

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Preddiplomski studij računarstva

Završni rad

**Web aplikacija za upravljanje virtualnim
hladnjakom uz preporuku recepata iz
dostupnih namirnica**

Rijeka, rujan 2021.

Barbara Breš
0069085153

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Preddiplomski studij računarstva

Završni rad

**Web aplikacija za upravljanje virtualnim
hladnjakom uz preporuku recepata iz
dostupnih namirnica**

Mentor: doc.dr.sc. Marko Gulić

Rijeka, rujan 2021.

Barbara Breš
0069085153

Umjesto ove stranice umetnuti zadatak
za završni ili diplomski rad

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradila ovaj rad.

Rijeka, rujan 2021.

Barbara Breš

Sadržaj

Popis slika	vii
Popis tablica	x
1 Uvod	1
2 Tehnologije i primjeri	3
2.1 Laravel	3
2.2 Laravel Jetstream	5
2.3 Livewire	6
2.4 Laravel Homestead	8
2.5 MariaDB	9
2.6 Bootstrap	9
2.7 Primjer	10
2.7.1 Instalacija i konfiguracija tehnologija	10
2.7.2 Kreiranje modela i migracije	13
2.7.3 Izrada Livewire komponente	15
2.7.4 Stvaranje rute	19
2.7.5 Izrada Blade pogleda	19
2.7.6 Pokretanje servera i prikaz aplikacije	23

Sadržaj

2.8	Bootstrap primjeri	24
2.8.1	Prijelomne točke, <i>eng.(breakpoints)</i>	24
2.8.2	Kontejneri	25
2.8.3	Mrežni sustav, <i>eng.(grid)</i>	25
2.8.4	Gumbi	27
3	Opis aplikacije	28
3.1	Funkcionalnosti	28
4	Opis funkcionalnosti - Dodavanje novog recepta	42
4.1	Opis pogleda <i>proba_create</i>	42
4.2	ER dijagram i modeli	46
4.3	Opis funkcija <i>create()</i> i <i>store()</i>	51
5	Zaključak	57
	Bibliografija	58
	Sažetak	61

Popis slika

2.1	Interakcije između modela, kontrolera i pogleda	4
2.2	Početna stranica (<i>Dashboard</i>) Jetstream aplikacije nakon prijave . .	5
2.3	Vežanje podataka u Livewireu	7
2.4	Učitavanje Livewire komponente unutar samog Blade predloška . . .	7
2.5	Početni zaslon Jetstream instalacije	12
2.6	Student model	13
2.7	Students tablica i funkcije migracije	14
2.8	Funkcija <i>render()</i>	15
2.9	Funkcije <i>create()</i> , <i>openModalPopover()</i> , <i>closeModalPopover()</i> i <i>resetCreateForm()</i>	16
2.10	Funkcija <i>store()</i>	17
2.11	Funkcija <i>edit()</i>	18
2.12	Funkcija <i>delete()</i>	18
2.13	Sintaksa zaglavlja aplikacije	19
2.14	Ispis poruka upozorenja	20
2.15	Gumb za kreiranje studenta	20
2.16	Uključivanje <i>laravel.create</i> pogleda	20
2.17	Sadržaj stranice unutar tablice	21
2.18	Obrazac za kreiranje studenta	22

Popis slika

2.19	Gumbi <i>Store</i> i <i>Close</i>	23
2.20	Završni izgled aplikacije	23
2.21	Razlika između <i>.container</i> i <i>.container-fluid</i> klase	25
2.22	Osnovni Bootstrapov mrežni sustav	26
2.23	Osnovni Bootstrapovi gumbi	27
3.1	Izgled početne stranice web aplikacije	28
3.2	Prozor za prijavu	29
3.3	Prozor za poništavanje lozinke	30
3.4	Prozor za registraciju	31
3.5	Glavna stranica web aplikacije	32
3.6	Filteri za kategoriju recepta i vremensko trajanje	32
3.7	Padajući izbornik unutar navigacijske trake	33
3.8	Prikaz glavne stranice aplikacije za mobilne uređaje	34
3.9	Stranica detalja recepta, 1.dio	35
3.10	Stranica detalja recepta, 2.dio	36
3.11	Stranica <i>Uredi recept</i> , 1.dio	37
3.12	Stranica <i>Uredi recept</i> , 2.dio	38
3.13	Stranica <i>Hladnjak</i>	39
3.14	Stranica <i>Hladnjak</i>	40
3.15	Stranica <i>Informacije o namirnicima</i>	41
4.1	Sintaksa za gumb <i>+Dodaj novi recept</i>	42
4.2	Linija koda za otvaranje obrasca	43
4.3	Dijelovi obrasca	44
4.4	Ispis svih sastojaka iz tablice <i>ingredients</i> unutar obrasca	45
4.5	Gumbi <i>Choose file</i> i <i>Spremi</i> te zatvaranje obrasca	45

Popis slika

4.6	ER dijagram tablica aplikacije	46
4.7	Veza jedan-na-više na strani modela <i>User</i>	47
4.8	Veza jedan-na-više na strani modela <i>Proba</i>	47
4.9	Veza jedan-na-jedan na strani modela <i>User</i>	48
4.10	Veza jedan-na-jedan na strani modela <i>Hladnjak</i>	48
4.11	Kreiranje tablice <i>ingredients_proba</i> unutar Laravel migracija	49
4.12	Veza više-na-više unutar <i>Ingredients</i> modela	49
4.13	Veza više-na-više unutar <i>Proba</i> modela	50
4.14	Pivot model <i>IngredientsProba</i>	50
4.15	Funkcija <i>create()</i>	51
4.16	Funkcija <i>store()</i> , prvi dio - validacija	52
4.17	Funkcija <i>store()</i> , drugi dio - čišćenje podataka	53
4.18	Funkcija <i>store()</i> , treći dio - spremanje slika	54
4.19	Funkcija <i>store()</i> , četvrti dio - spremanje recepta	55
4.20	Funkcija <i>store()</i> , peti dio - spremanje u pivot tablicu	56

Popis tablica

2.1	Bootstrap prijelomne točke	24
-----	--------------------------------------	----

Poglavlje 1

Uvod

Jedan od učestalih problema ljudske svakodnevice je konstantno razmišljanje što danas kuhati ili kako iskoristiti namirnice koje su dostupne kod kuće. Upravo taj problem inspirirao je kreiranje ove aplikacije.

Web aplikacija za upravljanje virtualnim hladnjakom uz preporuku recepata iz dostupnih namirnica, naziva *Virtualni hladnjak*, radi upravo ono što i sam naziv rada govori. Korisnikove upisane namirnice sprema u njegov virtualni hladnjak te pomoću njih na početnoj stranici prikazuju se svi dostupni recepti za koje korisnik trenutno ima sve namirnice u dovoljnim količinama.

Za razvoj poslužiteljskog dijela web aplikacije korišten je PHP-ov radni okvir **Laravel 8** dok je za razvoj klijentskog dijela web aplikacije korišten **Laravel Jetstream** paket s **Livewire** knjižnicom kao i HTML-ov, CSS-ov i JavaScriptov radni okvir **Bootstrap** za sam dizajn web aplikacije.

Laravel je besplatni PHP-ov razvojni okvir, otvorenog koda, kojeg je izradio Taylor Otwell i namijenjen je razvoju web aplikacija prema arhitektonskom uzorku model-pogled-kontroler (MVC) [1]. Također uvelike olakšava izradu samih web aplikacije svojom izražajnom i elegantnom sintaksom [2].

Poglavlje 1. Uvod

Laravelov paket **Jetstream** služi kao odlična početna točka prilikom izrade Laravel aplikacija dolazeći s već implementiranim stranicama za prijavu, registraciju, verifikaciju e-pošte, itd., omogućavajući fokusiranje na izradu glavnog dijela web aplikacije bez razmišljanja o implementaciji svega navedenog [3].

Za izradu sučelja, kao dodatak Laravel Jetstreamu, koristi se **Livewire** knjižnica koja na jednostavan način omogućuje izradu dinamičkog sučelja, ne napuštajući udobnost Laravela [4].

Za dizajn web aplikacija zadužen je HTML-ov, CSS-ov i JavaScriptov besplatni radni okvir otvorenog koda, **Bootstrap**. Jedan od njegovih zadataka je napraviti web aplikacije s mogućnošću učinkovitog renderiranja na uređajima sa zaslonima različitih veličina te sadrži predloške za različite komponente web aplikacije, npr. gumbi, navigacijske trake, obrasci, itd. [5].

