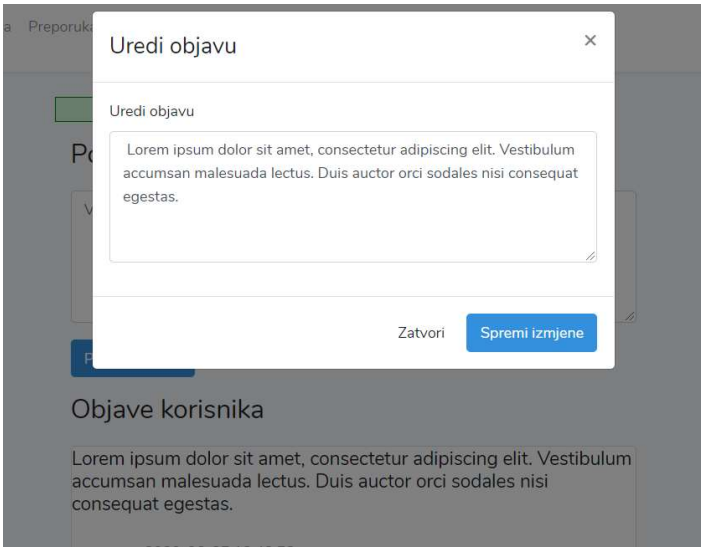
kreiran i spremljen te će se ponovno učitati stranica s prikazom objave i porukom da je objava uspješno kreirana što je vidljivo na slici 38.



Slika 38. Dijeljenje objave

8.2.1. Izmjena objave

Osim mogućnosti dijeljenja, moguće je i izmijeniti podijeljenu objavu. Dijeljenju objave pristupa se klikom na *Uredi objavu* kada se otvara skočni prozor s objavom i mogućnošću izmjene kao što je vidljivo na slici 39.



Slika 39. Uređivanje objave

Forma za otvaranje skočnog prozora definirana je prikazu *dashboard.blade.php.* i na slici 40 je vidljiv dio koda za formu.

```
<div class="modal fade" tabindex="-1" role="dialog" id="edit-modal">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h4 class="modal-title">Uredi objavu</h4>
                <button type="button" class="close" data-dismiss="modal" aria-label="Close"><span aria-hidden="t
            </div>
            <div class="modal-body">
                <form>
                    <div class="form-group">
                        <label for="post-body">Uredi objavu</label>
                        <textarea class="form-control" name="post-body" id="post-body" rows="5"></textarea>
                    </div>
                </form>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-default" data-dismiss="modal">Zatvori</button>
                <button type="button" class="btn btn-primary" id="save-modal">Spremi izmjene</button></div>
            </div><!-- /.modal-content -->
        </div><!-- /.modal-dialog -->
    </div><!-- /.modal -->


    <script type="application/javascript">
        var token = '{{ Session::token() }}';
        var urlEdit = '{{ route('edit') }}';
        var urlLike = '{{ route('like') }}';
    </script>
```

Slika 40. Kod skočnog prozora

U kodu je oblikovan izgled forme te je definiran URL koji vodi do upravljača *PostController* u kojem je definirana funkcija za kreiranje posta. Razlog zbog kojeg je URL definiran i unutar prikaza je taj što je za otvaranje skočnog prozora korišten *javascript* i *ajax* (metoda koja služi za automatsko ažuriranje stranice, tj. komunikaciju sa serverom, bez potrebe za ponovnim učitavanjem stranice) te je taj URL potreban za povezivanje *javascript* koda s prikazom i upravljačem. U upravljaču je definirana funkcija za izmjenu objave, a kod je vidljiv na slici 41.

```php
public function postEditPost(Request $request)
{
    $this->validate($request, [
        'body' => 'required'
    ]);
    $post = Post::find($request['postId']);
    if(Auth::user() != $post->user){
        return redirect()->back();
    }
    $post->body = $request['body'];
    $post->update();
    return response()->json(['new_body' => $post->body], status: 200);
}
```

Slika 41. Kod za izmjenu objave

U kodu za izmjenu objave definirano je da se objava pronađe prema ID-u i da mora biti objava korisnika koji je prijavljen u aplikaciju. Nakon toga se dohvaća tekst objave koji se nakon izmjene sprema u izmijenjenom obliku.

Slika 42 prikazuje kako se unutar baze podataka postovi spremaju pod određenim ID-em s kojim su povezani i s korisnikom, a sprema se cijeli tekst objave.

homestead.posts: 8 rows total (approximately)

id	created_at	updated_at	body	user_id
1	2020-06-22 16:46:36	2020-06-22 16:46:36	Ovo je moj prvi post.	2
2	2020-06-23 04:47:14	2020-06-23 04:47:14	A ovo je moj post	11
3	2020-06-24 09:57:39	2020-06-24 09:57:39	Dijelim Post	2
5	2020-06-24 12:01:45	2020-06-24 12:01:45	Zasto ne	7
6	2020-06-24 12:15:50	2020-06-24 12:15:50	A ovaj post	7
7	2020-06-25 03:39:57	2020-06-25 03:39:57	Moj novi post	10
8	2020-06-25 13:02:08	2020-06-25 13:05:16	Ovo je objava korisnika username.	12
10	2020-06-25 13:42:58	2020-06-25 13:42:58	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum ac...	12

Slika 42. Tablica posts u bazi podataka

8.2.2. Brisanje objave

U aplikaciji je omogućeno i brisanje vlastitih objava. Unutar *PostController* datoteke definirana je funkcija *getDeletePost()* (slika 43) koja se poziva klikom na opciju *Obriši* ispod objave korisnika (Slika 44). Funkcija povlači post prema ID-u i briše ga te se ta funkcija omogućava samo prijavljenom korisniku, a vraća poruku o uspješnom brisanju.

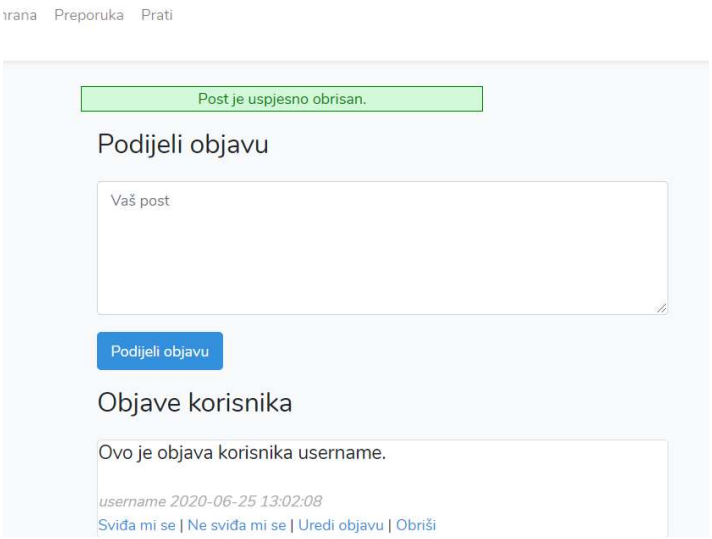
```
public function getDeletePost($post_id)
{
    $post=Post::where('id', $post_id)->first();
    if(Auth::user() != $post->user){
        return redirect()->back();
    }
    $post->delete();
    return redirect()->route( route: 'dashboard')->with(['message' => 'Post je uspjesno obrisan.']);
}
```

Slika 43. *getDeletepost* funkcija

```
<section class="row posts">
    <div class="col-md-6 offset-md-3">
        <header><h3>Objave korisnika</h3></header>
        @foreach($posts as $post)
            <article class="post card" data-postid="{{$post->id}}">
                <h5><p> {{ $post->body }}</p></h5>
                <div class="info">
                    {{$post->user->name}} {{ $post->created_at }}
                </div>
                <div class="interaction">
                    <a href="#" class="like">{{ Auth::user()->likes()->where('post_id', $post->id)->f
                    |
                    <a href="#" class="like">{{ Auth::user()->likes()->where('post_id', $post->id)->f
                    @if(Auth::user() == $post->user)
                        |
                        <a href="#" class="edit">Uredi objavu</a>
                    |
                        <a href="{{ route('post.delete', ['post_id' => $post->id]) }}">Obriši</a>
                    @endif
                    @if(Auth::user() !=$post->user)
                        <div class="form-group">
                            <textarea class="form-control" name="comment-body" id="comment-body" rows
                            <input type="submit" class="btn btn-primary" value="Komentiraj"/>
                        </div>
```

Slika 44. */dashboard* prikaz

Nakon brisanja javit će se poruka o uspješnom brisanju objave (slika 45), te će objava *„Lorem ipsum...“* biti izbrisana iz baze podataka (slika 46).



Slika 45. Brisanje objave

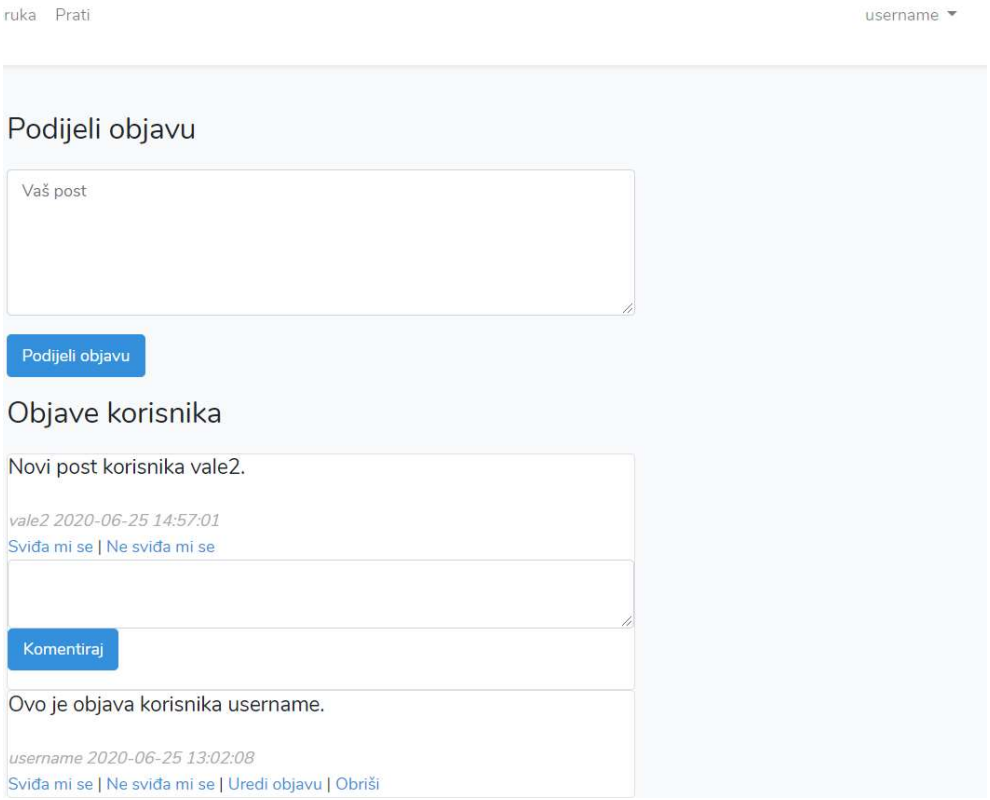
ıomestead.posts: 7 rows total (approximately)

id	created_at	updated_at	body	user_id
1	2020-06-22 16:46:36	2020-06-22 16:46:36	Ovo je moj prvi post.	2
2	2020-06-23 04:47:14	2020-06-23 04:47:14	A ovo je moj post	11
3	2020-06-24 09:57:39	2020-06-24 09:57:39	Dijelim Post	2
5	2020-06-24 12:01:45	2020-06-24 12:01:45	Zasto ne	7
6	2020-06-24 12:15:50	2020-06-24 12:15:50	A ovaj post	7
7	2020-06-25 03:39:57	2020-06-25 03:39:57	Moj novi post	10
8	2020-06-25 13:02:08	2020-06-25 13:05:16	Ovo je objava korisnika username.	12

Slika 46. Baza podataka nakon brisanja objave

8.2.3. „Sviđa mi se/Ne sviđa mi se“ oznaka

Ove oznake omogućene su za sve objave; i za one prijavljenog korisnika i za objave drugih korisnika, prikazano na slici 47.