Poglavlje 2

Tehnologije i primjeri

2.1 Laravel

PHP-ov besplatni radni okvir otvorenog koda, **Laravel**, kreiran je od strane Taylora Otwellia kao pokušaj naprednije alternative dotadašnjeg CodeIgniter radnog okvira, pružajući ugrađenu podršku za autentifikacija i autorizacija korisnika. Prva verzija izašla je 9. srpnja 2011. godine i sadržavala je ugrađenu podršku za autentifikacija i autorizacija korisnika, modele, poglede, sesije, usmjeravanja itd., no nije imala podršku za kontrolere što ju je spriječilo da bude potpuni MVC (*eng. (Model-view-controller)*) radni okvir. Laravel postaje potpuni MVC radni okvir tek u Laravel 2 verziji.[1]

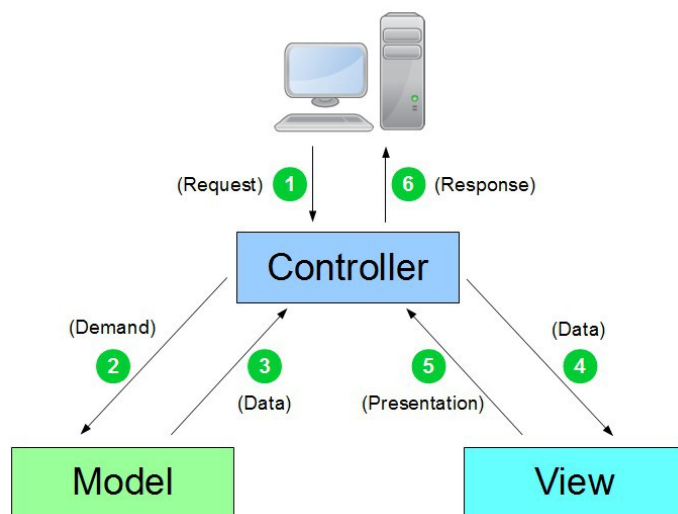
MVC ili *eng. (Model-view-controller)* je softverski dizaj korišten za razvoj korisničkih sučelja koji dijele povezanu programsku logiku na tri međusobno povezana elementa (modeli, pogledi i kontroleri). Modeli su središnja komponenta softverskog dizajna te su također i dinamička struktura podataka aplikacije. Oni izravno upravljaju podacima, logikom i pravilima aplikacije. Pogledi su bilo koji prikaz informacija poput grafikona, dijagrama, tablice ili same web stranice te je moguće imati više prikaza istih informacija na istim ili različitim pogledima. Kontroleri prihvataju unos i pretvaraju ga u naredbe za modele ili prikaze.

Poglavlje 2. Tehnologije i primjeri

Definirane interakcije između dijelova MVC softverskog dizajna su:

1. Model je odgovoran za upravljanje podacima aplikacije te od kontrolera prima korisnički unos.
2. Pogled prikazuje podatke iz modela u određenom formatu.
3. Kontroler prima ulaz tj. unos korisnika, reagira na njega i vrši interakcije na objektima modela podataka. Po želji može prvo potvrditi unos korisnika, a zatim prenijeti unos modelu. [6]

Slika 2.1 prikazuje navedene interakcije.



Slika 2.1 Interakcije između modela, kontrolera i pogleda [7]

Neke od bitnih značajki koje su dodane kroz razne verzije Laravela su **Blade** - sistem predložaka koji kombiniraju jedan ili više predložaka s modelom podataka za stvaranje rezultirajućih prikaza, čineći to prevođenjem predložaka u predmemorirani PHP kod radi poboljšanih performansi, **Artisan** - Laravelovo sučelje naredbenog retka *eng.(command-line interface)* čije su neke od **upotrebe**: migracija tablica u bazu podataka, kreiranje modela i kontrolera te izrada Livewire komponenti; **migracije baza podataka** te u najnovijoj verziji Laravela, Laravel 8, koji je objavljen 8.rujna 2020. godine dodan je paket **Laravel Jetstream** o kojem će biti pisano u nastavku. [1]

2.2 Laravel Jetstream

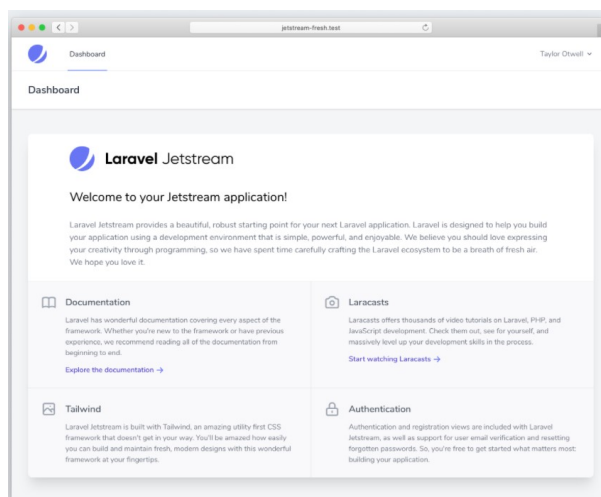
Laravel Jetstream je Laravelov paket kreiran kao početna točka za izradu Laravel aplikacija već implementirajući stranice za prijavu, registraciju, provjeru e-pošte, itd. Dizajniran je koristeći Tailwind CSS te nudi dvije knjižnice za izradu dinamičkog sučelja, Livewire, o kojem će biti riječi kasnije, i Inertia [3].

Tailwind CSS je CSS-ov radni okvir koji, za razliku od Bootstrapa koji je *eng.(UI kit)* tj. ima unaprijed definiranu temu i komponente korisničkog sučelja, se bazira na *eng.(utility-first)* principu tj. dolazi s već unaprijed definiranim grafičkim elementima *eng.(widgets)* kojima se može brže izgraditi cijela aplikacija ispočetka [8].

Sam Jetstream dolazi s dva unaprijed definirana izgleda stranica *eng.(layout)* za aplikaciju, koja se nalazi na `resources/views/layouts/app.blade.php` putanji i generirana je pomoću `App \ View \ Components \ AppLayout` klase te za goste tj. izgled stranice dok korisnik još nije prijavljen, a nalazi se na `resources/views/layouts/guest.blade.php` putanji i generirana je pomoću

`App \ View \ Components \ GuestLayout` klase. Ovo vrijedi samo prilikom korištenja Livewire knjižnice, za Inertiu je malo drugačije. [9]

Na slici 2.2 prikazana je početna stranica imena *Dashboard* kojoj se može pristupiti nakon prijave u aplikaciju te unaprijed definirani izgled aplikacije *eng.(app layout)*.



Slika 2.2 Početna stranica (*Dashboard*) Jetstream aplikacije nakon prijave [3]

2.3 Livewire

Livewire je knjižnica koja pojednostavljuje izradu modernih, reaktivnih i dinamičnih sučelja koristeći **Laravel Blade** kao jezik za predloške. [4]

Laravel Blade je jednostavan, ali moćan predložak koji je uključen u sam Laravel. Za razliku od nekih PHP-ovih predložaka, Blade ne ograničava korištenje običnog PHP koda unutar predložaka. Zapravo, svi Blade predlošci su prevedeni u običan PHP kod i predmemorirani dok se ne dogodi promijena. Datoteke Blade predložaka koriste ekstenziju *.blade.php* i obično su pohranjene u direktoriju *resources/views*. Pogledi Bladea mogu se vratiti s rutama ili kontrolerima pomoću globalnog pomoćnika za prikaz, *view()*: npr. *return view('greeting');* gdje je *greeting* ime blade datoteke. Također podaci se mogu proslijediti na pogled Bladea pomoću drugog argumenta pomoćnika za prikaz: *return view('greeting', ['name' => 'Finn']);* ili pomoću metode *with()*: *return view('greeting')->with('name', \$name);*. [10]

Za stvaranje nove Livewire komponente koristi se *artisan* naredba:

```
php artisan make:livewire ServerList
```

Nakon pokretanja *artisan* naredbe stvaraju se dvije nove datoteke, Livewire komponenta *ServerList* na putanji *app/Http/Livewire/ServerList.php* i Livewire pogled *server-list* na putanji *resources/views/livewire/server-list.blade.php*. [11]

Pomoću Livewire komponenti može se "vezati" (ili "sinkronizirati") trenutna vrijednost nekog HTML-ovog elementa s određenim svojstvom u komponenti. To svojstvo "vezanja" naziva se vezivanje podataka, *eng.(data binding)* i prikazano je na slici 2.3 gdje pomoću svojstva *wire:model*, koji je povezan s vrijednosti *\$message* svojstva, svakim novim unosom unutar polja za tekst automatski će se ažurirati vrijednost *\$message* svojstva. Svojstvo *wire:model* može biti dodano na bilo koji HTML-ov element koji otprema ulazni događaj *eng.(input event)* kao što su *<input type="text">*, *<input type="radio">*, *<select>*, itd. [12]

Poglavlje 2. Tehnologije i primjeri

```
1 class HelloWorld extends Component
2 {
3     public $message;
4 }
```

```
1 <div>
2     <input wire:model="message" type="text">
3
4     <h1>{{ $message }}</h1>
5 </div>
```

Slika 2.3 Vezanje podataka u Livewireu [12]

Učitavanje Livewire komponente unutar Blade predloška prikazano je na slici 2.4. Pomoću Blade direktive *@livewire* na jednostavan način je moguće učitati Livewire komponentu, na slici 2.4 imena *server-list*, unutar Blade predloška. [13]

```
<div class="mt-4">
    @livewire('server-list')
</div>
```

Slika 2.4 Učitavanje Livewire komponente unutar samog Blade predloška [4]

2.4 Laravel Homestead

Laravel Homestead službena je, već zapakirana, Vagrant kutija koja pruža razvojno okruženje bez potrebe za instaliranjem PHP-a, web poslužitelja i bilo kojeg drugog poslužiteljskog softvera na lokalnom računalu. Homestead radi na bilo kojem operacijskom sustavu i uključuje Nginx, PHP, MySQL, PostgreSQL, Redis, Memcached, Node i sav drugi softver koji je potreban za razvoj Laravel aplikacija . [14]

Vagrant je alat otvorenog koda koji omogućuje stvaranje, konfiguriranje i upravljanje kutijama virtualnih strojeva putem naredbenog sučelja koje je jednostavno za korištenje tj. to je sloj softvera instaliran između alata za virtualizaciju (poput VirtualBox-a, Dockera, Hyper-V-a) i virtualne mašine, *eng. (VM, Virtual Machine)*. [15]

Prije same instalacije i postavljanja Homestead okruženja potrebno je instalirati Vagrant i jednog od podržanih pružatelja usluga kao što je VirtualBox.