Slika 47. Pregled objava

Za ovu je opciju kreiran novi model *Like.php* (Slika 48) u kojem su definirane relacije prema *User* i *Post* modelu, a prema kojem *like* pripada (*belongsTo)* određenom korisniku i objavi. U *User* i *Post* modelima definirana je relacija *hasMany*, koja označava da i korisnik i objava mogu imati više oznaka *„Sviđa mi se“*.

```php
namespace App;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Support\Facades\Auth;

class Like extends Model
{
    public function user()
    {
        return $this->belongsTo( related: 'App\User');
    }

    public function post()
    {
        return $this->belongsTo( related: 'App\Post');
    }
}
```

Slika 48. *Like.php* datoteka

U datoteci *PostController* definirana je funkcija *postLikePost()* (slika 49) kojom se definiraju akcije koje se pozivaju pritiskom na *„Sviđa mi se/Ne sviđa mi se“* oznaku. Funkcija definira pozivanje objave prema ID-u odabirom jedne od ponuđenih opcija i dostupnost opcije samo prijavljenim korisnicima. Također, definira nekoliko scenarija; u slučaju da je već odabrana opcija *„Sviđa mi se/Ne sviđa mi se“* i ponovno se odabere, ista će se obrisati, a ako ne postoji, u bazu podataka će se spremati nova oznaka. Oznaka se veže za objavu i za korisnika u bazi podataka pomoću ID-eva.

```php
public function postLikePost(Request $request)
{
    $post_id = $request['postId'];
    $is_like= $request['isLike'] === 'true';
    $update = false;
    $post = Post::find($post_id);
    if (!$post) {
        return null;
    }
    $user = Auth::user();
    $like = $user->likes()->where('post_id', $post_id)->first();
    if ($like) {
        $already_like = $like->like;
        $update = true;
        if ($already_like == $is_like) {
            $like->delete();
            return null;
        }
    } else {
        $like = new Like();
    }
    $like->like = $is_like;
    $like->user_id = $user->id;
    $like->post_id = $post->id;
    if ($update) {
        $like->update();
    } else {
        $like->save();
    }
    return null;
```

Slika 49. postLikePost funkcija

U prikazu je definirano što će se događati prilikom klika na određenu opciju (slika 50).

```php
<div class="interaction">
    <a href="#" class="like">{{ Auth::user()->likes()->where('post_id', $post->id)->first() ?
    Auth::user()->likes()->where('post_id', $post->id)->first()->like == 1 ?
    'Sviđa ti se ova objava.' : 'Sviđa mi se' : 'Sviđa mi se'}}</a>
    |
    <a href="#" class="like">{{ Auth::user()->likes()->where('post_id', $post->id)->first()
    ? Auth::user()->likes()->where('post_id', $post->id)->first()->like == 0 ?
     'Ne sviđa ti se ova objava.' : 'Ne sviđa mi se' : 'Ne sviđa mi se'}}</a>
```

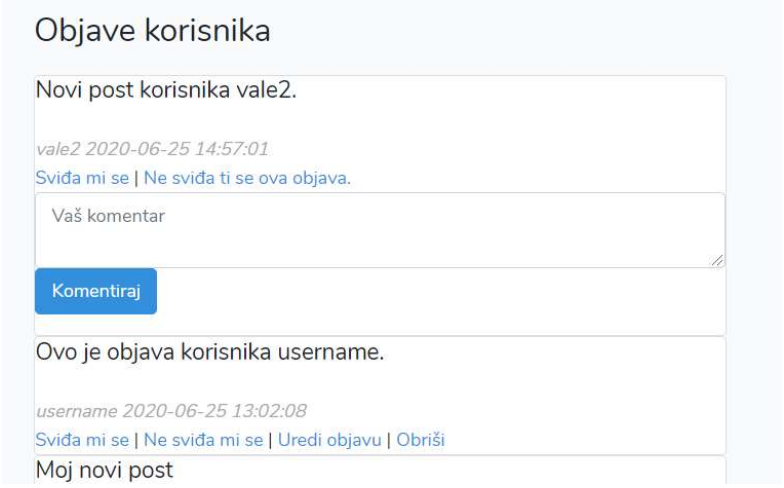
Slika 50. Opcija unutar prikaza

Definirano je da se, u slučaju da je vrijednost varijable *$isLike=1* (iz datoteke *PostController*), pojavi tekst kako se korisniku sviđa objava, a u slučaju ponovnog odabira opcije vratit će se tekst *„Sviđa mi se*“, tj. vrijednost će se vratiti na početnu. U slučaju da se odabere opcija *„Ne sviđa mi se“*, pojavit će se tekst *„Ne sviđa ti se ova objava“*, ali pritiskom na opciju *„Sviđa mi se“* automatski će se maknuti oznaka *„Ne sviđa mi se“*. Prikazano na slici 51. Ovaj automatski prijelaz prilagođen je unutar *.js* datoteke pomoću *jQuery.ajax* metode koji je korišten za prikaz pojedinih dijelova aplikacije (*frontend).*

Slika 51. Odabir opcije "Sviđa mi se/Ne sviđa mi se"

8.2.4. Dodavanje komentara

U aplikaciji je omogućeno dodavanje komentara kod svake objave nekog drugog korisnika. Na slici 52 prijavljen je korisnik *username* te za njegovu objavu ne postoji opcija komentiraj, ali je za objavu korisnika *vale2* opcija omogućena.



Slika 52. Prikaz opcije za komentiranje

Za dodavanje ove opcije kreirana je migracija *create_comments_table* te su postavljena polja koja u tablicu dodaju *body* komentara, i povezuju komentar s *user_id* i *post_id* poljima u tablicama korisnika i objava. Migracija je vidljiva na slici 53.

```php
class CreateCommentsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create( table: 'comments', function (Blueprint $table) {
            $table->increments( column: 'id');
            $table->text( column: 'body');
            $table->integer( column: 'user_id')->onDelete('cascade');
            $table->integer( column: 'post_id')->onDelete('cascade');
            $table->timestamps();
        });
    }
```