Sam Homestead se može instalirati kloniranjem Homestead repozitorija na računalo domaćina. Preporučeno je spremati klonirani repozitorij u mapu Homestead koja se nalazi unutar "*home*" ili *~* direktorija jer će virtualni stroj Homestead poslužiti kao domaćin svim Laravel aplikacijama. Naredba za kloniranje Homestead repozitorija je: ***git clone https://github.com/laravel/homestead.git ~/Homestead***. Detaljne upute za postavljanje samog okruženja dostupne su na službenoj stranici Laravel Homestead a čija *poveznica se* može pronaći u bibliografiji.

Nakon uspješnog postavljanja, okruženje se pokreće iz *release* grane do koje se može doći idućim naredbama: *cd ~/Homestead* i *git checkout release*. Ulaskom u granu *release* upisuje se naredba za podizanje Homestead okruženja: ***vagrant up*** i naredba za spajanje na virtualni stroj i provjeru konekcije pomoću SSH konekcije: ***vagrant ssh***. Na kraju odlazi se do direktorija u kojem se nalaze već kreirane Laravel aplikacije ili u direktorij gdje će biti kreirane, otkud se za daljnji rad s aplikacijama koriste Laravel ***artisan*** naredbe. [14]

2.5 MariaDB

MariaDB jedna je od najpopularnijih relacijskih baza podataka MySQL-ovog sustava za upravljanje relacijskim bazama podataka, *eng.(RDBMS)*, koja je također otvorenog koda. Napravljena je od strane originalnih programera MySQL-a 2009. godine kao alternativa MySQL-u koji je tada prešao u vlasništvo Oracle Corp'sa. Od svojeg nastanka pa do danas u širokoj je upotrebi kao dio ponude većina usluga u oblaku, *eng.(cloud services)*, a također je zadana baza podataka u većini distribucija Linuxa. Temelji se na SQL-u i podržava obradu podataka u stilu ACID-a sa zajamčenom atomarnosti, konzistentnosti, izolacijom i durabilnosti za transakcije, *eng.(atomicity, consistency, isolation and durability)*. Između ostalih značajki, baza podataka također podržava JSON API-je, paralelnu replikaciju podataka i više strojeva za pohranu, uključujući InnoDB, MyRocks, Spider, Aria, TokuDB, Cassandra i MariaDB ColumnStore. Bitna značajka je ta što je binarno kompatibilna s MySQL-om pri čemu je prijelaz s jedne baze podataka na drugu skoro neprimjetan. [16]

2.6 Bootstrap

Bootstrap je besplatan HTML-ov, CSS-ov i JavaScriptov radni okvir otvorenog koda. Kreiran od strane Mark Otta i Jacoba Thorntona, pod originalnim imenom *Twitter Blueprint*, u želji poticanja uniformnosti korisničkog sučelja Twittera kao i lakšeg održavanja samog sučelja. Tek 2011. godine je pušten u javnost kao radni okvir otvorenog koda. Jedna od glavnih značajki Bootstrapa je dinamično skaliranje web aplikacija tj. učinkovito renderiranje na uređajima sa zaslonima različitih veličina. Osim toga Bootstrap održava široku kompatibilnost za različite preglednike, nudi dosljedan dizajn pomoću komponenti koje se mogu ponovno koristiti, a vrlo je jednostavan za korištenje i brzo se uči. Također nudi bogatu proširivost pomoću JavaScripta, s ugrađenom podrškom za jQuery dodatke i programskim JavaScript API-jem. Bootstrap se može koristiti s bilo kojim IDE-om ili alatom za uređivanje te bilo kojom tehnologijom i jezikom na strani poslužitelja, od ASP.NET-a do PHP-a. Zbog svega navedenog Bootstrap je izbor većine programera za gradnju korisničkog

sučelja jer dopušta koncentriranje na razvoj web aplikacije, bez brige o dizajnu. Obrnuto, web dizajnerima daje čvrste temelje za stvaranje zanimljivih Bootstrap tema zbog jednostavnosti prilagođavanja. Samo neke od UI komponenti s kojima već dolazi ugrađen su: gumbi, obrasci, navigacijske trake, upozorenja, itd. [17] Primjeri za neke od navedenih UI komponenti biti će prikazani u kasnijoj sekciji naziva *Bootstrap primjeri*.

Sama implementacija Bootstrapa u web aplikacije je poprilično jednostavna i može se pronaći na službenoj stranici Bootstrapa čija se poveznica nalazi u bibliografiji pod brojem: [18]

2.7 Primjer

Naredni primjer preuzet je s interneta, no neki koraci će biti nadodani kako bi se pokrila većina gore navedenih tehnologija te će koraci biti opisani za rad na Windows operacijskom sustavu. Poveznica na primjer može se pronaći u bibliografiji pod imenom *Build Laravel Livewire CRUD Application using Jetstream*. [19]

2.7.1 Instalacija i konfiguracija tehnologija

Prije same izrade **CRUD**, *eng.(create, read, update and delete)*, aplikacije potrebno je instalirati sve potrebne tehnologije te pripremiti samo virtualno okruženje.

Potrebno je na računalu imati instaliranu najnoviju verziju PHP-a, no ukoliko na računalu uopće ne postoji PHP ili postoji starija verzija to se može učiniti odlaskom na poveznicu koja se nalazi u bibliografiji pod nazivom *Download PHP* tj. brojem: [20]. Nakon same instalacije ili ažuriranja PHP-a potrebno je u varijablu okruženja *Path* dodati putanju do PHP mape.

Nakon PHP-a potrebno je instalirati VirtualBox, Vagrant kao i Git koji su potrebni za Laravel Homestead virtualno okruženje. Poveznice za instalaciju potrebnih komponenti mogu se pronaći u bibliografiji pod brojevima [21] za VirtualBox, [22] za Vagrant i [23] za Git.

Poglavlje 2. Tehnologije i primjeri

Završetkom gore navedenih potrebnih instalacija potrebno je ući u Git Bash, instalirati Homestead Vagrant kutiju naredbom: `vagrant box add laravel/homestead`, a zatim instalirati Laravel Homestead kloniranjem repozitorija u *Homestead* mapu unutar "*home*" ili `~` direktorija naredbom: `git clone https://github.com/laravel/homestead.git ~/Homestead`. Nakon uspješnog kloniranja potrebno je naredbama: `cd ~/Homestead` i `git checkout release` ući unutar *release* grane otkuda će se pozvati naredba: `bash init.sh` kojom se kreira *Homestead.yaml* konfiguracijska datoteka unutar koje se konfiguriraju sve postavke za instalaciju Homestead. Naredbom: `ssh-keygen -t rsa -C "you@homestead"` generiraju se SSH ključevi. Unutar *Homestead.yaml* datoteke potrebno je promijeniti linije koda ispod *folders* i *sites* svojstva. Unutar *folders: map:* svojstva zapisuje se putanja do mape unutar koje se nalaze ili će se nalaziti Laravel aplikacije, dok *folders: to:* svojstvo će mapirati putanju iz *folders: map:* svojstva u **virtualnu mašinu** tako da ime može biti isto kao ime mape tj. umjesto `/home/vagrant/code` pisati će `/home/vagrant/ime-mape-napisane-unutar-map-svojstva`. Svojstvo *sites: map:* sadrži domenu s kojom je moguće pristupiti Laravel aplikaciji kroz preglednik, a *sites: to:* svojstvo pokazuje na *public* direktorij Laravel aplikacije stoga će *to:* svojstvo izgledati otprilike ovako: `/home/vagrant/putanja-koja-se-nalazi-unutar-folder:map:/ime-mape-aplikacije/public`. Domenu koja je zapisana pod *sites: map:* svojstvom potrebno je dodati unutar *hosts* datoteke koja se nalazi na putanji `C:\Windows\System32\drivers\etc` te koju je potrebno otvoriti kao administrator kako bi se mogla uređivati. Dodati ip adresu, koja piše pod *ip:* svojstvom unutar *Homestead.yaml* datoteke, kao i domen, unutar *hosts* datoteke ispod navedenih **ip** adresa i domena te spremiti promjene.

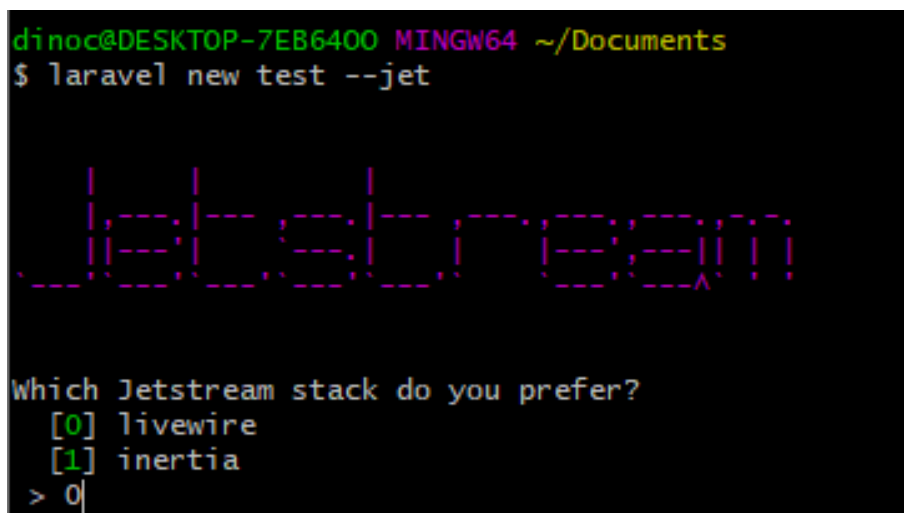
Homestead dolazi već s raznim bazama podataka te je moguće uključiti željenu samo promjenom vrijednosti iz *false* u *true*. Za ovaj primjer koristiti će se MariaDB tj. MySQL kod kojeg će se napisati vrijednost *true*. Ukoliko se želi koristiti drugačija baza podataka od već ugrađene *homestead* baze, potrebno ju je dodati pod *databases:* svojstvo te zatim kreirati novog korisnika sa svim ovlastima pošto iz nekog razloga za *root* korisnika ne radi pristup. Nakon kreacije novog korisnika potrebno je otići u *.env* datoteku aplikacije te promijeniti vrijednosti ovih varijabli: *DB_DATABASE*, *DB_USERNAME* i *DB_PASSWORD* u vrijednosti koje su novozadane.

Poglavlje 2. Tehnologije i primjeri

Za podizanje Homestead okruženja koristi se naredba *vagrant up* te *vagrant ssh* naredba za spajanje na Homestead okruženje tj. virtualnu mašinu.

Nakon navigacije do mape koja je specificirana kao *"root"* mapa za sve Laravel aplikacije, pomoću naredbe *cd imeMape*, pokreće se naredba za instalaciju Laravla: *composer global require laravel/installer*.

Kreiranje nove Jetstream aplikacije vrši se naredbom: *laravel new ime-aplikacije --jet*. Ukoliko je naredba bila uspješna trebao bi se prikazati početni zaslon Jetstream instalacije poput ovog na slici 2.5.

A screenshot of a terminal window showing the output of the 'laravel new test --jet' command. The terminal title is 'dinoc@DESKTOP-7EB6400 MINGW64 ~/Documents'. The command executed is '\$ laravel new test --jet'. The output shows the 'Jetstream' logo in a stylized, dashed font. Below the logo, it asks 'Which Jetstream stack do you prefer?' and lists two options: '[0] livewire' and '[1] inertia'. The user has entered '> 0'.

Slika 2.5 Početni zaslon Jetstream instalacije [24]

Na slici 2.5 prikazane su također i opcije za stogove koji se koriste s Jetstreamom. U ovom primjeru koristiti će se Livewire.

Završetkom Livewire instalacije potrebno je pokrenuti još par naredbi. Prije pokretanja bilo kakvih naredbi potrebno je ući u novostvorenu mapu aplikacije naredbom *cd ime-mape*. Unutar te mape pokreću se naredbe *npm install* i *npm run dev*, no ukoliko te naredbe ne prolaze unutar Git Basha, mogu se pokrenuti iz naredbenog retka, *eng.(command prompt, cmd)*.

Kao zadnji korak potrebno je pokrenuti migraciju baze podataka naredbom `php artisan migrate` te nakon toga moguće je pristupiti aplikaciji putem web preglednika koristeći domenu koja je bila definirana unutar `sites: map:` svojstva.

2.7.2 Kreiranje modela i migracije

Pomoću Laravel Artisana moguće je stvoriti model i migraciju samo jednom naredbom: `php artisan make:model Student -m` (`-m` poziva se stvaranje migracije). Izvršenjem naredbe stvorit će se datoteke modela `Student.php` te migracije `create_students_table.php`.

Potrebno je unutar datoteke `Student.php` dodati `$fillable` polje s imenima redaka koji se nalaze unutar tablice `students` kao što je prikazano na slici 2.6.

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Student extends Model
{
    use HasFactory;

    protected $fillable = [
        'name',
        'email',
        'mobile'
    ];
}
```

Slika 2.6 Student model [19]

Poglavlje 2. Tehnologije i primjeri

Također, potrebno je konfigurirati i tablicu *students* dodavanjem potrebnih redaka i njihovih svojstva unutar migracije što je vidljivo iz slike 2.7. Funkcijom *up()* stvara se tablica *student* koja sadrži retke: *id*, *name*(tip string, do 100 znakova), *email*(tip string), *mobile*(tip string) te *timestamp*(kada je red kreiran i ažuriran) dok funkcijom *down()* se briše tablica *students*.

```
<?php

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

class CreateStudentsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('students', function (Blueprint $table) {
            $table->id();
            $table->string('name', 100);
            $table->string('email');
            $table->string('mobile');
            $table->timestamps();
        });
    }

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

Slika 2.7 Students tablica i funkcije migracije [19]

Nakon dodavanja novih redaka i konfiguracije tablice *students*, potrebno je pokrenuti naredbu *php artisan migrate* kako bi se sve nove izmjene primijenile unutar tablice koja se nalazi u bazi podataka.

2.7.3 Izrada Livewire komponente

Naredbom *php artisan make:livewire crud* kreira se Livewire *Crud.php* komponenta kao i Livewire *crud.blade.php* pogled.

Potrebno je definirati **CRUD, eng. (create, read, update and delete)**, metode unutar *Crud.php* datoteke.

Funkcija *render()*, princip *read* metode, prikazana na slici 2.8, dohvaća sve *Student* objekte iz *students* tablice te ih sprema u varijablu *\$this->students*. Na kraju vraća *livewire.crud* pogled na kojem će biti prikazani svi studenti.

```
<?php

namespace App\Http\Livewire;
use Livewire\Component;
use App\Models\Student;

class Crud extends Component
{
    public $students, $name, $email, $mobile, $student_id;
    public $isModalOpen = 0;

    public function render()
    {
        $this->students = Student::all();
        return view('livewire.crud');
    }
}
```

Slika 2.8 Funkcija *render()* [19]

Poglavlje 2. Tehnologije i primjeri

Create() funkcija poziva dvije funkcije: *resetCreateForm()* i *openModalPopover()*. Funkcija *resetCreateForm()* postavlja sve vrijednosti na prazan niz, eng. (*string*), dok funkcija *openModalPopover()* postavlja vrijednost *isModalOpen* na vrijednost *true*. Suprotno njoj funkcija *closeModalPopover()* postavlja vrijednost *isModalOpen* na vrijednost *false*. Te vrijednosti biti će potrebne za pozivanje Livewire pogleda *livewire.create* unutar pogleda *livewire.crud*. Navedene funkcije prikazane su na slici 2.9.

```
public function create()
{
    $this->resetCreateForm();
    $this->openModalPopover();
}

public function openModalPopover()
{
    $this->isModalOpen = true;
}

public function closeModalPopover()
{
    $this->isModalOpen = false;
}

private function resetCreateForm(){
    $this->name = '';
    $this->email = '';
    $this->mobile = '';
}
```

Slika 2.9 Funkcije *create()*, *openModalPopover()*, *closeModalPopover()* i *resetCreateForm()* [19]

Poglavlje 2. Tehnologije i primjeri

Za spremanje novih studenata i njihovih informacija u tablicu *students* ili ažuriranje podataka već postojećih studenata koristi se funkcija *store()* prikazana na slici 2.10. Unutar same funkcije validiraju se polja *name*, *email* i *mobile* koja su potrebna za ispuniti. Završetkom validacije provjerava se postoji li već student unutar tablice *students*, ili ne, jer o toj informaciji ovisi hoće li biti dodan novi student ili će se ažurirati informacije već postojećeg. Na samom kraju pojavit će se poruka "*Student updated.*" ako su vrijednosti ažurirane ili "*Student created.*" ako je dodan novi student. Pozvat će se funkcije *closeModalPopover()* i *resetCreateForm()* kako bi se vrijednost *isModalOpen* prebacila iz *true* u *false* te same vrijednosti unutar obrasca se resetirale tj. postale prazan niz.

```
public function store()
{
    $this->validate([
        'name' => 'required',
        'email' => 'required',
        'mobile' => 'required',
    ]);

    Student::updateOrCreate(['id' => $this->student_id], [
        'name' => $this->name,
        'email' => $this->email,
        'mobile' => $this->mobile,
    ]);

    session()->flash('message', $this->student_id ? 'Student updated.' : 'Student created.');
```

```
    $this->closeModalPopover();
    $this->resetCreateForm();
}
```

Slika 2.10 Funkcija *store()* [19]

Poglavlje 2. Tehnologije i primjeri

Uređivanje informacija studenta vrši funkcija *edit()* prikazana na slici 2.11. Provjerava postoji li traženi student u tablici *students*, te ako postoji, ažurira njegove informacije i poziva *openModalPopover()* funkciju koja će pridodijeliti *isModalOpen* varijabli vrijednost *true*.

```
public function edit($id)
{
    $student = Student::findOrFail($id);
    $this->student_id = $id;
    $this->name = $student->name;
    $this->email = $student->email;
    $this->mobile = $student->mobile;

    $this->openModalPopover();
}
```

Slika 2.11 Funkcija *edit()* [19]

Zadnja funkcija, *delete()*, briše traženog studenta iz tablice *students* ukoliko on postoji te vraća poruku "*Student deleted.*" ukoliko je uspješno obavljeno brisanje. Sve navedeno vidljivo je na slici 2.12

```
public function delete($id)
{
    Student::find($id)->delete();
    session()->flash('message', 'Student deleted.');
```

Slika 2.12 Funkcija *delete()* [19]

2.7.4 Stvaranje rute

Unutar datoteke *routes/web.php* potrebno je dodati iduću liniju koda kako bi bilo moguće pristupiti *students* pogledu tj. web stranici na kojoj će biti prikazani svi studenti: *Route::get('students', Crud::class)*. Također, potrebno je uvesti *Crud* klasu pomoću *use App\Http\Livewire\Crud*; linije kako bi se ona mogla koristiti unutar rute. Sada je moguće pristupiti *students* web stranici, no ona trenutno ne postoji pa ju je potrebno napraviti.

2.7.5 Izrada Blade pogleda

Detaljan izgled pogleda nalazi se na poveznici pod brojem [19]. Ovdje će biti opisani samo mali dijelovi kao i završni izgled aplikacije.