Slika 53. *create_comments_table* migracija

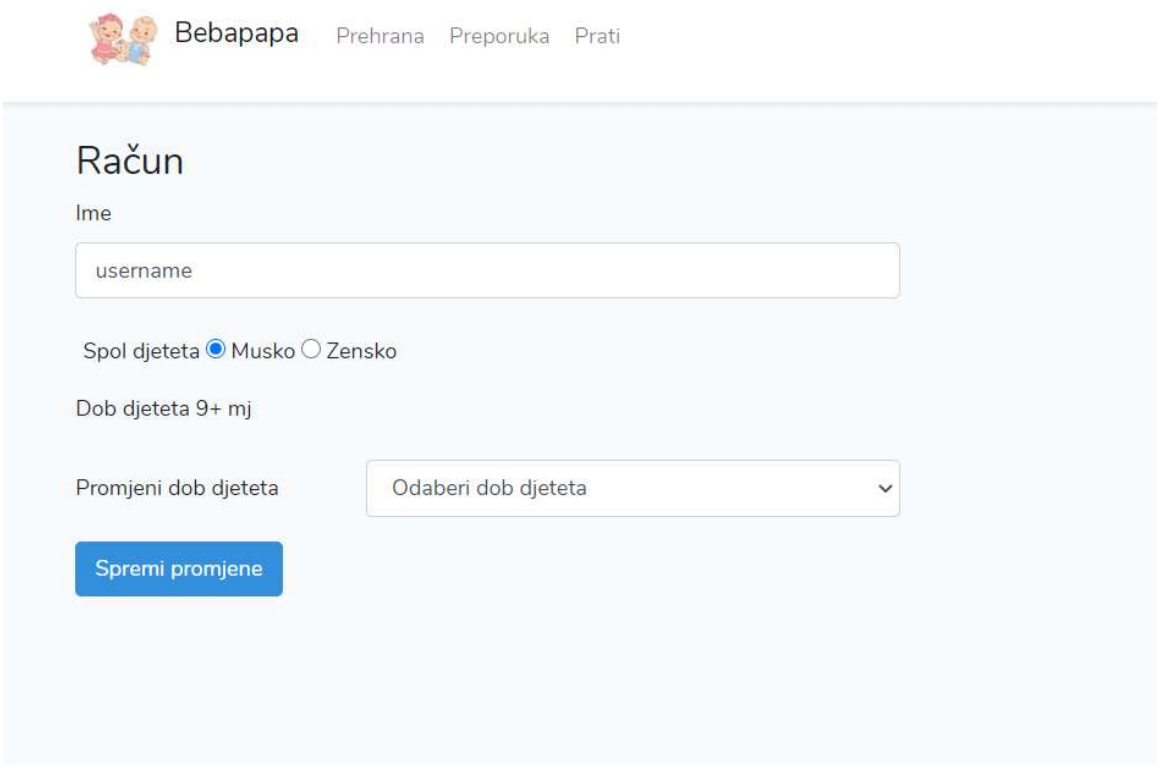
Kreiran je i model *Comment.php* koji definira relacije s modelom *User* i s modelom *Post.* U *CommentController* datoteci kreirana je funkcija *postCreateComment* koja povezuje komentar s ID-em korisnika i objave i prikazuje ga uz vezanu objavu (slika 54).

```php
class CommentController extends Controller
{
    public function __construct() {
        $this->middleware( middleware: 'auth');
    }


    public function postCreateComment(Request $request)
    {
        $this->validate($request, [
            'body' => 'required|max:200'
        ]);
        $post_id = $request['postId'];
        $comment = new Comment();
        $comment->body = $request['body'];
        $post = Post::find($post_id);
        $user=Auth::user();
        $comment = $user->comments()->where('post_id', $post_id)->first();
        if ($request->user()->comments()->save($comment)) {
            $message = 'Post je uspješno kreiran!';
        }
        return redirect()->route( route: 'dashboard')->with(['message' => $message]);
```

Slika 54. CommentController

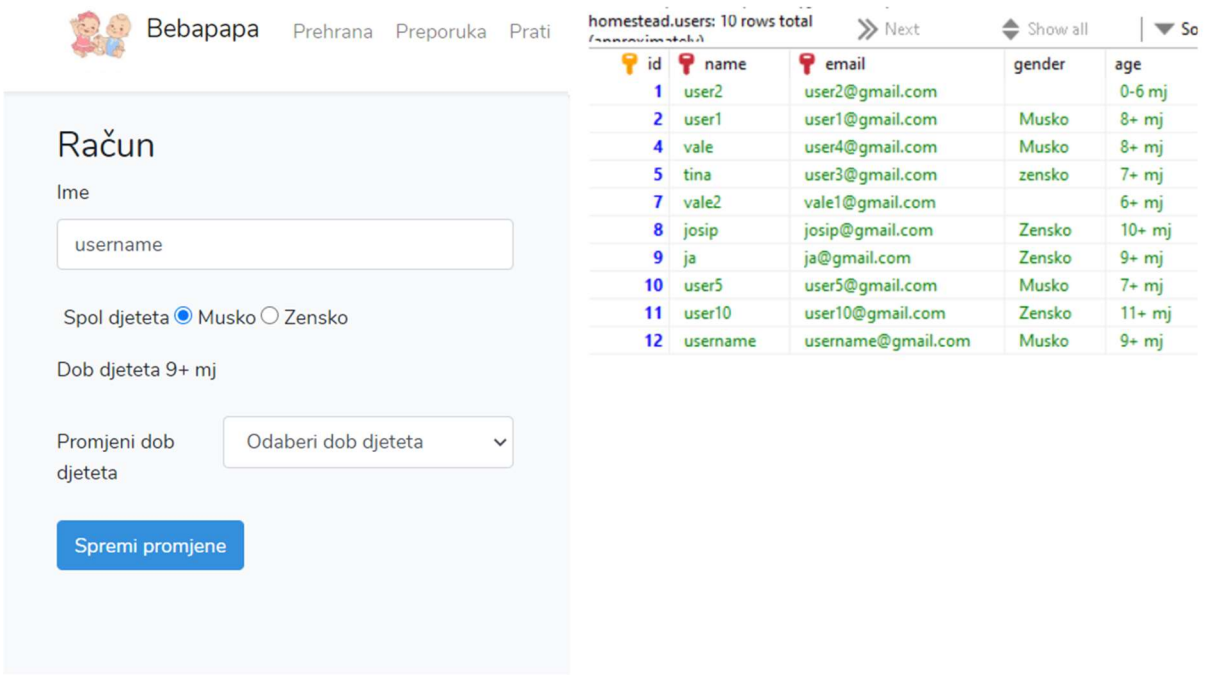
8.3. Uređivanje korisničkog profila

Nakon prijave korisnika u aplikaciju, omogućena mu je opcija promjene podataka u obliku izmjene u računu što je vidljivo na slici 55.

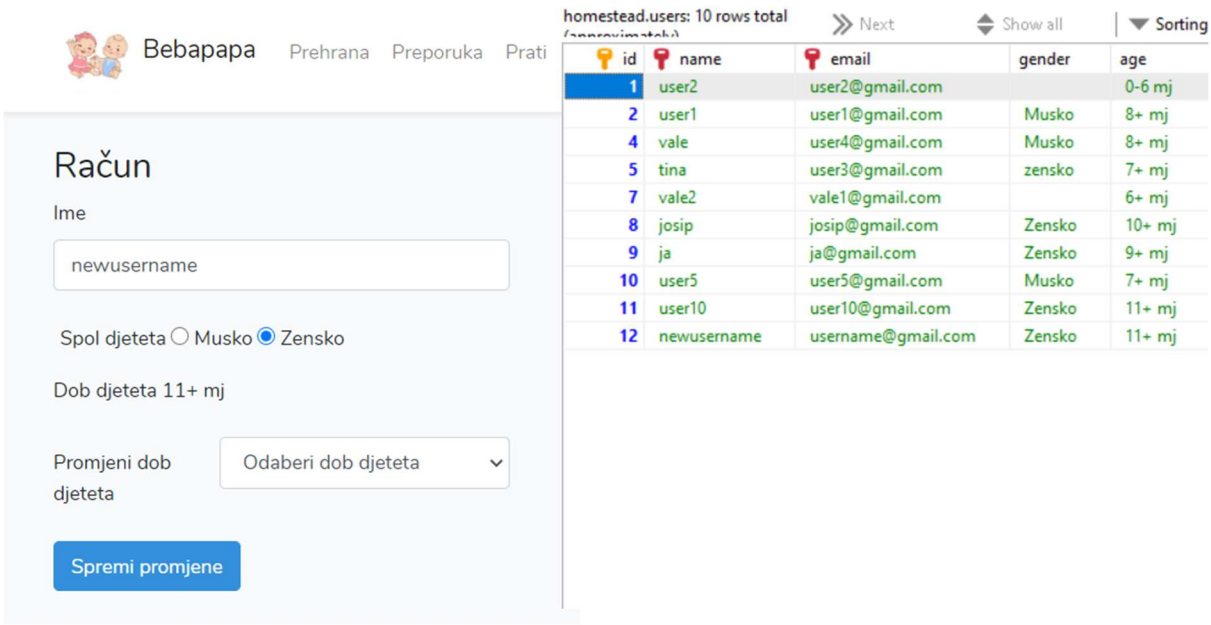


Slika 55. Prikaz računa

Na stranici računa vidljivo je odabrano korisničko ime, spol djeteta te dob djeteta. Nakon dobi omogućen je padajući izbornik za promjenu dobi djeteta te se nakon promjene podataka podaci spremaju u bazu podataka. Sve podatke aplikacija automatski dohvaća iz baze podataka. Na slici 56 vidljiv je račun korisnika *username* i tablica *users* u koju su spremljeni podaci o tom računu pod ID-em 12. Nakon promjene podataka, podaci su automatski izmijenjeni u bazi podataka i vidljivi korisniku (slika 57).



Slika 56. Račun prikaz i baza podataka



Slika 57. Račun i baza podataka nakon promjene

Za kreiranje mogućnosti izmjene računa kreirana je nova datoteka (prikaz) *account.blade.php* u kojoj su definirana pravila za dohvaćanje podataka iz baze podataka. Na slici 58 vidljiv je isječak koda ovog prikaza.

```
@section('content')
    <section class="row new-post">
        <div class="col-md-6 col-md-offset-3">
            <header><h3>Račun</h3></header>
            <form action="{{ route('account.save') }}" method="post" enctype="multipart/form-data">
                <div class="form-group">
                    <label for="name">Ime</label>
                    <input type="text" name="name" class="form-control" value="{{ $user->name }}" id="name">
                </div>
                <div class="form-check row">
                    <label for="gender" class="col-form-label">Spol djeteta</label>
                    <input type="radio" name="gender" id=male value="Musko"
                    {{$user->gender == 'Musko' ? 'checked' : ''}}>
                    <label for="gender">Musko</label>
                    <input type="radio" name="gender" id="female" value="Zensko"
                           {{$user->gender == 'Zensko' ? 'checked' : ''}}>
                    <label for="gender">Zensko</label>
                </div>

                <div class="form-group">
                    <label for="age" class="col-form-label">Dob djeteta {{ Auth::user()->age }} </label>
                </div>

                <div class="form-group row">
                    <label for="" class="col-md-4 col-form-label" >Promjeni dob djeteta</label>
                    <div class="col-md-8">
                        <select id="" class="form-control" name="age">
                            <option value="0" selected disabled>Odaberi dob djeteta</option>
                            <option value="0-6 mj" name="nula">0-6 mj</option>
                            <option value="6+ mj" name="sest">6+ mj</option>
```

Slika 58. Isječak koda account.blade.php datoteke

Datoteka *account.blade.php* povezana je s *UserController* datotekom u kojoj je definirana funkcija *postSaveAccount()* za spremanje izmjena u bazu podataka (slika 59). Funkcija dohvaća nove, unesene informacije o prijavljenom korisniku i sprema ih pod trenutnog korisnika. Nakon toga se vraća prikaz računa s izmijenjenim podacima.

```
public function  postSaveAccount (Request $request)
{
    $user=Auth::user();
    $user->name=$request['name'];
    $user->gender=$request['gender'];
    $user->age=$request['age'];

    $user->save();
    return Redirect::to( path: 'account');

}
```