Sintaksa za zaglavlje aplikacije vidljiva je na slici 2.13. Oznaka *<x-slot>* je Bladeova oznaka (sve Bladeove oznake započinju s *x-*) za utor koji će popuniti mjesto za zaglavlje sa sadržajem HTML-ove oznake *<h2>*.

```
<x-slot name="header">
    <h2 class="text-center">Laravel 8 Livewire CRUD Demo</h2>
</x-slot>
```

Slika 2.13 Sintaksa zaglavlja aplikacije [19]

Slika 2.14 prikazuje provjeru ukoliko sesija ima kakvu poruku te ju ispisuje unutar prozora za upozorenje, *eng.(alert)*.

Poglavlje 2. Tehnologije i primjeri

```
@if (session()->has('message'))
<div class="bg-teal-100 border-t-4 border-teal-500 rounded-b
    role="alert">
    <div class="flex">
        <div>
            <p class="text-sm">{{ session('message') }}</p>
        </div>
    </div>
</div>
@endif
```

Slika 2.14 Ispis poruka upozorenja [19]

Livewire čeka da se događaj *click* okine pritiskom na gumb *Create Student*. Nakon okidanja događaja poziva se *create()* funkcija iz *Crud.php* datoteke što je vidljivo na slici 2.15

```
<button wire:click="create()"
    class="my-4 inline-flex justify-center w-full rounded-md
    Create Student
</button>
```

Slika 2.15 Gumb za kreiranje studenta [19]

Ukoliko je pritisnut gumb *Create Student* unutar *create()* funkcije pokreće se *open-ModalPopover()* funkcija koja postavlja vrijednost varijable *\$isModalOpen* na *true* te time uključuje *livewire.create* pogled unutar *livewire.crud* pogleda.

```
@if($isModalOpen)
@include('livewire.create')
@endif
```

Slika 2.16 Uključivanje *laravel.create* pogleda [19]

Poglavlje 2. Tehnologije i primjeri

Ostatak sadržaja pogleda prikazan je unutar tablice koja se popunjava sa studentima i njihovim informacijama pomoću *@foreach* petlje, iterirajući po svakom objektu studenta, kao što prikazuje slika 2.17. Pritiscima na gumbe *Edit*, odnosno *Delete* pozivaju se funkcije *edit()*, odnosno *delete()* u koje se šalje *id* vrijednost trenutnog tj. odabranog studenta.

```
<table class="table-fixed w-full">
  <thead>
    <tr class="bg-gray-100">
      <th class="px-4 py-2 w-20">No.</th>
      <th class="px-4 py-2">Name</th>
      <th class="px-4 py-2">Email</th>
      <th class="px-4 py-2">Mobile</th>
      <th class="px-4 py-2">Action</th>
    </tr>
  </thead>
  <tbody>
    @foreach($students as $student)
      <tr>
        <td class="border px-4 py-2">{{ $student->id }}</td>
        <td class="border px-4 py-2">{{ $student->name }}</td>
        <td class="border px-4 py-2">{{ $student->email }}</td>
        <td class="border px-4 py-2">{{ $student->mobile }}</td>
        <td class="border px-4 py-2">
          <button wire:click="edit({{ $student->id }})"
            class="flex px-4 py-2 bg-gray-500 text-gray-900 cursor-pointer">Edit</button>
          <button wire:click="delete({{ $student->id }})"
            class="flex px-4 py-2 bg-red-100 text-gray-900 cursor-pointer">Delete</button>
        </td>
      </tr>
    @endforeach
  </tbody>
</table>
```

Slika 2.17 Sadržaj stranice unutar tablice [19]

Poglavlje 2. Tehnologije i primjeri

Obrazac za kreiranje studenta prikazan je na slici 2.18. Sastoji se od okvir za tekst, za upis imena, te dva tekstualna područja, *eng.(textarea)*, za upis e-pošte i broja mobitela. Ukoliko se neko ili sva polja ne ispune, pojavit će se poruke pogreške da je određeno polje potrebno ispuniti.

```
<div class="mb-4">
  <label for="exampleFormControlInput1"
    class="block text-gray-700 text-sm font-bold mb-2">Name</label>
  <input type="text"
    class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
    id="exampleFormControlInput1" placeholder="Enter Name" wire:model="name">
  @error('name') <span class="text-red-500">{{ $message }}</span>@enderror
</div>
<div class="mb-4">
  <label for="exampleFormControlInput2"
    class="block text-gray-700 text-sm font-bold mb-2">Email:</label>
  <textarea
    class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
    id="exampleFormControlInput2" wire:model="email"
    placeholder="Enter Email"></textarea>
  @error('email') <span class="text-red-500">{{ $message }}</span>@enderror
</div>
<div class="mb-4">
  <label for="exampleFormControlInput2"
    class="block text-gray-700 text-sm font-bold mb-2">Mobile:</label>
  <textarea
    class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
    id="exampleFormControlInput2" wire:model="mobile"
    placeholder="Enter Mobile"></textarea>
  @error('mobile') <span class="text-red-500">{{ $message }}</span>@enderror
</div>
```

Slika 2.18 Obrazac za kreiranje studenta [19]

Poglavlje 2. Tehnologije i primjeri

Unutar pogleda *livewire.create* postoje dva gumba, *Store* čijim pritiskom se poziva *store()* funkcija te *Close* koji poziva *closeModalPopover* funkciju što je vidljivo i na slici 2.19.

```
<div class="bg-gray-50 px-4 py-3 sm:px-6 sm:flex sm:flex-row-reverse">
  <span class="flex w-full rounded-md shadow-sm sm:ml-3 sm:w-auto">
    <button wire:click.prevent="store()" type="button"
      class="inline-flex justify-center w-full rounded-md border border-transparent
      Store
    </button>
  </span>
  <span class="mt-3 flex w-full rounded-md shadow-sm sm:mt-0 sm:w-auto">
    <button wire:click="closeModalPopover()" type="button"
      class="inline-flex justify-center w-full rounded-md border border-gray-300
      Close
    </button>
  </span>
</div>
```

Slika 2.19 Gumbi *Store* i *Close* [19]

2.7.6 Pokretanje servera i prikaz aplikacije

Za pozivanje Laravel razvojnog poslužitelja, *eng.(server)*, koristi se naredba *php artisan serve*. Nakon pokretanja naredbe, u web pregledniku može se pristupiti aplikaciji na idućoj web adresi: *http://127.0.0.1:8000/students* te je sam izgled aplikacije prikazan na slici 2.20.



Slika 2.20 Završni izgled aplikacije [19]

2.8 Bootstrap primjeri

Svi primjeri koji će biti ovdje prikazani preuzeti su sa službene stranice Bootstrapa. Prikazat će se primjeri prijelomnih točaka, kontejnera, mrežnog sustava i gumbi. Primjere za ostale Bootstrap UI komponente moguće je pronaći na službenoj stranici Bootstrapa pod dokumentacijom.

2.8.1 Prijelomne točke, eng. (*breakpoints*)

Bootstrap je najpoznatiji po svojem responzivnom dizajnu, a prijelomne točke su gradivni elementi toga. Unutar Bootstrapa postoji šest zadanih prijelomnih točaka, koje se ponekad nazivaju i razine mreže, eng. (*grid tiers*). Popis prijelomnih točaka i njihovih dimenzija nalazi se u tablici 2.1.

Tablica 2.1 Bootstrap prijelomne točke [25]

Prijelomna točka	Infiks razreda	Dimenzija
X-Small	Ništa	<576px
Small	sm	≥ 576px
Medium	md	≥ 768px
Large	lg	≥ 992px
Extra large	xl	≥ 1200px
Extra extra large	xxl	≥ 1400px

Svaka prijelomna točka odabrana je za udobno držanje spremnika čija je širina višekratnik broja dvanaest. Prijelomne točke također su reprezentativne za podskup uobičajenih veličina uređaja i dimenzija okvira za prikaz. Sami rasponi pružaju snažan i dosljedan temelj za nadogradnju na gotovo svakom uređaju. [25]

2.8.2 Kontejneri

Kontejneri su najosnovniji elementi izgleda u Bootstrapu i potrebni su prilikom korištenja Bootstrapovog zadanog mrežnog sustava. Kontejneri se koriste za sadržavanje, umetanje te ponekad i centriranje sadržaja unutar njih.

Bootstrap dolazi s tri različita kontejnera:

1. *.container*, koji postavlja maksimalnu širinu na svakoj prijelomnoj točki
2. *.container-fluid*, čija je širina uvijek 100% za svaku prijelomnu točku
3. *.container-breakpoint*, čija je širina 100% do specificirane prijelomne točke [26]

Slika 2.21 prikazuje razliku između *.container* i *.container-fluid* klase.