Slika 59. postSaveAccount() funkcija

8.4. Ostale opcije

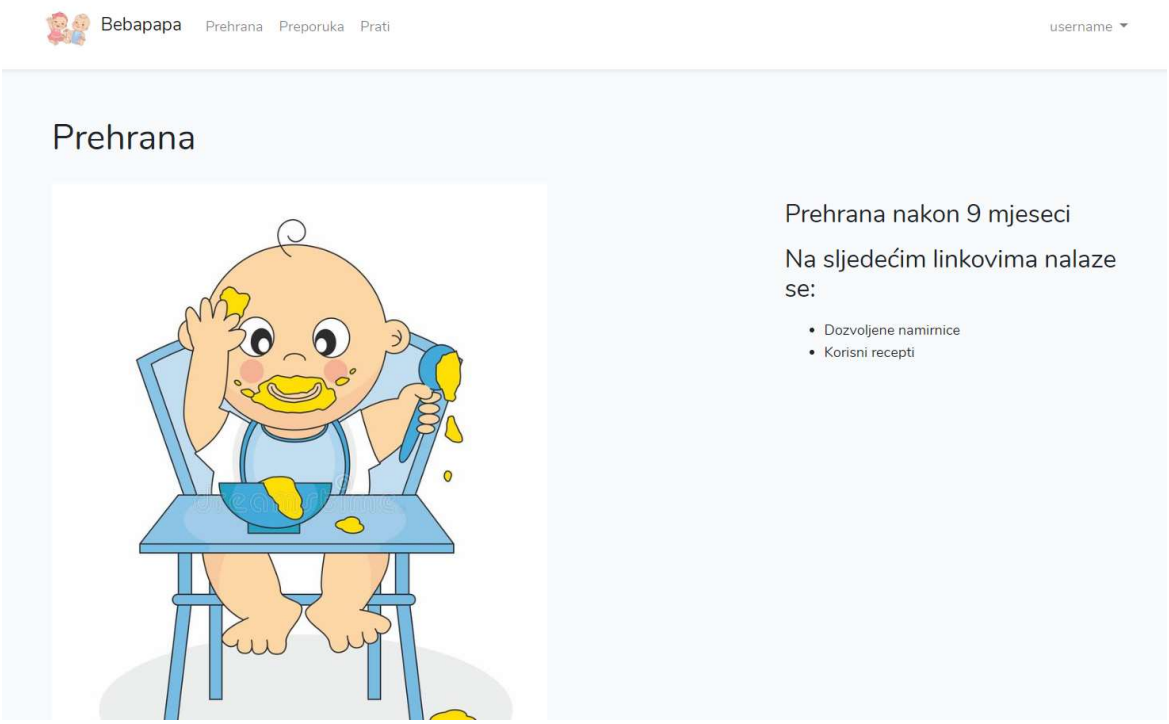
8.4.1. Preporučeni sadržaj

Nakon što se korisnik prijavi u aplikaciju, nudi mu se i opcija preporuke u navigacijskoj traci. Pod ovom opcijom korisniku će se otvoriti preporučene objave ovisno o dobi djeteta koju je korisnik postavio pri registraciji. Ova je opcija postavljena u *UserController* datoteci kao funkcija *getPreporuka()* kao *if* uvjet s obzirom na to da se dob djeteta sprema kao string u bazu podataka i ovaj dio aplikacije svakako zahtijeva dodatnu nadogradnju. Za svaku od preporučenih opcija kreiran je i novi prikaz, tj. *blade.php* datoteka. Funkcija je prikazana na slici 60.

```
public function getPreporuka()
{
    $age=Auth::user()->age;
    if ($age=='6+ mj') {
        return view( view: 'sest');
    }
    elseif ($age=='7+ mj') {
        return view( view: 'sedam');
    }
    elseif ($age=='8+ mj') {
        return view( view: 'osam');
    }
    elseif ($age=='9+ mj') {
        return view( view: 'devet');
    }
    elseif ($age=='10+ mj') {
        return view( view: 'deset');
    }
    elseif ($age=='11+ mj') {
        return view( view: 'jedanaest');
    }
    elseif ($age=='12+ mj') {
        return view( view: 'godina');
    }
    else {
        return view( view: 'prehrana');
    }
    }
```

Slika 60. getPreporuka() funkcija

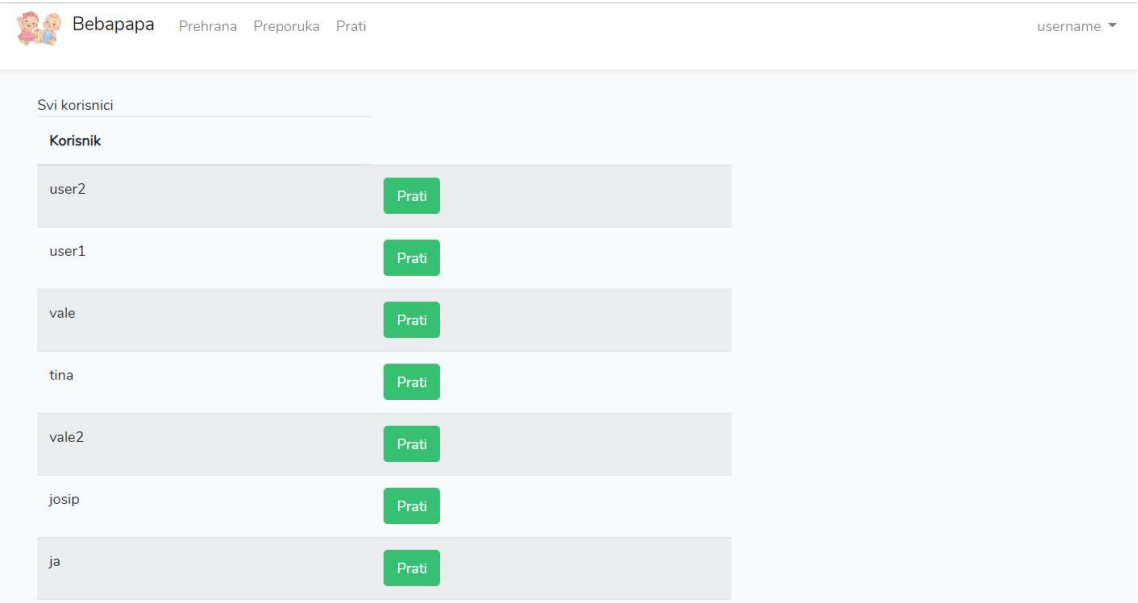
Korisnik *username* ima definiranu dob djeteta 9+ mjeseci te mu se otvara prikaz *devet,* a na slici 61 vidljiv je rezultat takvog odabira.



Slika 61. Preporuka

8.4.2. Praćenje korisnika

Korisnici mogu međusobno pratiti jedan drugoga, što je omogućeno opcijom za praćenje (engl. *follow/unfollow).* Za kreiranje ove mogućnosti kreiran je *Follow* model, *FollowController* datoteka te novi prikaz pod nazivom *index.blade.php* koji je spremljen u *views/users* direktorij. Kada korisnik otvori opciju *Prati,* otvorit će mu se popis svih registriranih korisnika koje može pratiti što je vidljivo na slici 62.