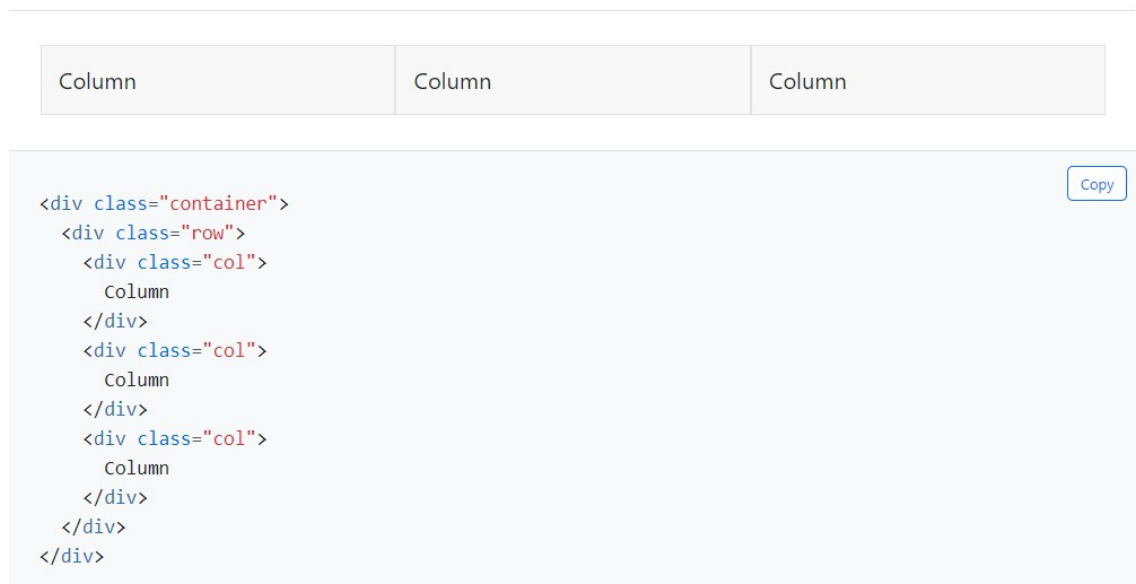


Slika 2.21 Razlika između *.container* i *.container-fluid* klase [27]

2.8.3 Mrežni sustav, *eng. (grid)*

Mrežni sustav koristi niz kontejnera, redaka i stupaca za postavljanje i poravnavanje sadržaja. Najosnovniji primjer Bootstrapovog mrežnog sustava prikazan je na slici 2.22 gdje su tri stupca jednake širine za sve veličine zaslona.

Gore prikazani primjer je moguć pošto Bootstrapov mrežni sustav funkcionira na principu da unutar svakog reda je moguće smjestiti dvanaest stupaca, stoga tri stupca jednakih širina nije ni najmanji problem za implementirati unutar jednog reda. [28]



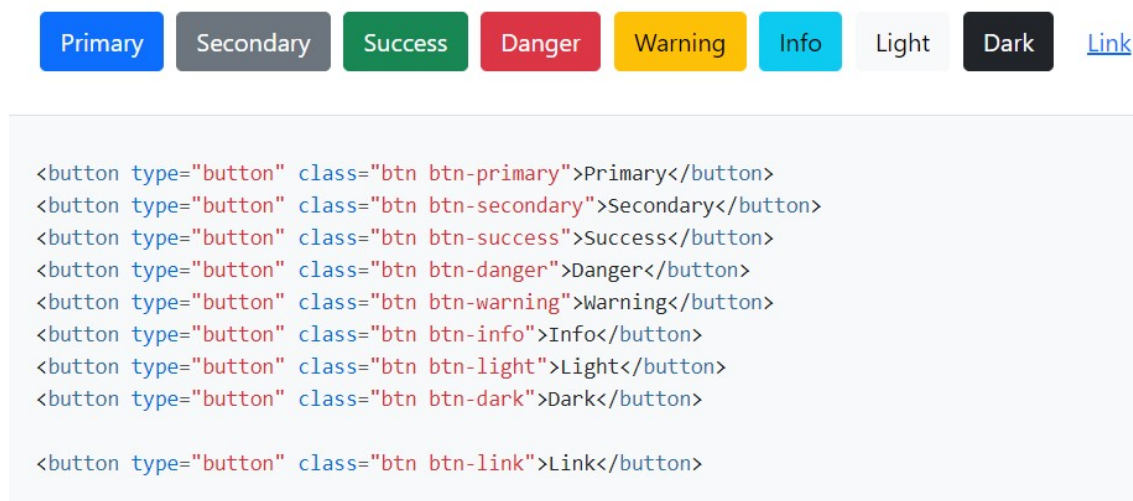
Slika 2.22 Osnovni Bootstrapov mrežni sustav [28]

Neka od pravila prilikom izrade Bootstrapovog mrežnog sustava su:

1. Redovi moraju biti smješteni unutar kontejnera s klasom `.container` ili `.container-fluid` za pravilno poravnanje i ispunjavanje *eng. (padding)*
2. Redovi se koriste za izradu horizontalnih grupa stupaca
3. Sadržaj bi trebao biti smješten unutar stupaca i samo stupci mogu biti neposredni potomci redaka
4. Mrežni stupci stvaraju se navođenjem broja od 12 dostupnih stupaca koje se želi obuhvatiti. Na primjer, tri jednaka stupca koristila bi tri `.col-sm-4` klase.
5. Širine stupaca su u postocima, tako da se uvijek mogu skalirati u odnosu na veličinu roditeljskog elementa [29]

2.8.4 Gumbi

Bootstrap je poznat po svojim gumbima tj. jako se lako zamijećuju osnovni Bootstrapovi gumbi. Slika 2.23 prikazuje sve izgledе kao i klase osnovnih gumbi. Još dodatno moguće im je mijenjati veličinu kao i omogućenost tj. onemogućenost pritiska.



Slika 2.23 Osnovni Bootstrapovi gumbi [30]

Poglavlje 3

Opis aplikacije

3.1 Funkcionalnosti

Ulaskom u web aplikaciju *Virtualni hladnjak* vidljiva je početna stranica prikazna na slici 3.1.



Slika 3.1 Izgled početne stranice web aplikacije

Poglavlje 3. Opis aplikacije

Na samoj početnoj stranici, osim objašnjenja što je web aplikacija *Virtualni hladnjak*, nalaze se i dvije poveznice: *Prijava* koja vodi, kako joj ime i govori, na prozor za prijavu; i *Registriraj se* koja otvara prozor za registraciju. Prozor za prijavu moguće je vidjeti na slici 3.2.



Slika 3.2 Prozor za prijavu

Sadržaj prozora za prijavu je uobičajen. Već postojeći korisnici upisuju svoju e-poštu i lozinku, te ukoliko žele, mogu označiti i *Zapamti me* polje kako ne bi trebali upisivati e-poštu i lozinku pri svakom ulasku u web aplikaciju. Osim gumba *Prijava*, koji šalje ispunjene podatke na poslužiteljski dio aplikacije gdje se oni provjeravaju jesu li ispravno upisani te postoji li uneseni korisnik u bazi podataka i preusmjerava korisnika na glavnu stranicu aplikacije, tu se nalazi i poveznica *Zaboravili ste lozinku?* koja vodi na prozor za poništavanje lozinke vidljiv na slici 3.3. Samo poništavanje funkcionira na način da se upiše e-pošta korisnika te se pritiskom na gumb *Veza za poništavanje lozinke pomoću e-pošte* šalje poveznicu za poništavanje i postavljanje nove lozinke na e-poštu korisnika.



Slika 3.3 Prozor za poništavanje lozinke

Ukoliko korisnik još nema napravljeni račun, to može napraviti pomoću registracijskog prozora prikazanog na slici 3.4. Izgled registracijskog prozora je također poprilično uobičajen s poljima za unos imena, e-pošte, lozinke te potvrde lozinke. Pritiskom na gumb *Registriraj se*, svi podaci se provjeravaju ukoliko su ispravno upisani te se zatim spremaju unutar baze podataka, a sam korisnik je preusmjeren na glavnu stranicu aplikacije. Pored gumba *Registriraj se* nalazi se i poveznica *Već ste registrirani?* koja vodi na prozor za prijavu.



Ime

Email

Lozinka

Potvrdite lozinku

Već ste registrirani? **Registriraj se**

Slika 3.4 Prozor za registraciju

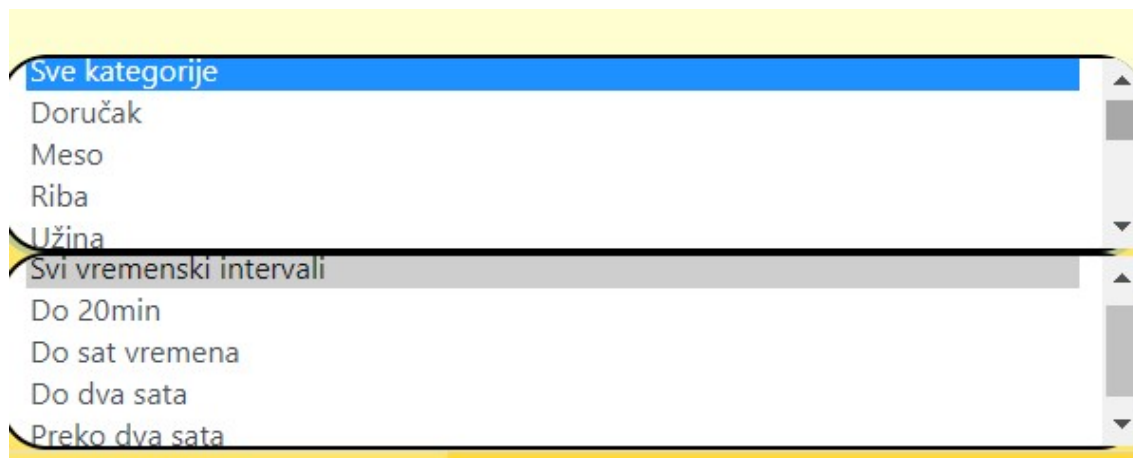
Uspješnom prijavom ili registracijom, kao što je već navedeno, korisnik je preusmjeren na glavnu stranicu web aplikacije. Izgled glavne stranice vidljiv je sa slike 3.5. Na glavnoj stranici prikazani su svi recepti za koje korisnik ima sve potrebne sastojke u dovoljnim količinama.

Poglavlje 3. Opis aplikacije



Slika 3.5 Glavna stranica web aplikacije

Osim samih recepata na stranici su dostupna i dva padajuća izbornika, vidljiva na slici 3.6, koji služe kao filteri za filtriranje recepata na stranici, bilo to po kategoriji recepta ili po vremenskom trajanju.



Slika 3.6 Filteri za kategoriju recepta i vremensko trajanje

Poglavlje 3. Opis aplikacije

Također vidljiva na svim stranicama web aplikacije je navigacijska traka na kojoj se nalaze poveznice: *Recepti*, koja vodi na glavnu stranicu kao i pritisak na logo sliku, *Moji recepti*, koja vodi na stranicu na kojoj se nalaze svi recepti koje je napisao korisnik, *Hladnjak*, koja vodi na stranicu na kojoj se nalaze sve korisnikove namirnice i njihove količine koje je unio u svoj virtualni hladnjak. Na samom kraju navigacijske trake nalazi se padajući izbornik s imenom korisnika koji je vidljiv na slici 3.7. Unutar padajućeg izbornika postoje dvije poveznice: *Odjava*, čijim se pritiskom korisnik odjavljuje sa svojeg računa te je preusmjeren na početnu stranicu, i *Profil* koja vodi na stranicu *Profil* unutar koje se može promijeniti lozinka, urediti korisničko ime i e-pošta, omogućiti autentifikaciju u dva koraka, odjaviti se sa svih ili određenih sesija na drugim uređajima te trajno izbrisati cijeli račun.



Slika 3.7 Padajući izbornik unutar navigacijske trake

Sama aplikacija je responzivna na promjene veličina zaslona te je također napravljena i za zaslone mobilnih uređaja s hamburger izbornikom za navigaciju. Prikaz glavne stranice aplikacije za mobilne uređaje prikazan je na slici 3.8.

Poglavlje 3. Opis aplikacije



Slika 3.8 Prikaz glavne stranice aplikacije za mobilne uređaje

Pritiskom na sliku ili naslov recepta ulazi se unutar detalja samog recepta. Detalji recepta sastoje se od dva dijela. Prvi dio prikazan je na slici 3.9, a sastoji se od zaglavlja unutar kojeg je moguće pronaći naslov recepta, sliku, kategoriju samog recepta, vrijeme potrebno za pripremu, koliko porcija se može pripremiti s izabranim receptom te sve potrebne namirnice kao i njihove količine.

Pileći batak



Kategorija: Meso

Vrijeme izrade: 30min

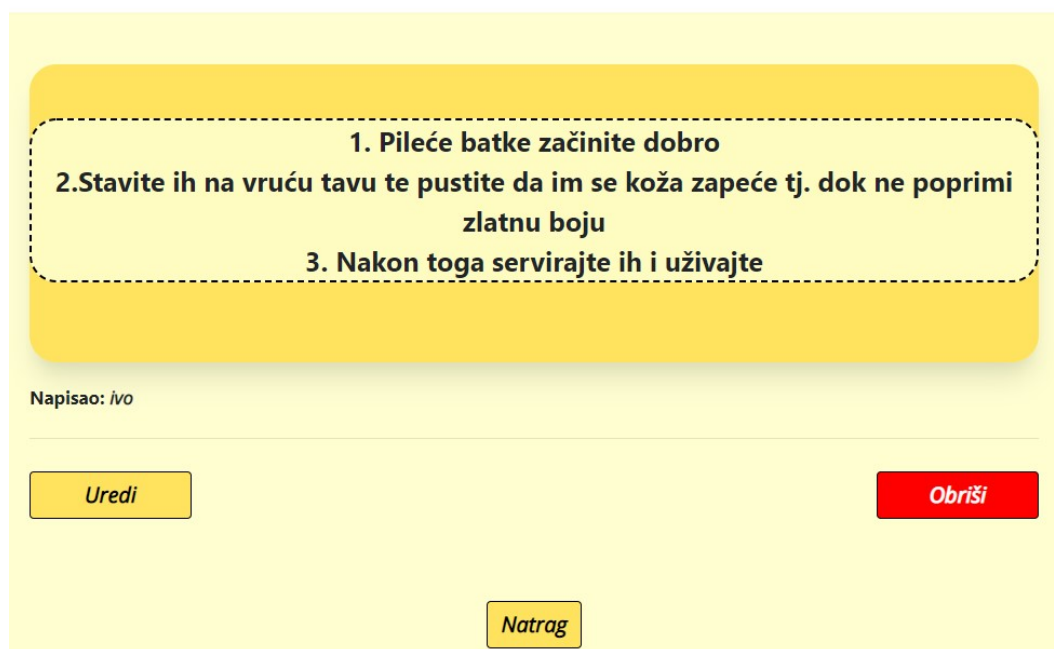
Porcije: 2

Namirnice: sol:3g, papar:2g, pileći batak:400g,

Slika 3.9 Stranica detalja recepta, 1.dio

Drugi dio detalja o receptu prikazan je na slici 3.10. Unutar srednjeg okvira napisani su detaljni koraci kako napraviti odabrani recept. Ispod samog okvira napisano je ime autora recepta te ukoliko je sam autor recepta ušao u svoj recept biti će mu vidljivi gumbi *Uredi*, koji vodi na stranicu za uređivanje recepta, i *Obriši*, koji briše recept iz baze podataka. Svim korisnicima je vidljiv gumb *Natrag* koji ih vraća na stranicu s njihovim receptima.

Poglavlje 3. Opis aplikacije



Slika 3.10 Stranica detalja recepta, 2.dio

Kao što je već spomenuto, ukoliko autor recepta uđe u vlastiti recept, prikazani će mu biti gumbi *Uredi* i *Obriši*. Pritiskom na gumb *Uredi*, korisnik je preusmjeren na stranicu za uređivanje recepta čiji je prvi dio vidljiv na slici 3.11. Sam obrazac za uređivanje i kreiranje recepta je isti, no prilikom odabira za uređivanje odabranog recepta, većina polja je **popunjeno** s podacima iz tablice *probas* gdje se spremaju informacije o svakom receptu. **Stoga** u prvom dijelu stranice popunjene vrijednosti su *Naslov*, *Kategorija*, *Vrijeme i vremenska jedinica* **te** *broj Porcija*. Namirnice je potrebno nanovo odabrati kao i upisati njihove količine i mjerne jedinice što je **jedna od mana ovog sistema.**

Uredi recept: Pileći batak

The form is titled "Uredi recept: Pileći batak". It contains the following fields:

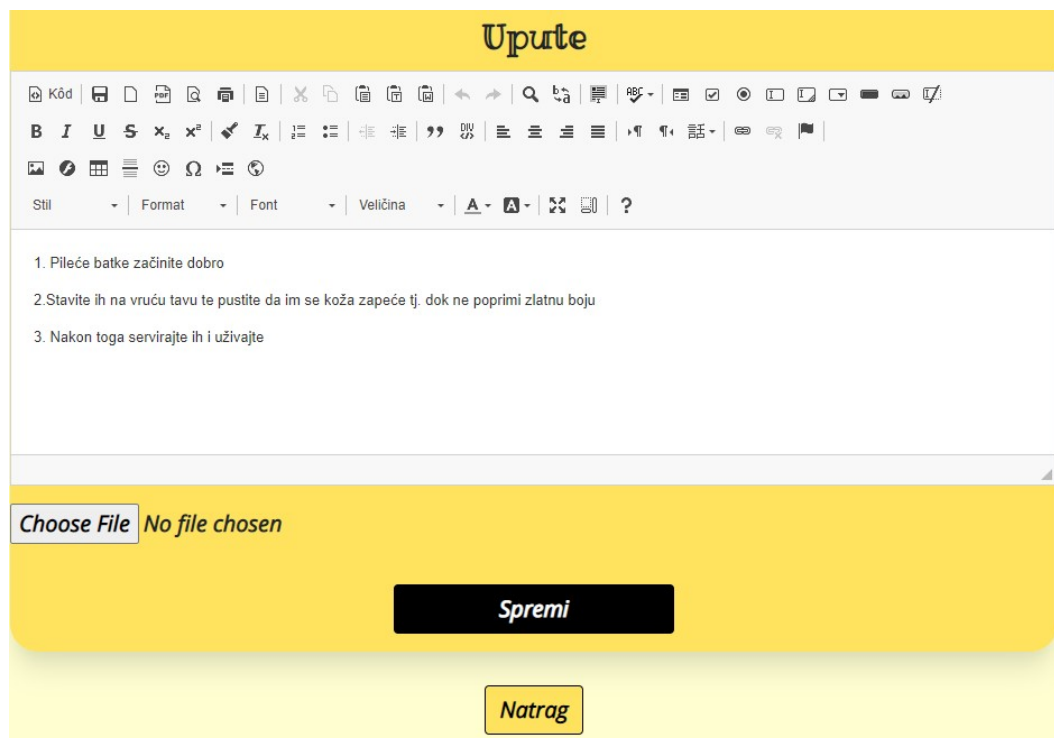
- Naslov**: A text input field containing "Pileći batak".
- Kategorija**: A dropdown menu with "Meso" selected.
- Vrijeme**: A text input field containing "30" and a dropdown menu with "min" selected.
- Porcija**: A text input field containing "2".
- Namirnice**: A section with four rows of ingredients, each with a checkbox, a label, a quantity input field, and a unit dropdown menu.
 - Brašno:** ☐ *glatko brašno* | Količina: | ---
 - Jaja:** ☐ *kokošja jaja* | Količina: | ---
 - Masti, Ulja I Ocat:** ☐ *maslinovo ulje* | Količina: | ---
 - ☐ *suncokretovo ulje* | Količina: | ---

Slika 3.11 Stranica *Uredi recept*, 1.dio

Drugi dio stranice *Uredi recept* sastoji se od tekstualnog okvira unutar kojeg su već upisani koraci pripreme recepta te unutar tog tekstualnog okvira moguće je urediti i sam tekst jer je riječ o *CKEditoru*, bogatom JavaScript uređivaču teksta [31]. Ispod tekstualnog okvira nalazi se gumb *Choose File* pomoću kojega se može učitati fotografija s računala na web stranicu te je potrebno pri svakom uređivanju recepta nanovo učitati sliku pošto ona ne bude učitana prilikom uređivanja nego se samo prebriše unutar baze podataka ukoliko se spremi bez nje. Na samom kraju nalazi se gumb *Spremi* kojim se šalju sve informacije unutar obrasca na poslužiteljsku stranu gdje se one validiraju, a zatim spremaju promjene u tablicu *probas*. Također ukoliko je uređivanje recepta prošlo bez pogreške, korisnik će biti preusmjeren na *Moji recepti*

Poglavlje 3. Opis aplikacije

stranicu te će se pojaviti poruka "Ažuriran recept!". Gumb *Natrag* vraća korisnika na stranicu *Moji recepti*.