Slika 62. Praćenje korisnika

Prije svega kreirana je migracija *create_follows_table* te su dodani novi stupci u tablicu (slika 63).

```
 */
public function up()
{
    Schema::create( table: 'follows', function (Blueprint $table) {
        $table->increments( column: 'id');
        $table->integer( column: 'user_id')->index();
        $table->integer( column: 'target_id')->index(); //followed person
        $table->timestamps();
    });
}
```

Slika 63. create_follows_table migracija

Modeli su povezani relacijama prema ranijim primjerima, a *FollowController* sadrži funkciju *index()* koja dohvaća popis korisnika za prikaz *index* te funkcije *follow()* i *unfollow()* kojima se definiraju uvjeti za praćenje drugog korisnika. Funkcija *follow()* prvo će provjeriti je li korisnik već prati tog korisnika te, ako ne prati, ponuditi opciju Prati i vraća poruku o uspješnom praćenju, a u slučaju da već prati, vratit će poruku kako već prati tog korisnika. Slično, funkcija *unfollow()* provjerava prati li korisnik određeni ID; ako prati, vratit će poruku da više ne prati nakon pokretanja te funkcije, a ako korisnik nije ni pratio, vratit će mu poruku da ne prati tog korisnika. Kod je vidljiv na slici 64.

```php
public function index()
{
    return view( view: 'users.index', [
        'users'=>User::where('id', '!=', Auth::id())->get()
    ]);
}

public function follow(User $user)
{
    if(!Auth::user()->isFollowing($user->id)) {
        Auth::user()->follows()->create([
            '$target_id' => $user->id,
        ]);
        return back()->with('succes', 'Sada pratiš ovu osobu.'. $user->name);
    }else {
        return back()->with('error', 'Već pratiš ovu osobu.');
    }
}

public function unfollow(User $user)
{
    if(Auth::user()->isFollowing($user->id)) {
        $follow = Auth::user()->follows()->where('target_id', $user->id)->first();
        $follow->delete();
        return back()->with('succes', 'Više ne pratiš ovu osobu.'. $user->name);
    } else {
        return back()->with('error', 'Ne pratiš ovu osobu.');
    }
}
```

Slika 64. FollowController

9. Zaključak

Ovaj rad prikazuje najpoznatije i najkorištenije radne okvire te detaljno uspoređuje neke njihove mogućnosti. Kroz rad je detaljno prikazan Laravel radni okvir te je prikazan razvoj aplikacije na primjeru razvoja aplikacije društvene mreže. U prikazu razvoja prikazane su osnovne funkcionalnosti Laravel radnog okvira, no aplikacija u budućnosti ima prostora za nadogradnje. Zamišljene nadogradnje su automatsko povećavanje dobi djeteta korisnika čime bi se onda generirale preporuke bez potrebe korisnika za promjenom tih podataka. Ono što bi bilo potrebno napraviti je dohvatiti datum kreiranja računa korisnika i dob djeteta koju je korisnik zabilježio te dob povećati za mjesec dana nakon prolaska tog razdoblja. Također je moguće nadograditi opciju praćenja korisnika tako da se omogući dohvaćanje poveznice za praćenje korisnika preko njegovog korisničkog imena na objavi koju je kreirao, ali i da se prijavljenom korisniku prikazuju samo objave korisnika koje je pratio. Osim toga, opciju brisanja objave moguće je nadograditi tako da se prije brisanja pokaže upit za potvrdu brisanja objave.

Od početka rada s Laravel radnim okvirom informacije su se lako pronalazile zahvaljujući detaljnoj Laravel dokumentaciji, ali i velikom broju korisnika. Korištenje Laravel radnog okvira je intuitivno, instalacija jednostavna, a sam alat znatno olakšava razvoj takve osnovne aplikacije i implementaciju mnogobrojnih funkcionalnosti kojima se izbjegava zalihost koda. Arhitektura koju Laravel koristi znatno je olakšala razumijevanje i snalaženje u aplikaciji, a personalizacija svih implementiranih svojstava je lako dostupna i podesiva. Jednostavnost i automatizacija mnogih postupaka pri razvoju aplikacije su ono što Laravel čini poželjnim i dobrim izborom za početnike u području razvoja aplikacija.

10. Literatura

1. *Basic Routing*. (n.d.). Preuzeto 17. lipnja 2020 iz laravel.com: https://laravel.com/docs/7.x/routing#basic-routing
2. Beatty, J. (14. prosinca 2011). *i18n vs l10n — what's the diff?* Preuzeto 10. lipanj 2020 iz https://blog.mozilla.org/: https://blog.mozilla.org/l10n/2011/12/14/i18n-vs-l10n-whats-the-diff/
3. Bhartia, S. (9. travnja 2020). *Best PHP Frameworks for Web Development*. Preuzeto 17. lipnja 2020 iz hackr.io: https://hackr.io/blog/best-php-frameworks
4. *Blade Templates*. (n.d.). Preuzeto 15. lipnja 2020 iz laravel.com: https://laravel.com/docs/7.x/blade
5. *CakePHP Documentation*. (n.d.). Preuzeto 19. lipnja 2020 iz book.cakephp.org: https://book.cakephp.org/4/en/index.html
6. Christensson, P. (7. ožujak 2013). *Framework Definition*. Preuzeto 14. lipnja 2020 iz techterms.com: https://techterms.com
7. *CodeIngiter4 User Guide*. (svibanj 2020). Preuzeto 10. lipnja 2020 iz codeigniter.com: https://codeigniter.com/user_guide/index.html
8. *Controllers*. (n.d.). Preuzeto 17. lipnja 2020 iz laravel.com: https://laravel.com/docs/7.x/controllers#introduction
9. *Databases*. (n.d.). Preuzeto 4. lipnja 2020 iz laravel.com: https://laravel.com/docs/7.x/database#running-queries
10. *Directory Structure*. (n.d.). Preuzeto 17. lipnja 2020 iz laravel.com: https://laravel.com/docs/7.x/structure#the-bootstrap-directory
11. *Eloquent: Relationships*. (n.d.). Preuzeto lipnja. 19 2020 iz laravel.com: https://laravel.com/docs/7.x/eloquent-relationships#introduction
12. *Getting Started with Zend Framework*. (n.d.). Preuzeto 10. lipnja 2020 iz Zend Framework: https://framework.zend.com/learn
13. Gibb, R. (31. svibanj 2016). *What is Web application?* Preuzeto 15. lipnja 2020 iz stackpath.com : https://blog.stackpath.com/

14. *Introduction to gettext*. (n.d.). Preuzeto 15. lipnja 2020 iz gnu.org: https://www.gnu.org/software/gettext/

15. Jadhav, M. L., & Sodawala, M. B. (2015). Study of MVC Architecture. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), 4*(6), 2601-2604. Preuzeto 16. lipnja 2020 iz http://ijarcet.org/wp-content/uploads/IJARCET-VOL-4-ISSUE-6-2601-2604.pdf

16. Kaluža, M. K. (2019). A comparison of back-end frameworks for web application development. *Zbornik veleučilišta u rijeci, 7*(1), 317-332. Preuzeto 15. lipnja 2020 iz https://doi.org/10.31784/zvr.7.1.10

17. Laaziri, M., Benmoussa, K., Khoulji, S., & Larbi Kerkeb, M. (2019). A comparative study of PHP frameworks performance. *Procedia Manufacturing, 32*, 864-871. Preuzeto 13. lipnja 2020 iz https://www.sciencedirect.com/science/article/pii/S2351978919303312

18. *Laravel Homestead.* (n.d.). Preuzeto 18. lipnja 2020 iz laravel.com: https://laravel.com/docs/7.x/homestead

19. *Localization*. (n.d.). Preuzeto lipnja. 14 2020 iz laravel.com: https://laravel.com/docs/7.x/localization

20. Otwell, T. S. (2. svibnja 2017). *Can Laravel be used for big enterprise apps?* Preuzeto 10. lipnja 2020 iz laraveldaily.com: https://laraveldaily.com/can-laravel-used-big-enterprise-apps-summary-laravel-podcast/

21. Prokofyeva, N., & Boltunova, V. (2017). Analysis and Practical Application of PHP Frameworks in Development of Web Information Systems. *Procedia Computer Science, 104*, 51-56. Preuzeto 15. lipnja 2020 iz https://www.researchgate.net/publication/313488649_Analysis_and_Practical_Application_of_PHP_Frameworks_in_Development_of_Web_Information_Systems

22. Smijulj, A., & Meštrović, A. (2014). Izgradnja MVC modularnog radnog okvira. *Zbornik Veleučilišta u Rijeci, 2*(1), 215-232. Preuzeto 14. lipnja 2020 iz https://hrcak.srce.hr/128904

23. Stauffer, M. (2019). *Laravel Up & Running, 2nd Edition.* Sebastopol, CA: O'Reilly Media, Inc.

24. *Symfony Documentation*. (n.d.). Preuzeto 15. lipnja 2020 iz symfony.com: https://symfony.com/doc/current/index.html

25. Tatroe, K., & MacIntyre, P. (2020). *Programming PHP, 4th edition.* O'Reilly Media, Inc.

26. *Upgrade Guide*. (n.d.). Preuzeto 17. lipnja 2020 iz laravel.com: https://laravel.com/docs/7.x/upgrade#miscellaneous

27. Varisco, C. (17. listopada 2016). *How to choose a framework?* Preuzeto 18. lipnja 2020 iz https://hackernoon.com: https://hackernoon.com/how-to-choose-a-framework-ea8b5b1e1f44

28. *Views*. (n.d.). Preuzeto 17. lipnja 2020 iz laravel.com: https://laravel.com/docs/7.x/views#creating-views

Popis kratica

API - *Application Programming Interface*

BSD - *Berkeley Software Distribution*

CPU - *Central Processing Unit*

CRUD - *Create-Read-Update-Delete*

CSS - *Cascading Style Sheets*

CSV - *Comma-Separated Values*

DRY - *Don't Repeat Yourself*

HTML - *HyperText Markup Language*

HTTP - *HyperText Transfer Protocol*

IP - *Internet Protocol*

JSON - *JavaScript Object Notation*

MAMP - *macOS-Apache-MySQL/MariaDB-PHP/Perl/Python*

MIT - *Massachusetts Institute of Technology*

MVC - *Model View Controller*

NPM - *Node Packet Manager*

OPENSSL - *Secure Sockets Layer (otvoreni kod)*

ORM - *Object Relational Mapping*

PHP - *Hypertext Preproccesor*

REPL - *Read Evaluate Print Loop*

SQL - *Structured Query Language*

SSH - *Secure Shell*

URI - *Uniform Resource Identifier*

URL - *Uniform Resource Locator*

WAMP - *Windows-Apache-MySQL-PHP*

XAMPP - *XAMPP Apache-MariaDB-PHP-Perl*

XML - *EXtensible Markup Language*

Popis slika

Popis tablica

Razvoj web-aplikacije pomoću Laravel radnog okvira

Sažetak

U radu se daje pregled najpopularnijih tehnologija, odnosno okvira za razvoj web-aplikacija te se uspoređuju njihove mogućnosti, prednosti i nedostaci. Posebno je analiziran Laravel radni okvir. U radu se detaljno opisuje njegova arhitektura, funkcionalnost, korisničko sučelje i posebnosti. U sklopu rada razvijena je aplikacija s karakteristikama društvene mreže, na temelju koje je prikazan rad alata te njegova praktičnost i funkcionalnost u različitim scenarijima korištenja. Aplikacija sadrži tekstove o prehrani novorođenčadi i starije djece, a korisnici mogu kreirati sadržaj i komentirati te pratiti druge korisnike. Korisnik može definirati određene postavke na temelju kojih će sustav davati preporuke za prehranu djece.

Ključne riječi: Laravel, radni okvir, MVC arhitektura, web-aplikacija

Web application development using Laravel framework

Summary

The master's thesis provides an overview of the most popular technologies or frameworks for web application development and compares their functionalities, advantages and disadvantages. The Laravel framework is specifically analyzed and its architecture, functionality, user interface and features are described in detail. As a project work which represents use of Laravel framework, social network application was developed. Application contains texts on the nutrition of newborn and older children and users can create content, follow other users and comment on their posts. The user can define certain settings based on which the system will make recommendations for the nutrition of children.

Key words: Laravel, framework, MVC architecture, web application