Slika 3.12 Stranica *Uredi recept*, 2.dio

Da bi se pristupilo obrascu za kreiranje recepta, potrebno je stisnuti na gumb *+Dodaj novi recept* koji se nalazi na dnu stranice *Moji recepti*. Kao što je već spomenuto, obrazac za kreiranje novog recepta istog je izgleda kao i obrazac za uređivanje recepta osim što su sva polja prazna te gumb *Spremi*, osim što šalje upisane informacije na poslužiteljsku stranu gdje se one validiraju te se sprema unutar tablice *probas*, sada vraća poruku "Dodan recept!". Ostale funkcionalnosti su iste. Polje *Naslov* je običan okvir za tekst, *Kategorija* je padajući izbornik iz kojega se odabire kategorija recepta. *Vrijeme* je polje s bročanim unosom s korakom od 0.1, moguć je unos decimalnih brojeva, te padajući izbornik s vremenskim jedinicama *min* ili *h* (za sat). *Porcija* je također polje s bročanim unosom, koraka 1 što govori da je moguć upis samo prirodnih brojeva. Sekcija *Namirnice* sastoji se od imena kategorija namirnica,

Poglavlje 3. Opis aplikacije

imena namirnica pored kojih se nalazi potvrdni okvir, brojčanog polja s korakom od 0.1, za decimalni unos količine namirnica, te padajući izbornik s mjernim jedinicama za količine *g*, *kg*, *mL* i *L*. U zadnjem dijelu obrasca nalaze se prazan tekstualni okvir, *Upute*, s tekstualnim uređivačem, gumb za učitavanje slike kao i gumbi za spremiti recept te *Natrag* za povratak na stranicu *Moji recepti*.

Zadnja poveznica unutar navigacijske trake, *Hladnjak*, vodi do stranice gdje su prikazane sve korisnikove namirnice kao i njihove količine, što je vidljivo na slici 3.13. Osim samih namirnica, na stranici se nalazi i gumb *+Dodaj novu namirnicu* kojim se pristupa obrascu za kreiranje nove namirnice.



Recepti Moji recepti Hladnjak iVO ▾

Hladnjak

kokošja jaja: 10kom

pileći batak: 500g

ABC sir: 100g

kruške: 1kg

papar: 2.01kg

sol: 1.11kg

+ Dodaj novu namirnicu

Slika 3.13 Stranica *Hladnjak*

Poglavlje 3. Opis aplikacije

Nakon pritiska na gumb *+Dodaj novu namirnicu* korisnik je preusmjeren na stranicu za dodavanje nove namirnice na kojoj se nalazi obrazac. Obrazac se sastoji od padajućeg izbornika *Kategorija*, kojim se bira kategorija namirnice, padajućeg izbornika *Namirnica* iz kojeg se odabire željena namirnica te polja za brožčani unos *Količina* kao i padajući izbornik za mjerne jedinice. Na samom dnu nalaze se gumbi *Spremi*, koji šalje sve vrijednosti obrasca na poslužiteljsku stranu gdje se namirnica onda sprema unutar tablice *hladnjaks*, i *Natrag* koji vodi natrag do stranice *Hladnjak*. Sve navedeno prikazano je na slici 3.14.

Nova namirnica



The screenshot shows a mobile application interface for adding a new food item. The form is titled 'Nova namirnica' and is set against a light yellow background. It contains three main sections: 'Kategorija' (Category) with a dropdown menu showing 'Brašno', 'Grickalice', and 'Jaja'; 'Namirnica' (Food item) with a dropdown menu showing 'brašno', 'glatko brašno', 'jaja', and 'kokošja jaja'; and 'Količina' (Quantity) with a text input field and a dropdown menu for 'Mjerna jedinica' (Unit of measurement) showing 'kg', 'g', and 'L'. At the bottom of the form are two buttons: 'Spremi' (Save) in a black box with white text, and 'Natrag' (Back) in a yellow box with black text.

Slika 3.14 Stranica *Hladnjak*

Obrazac za uređivanje informacija namirnica isto je strukturiran kao i gore prikazan i objašnjen obrazac za dodavanje nove namirnice, samo su sva polja popunjena s vrijednostima koje su upisane unutar tablice *hladnjaks* za tu namirnicu. Pristupa

Poglavlje 3. Opis aplikacije

mu se pritiskom na gumb *Uredi* koji se nalazi na stranici *Informacije o namirnici* prikazanoj na slici 3.15.

Posljednoj stranici pristupa se pritiskom na ime namirnice na stranici *Hladnjak*. Korisnik je zatim preusmjeren na stranicu *Informacije o namirnici* gdje može pogledati u koju kategoriju spada odabrana namirnica, koliku količinu te namirnice ima te ju može urediti pritiskom na gumb *Uredi*, koji ga vodi na obrazac za uređivanje informacija namirnica, gdje može promijeniti npr. iznos količine. Također može i obrisati odabranu namirnicu iz svog virtualnog hladnjaka pritiskom na gumb *Obriši*. Gumb *Natrag* vodi korisnika natrag na stranicu *Hladnjak* gdje može vidjeti sve svoje namirnice.

pileći batak



Slika 3.15 Stranica *Informacije o namirnici*

Poglavlje 4

Opis funkcionalnosti - Dodavanje novog recepta

4.1 Opis pogleda *proba_create*

Za dodavanje novog recepta, korisnik pristupa obrascu za dodavanje novog recepta putem gumba *+Dodaj novi recept*, koji se nalazi na stranici *Moji recepti* te čiji kod je vidljiv na slici 4.1, koji ga šalje na stranicu */proba/create* gdje se nalazi spomenuti obrazac.

```
<div style="text-align:center">  
  <a href="/proba/create" class="btn btn-primary btn-rounded" style = 'background-color:black;  
    border-color:black; color:white; font-family:"Open Sans"; font-weight:bold; text-align:center;  
    width:250px; font-size:20px'>+ Dodaj novi recept</a>  
</div>
```

Slika 4.1 Sintaksa za gumb *+Dodaj novi recept*

Unutar *proba_create.blade.php* pogleda, na koji je korisnik preusmjeren, nalazi se obrazac za kreiranje novog recepta.

Za kreiranje obrasca korišten je Laravel Collective paket [32]. Samo otvaranje obrasca prikazano je linijom koda na slici 4.2. Za otvaranje Laravel Collective obrazaca potrebna je sintaksa `{{!! Form::open()!!}}`. Unutar metode *open()* upisuju se atributi i

Poglavlje 4. Opis funkcionalnosti - Dodavanje novog recepta

njihove pridodijeljene vrijednosti. U Collective obrascima, umjesto klasičnog HTML-ovog pisanja atributa i dodijeljivanja vrijednosti, npr. *id* = "*form*", vrijednost se pridodijeljuju znakom =>. Kod otvaranja obrasca potrebno je definirati rutu na koju će obrazac, prilikom spremanja unesenih podataka, poslati sve unesene podatke kao i metodu kojom ih šalje. Pošto *store()* funkcija, koja prima poslano podatke iz obrasca, prima podatke *POST* metodom, ona je specificirana prilikom otvaranja. Također kako ova metoda šalje i slike koje su učitane u nju, potrebno je definirati metodu kodiranja podataka prilikom slanja podataka iz obrazaca. Pošto se u formi, kao što je i navedeno, koristi tip podatka *file*, pod koji spadaju i slike, potrebno je koristiti *multipart/form-data* metodu kodiranja.

```
{!! Form::open(['id'=>'form','route' => 'proba.store',  
  'method' => 'POST', 'enctype' => 'multipart/form-data']) !!}
```

Slika 4.2 Linija koda za otvaranje obrasca

Neki od elemenata koji se nalaze u obrascu su labele, okvir za tekst, padajući izbornici, tekstualno područje, polja za numeričke vrijednosti i već spomenuti gumb za učitavanje datoteka tipa *file*. Na slici 4.3 prikazana je sintaksa pisanja labela, okvira za tekst i padajućeg izbornika. Svi elementi obrasca imaju istu sintaksu *{{Form::tip_elementa('ime_elementa', 'vrijednost', [popis atributa i njihovih vrijednosti])}}*. Dodatan atribut koji još nije objašnjen je *placeholder*, a to je opis očekivanog upisa koji je prikazan dok korisnik ne odabere vrijednost ili dok ju ne krene upisivati.

Bitan dio obrasca, kojega korisnik vidi, je popis svih sastojaka koji je poslan putem *ProbaControllera* unutar varijable *\$in*. Sam proces slanja dohvaćenih vrijednosti iz tablice putem kontrolera biti će opisan nakon objašnjenja blade pogleda. Imena kategorija i namirnica, kao i polje za numeričke vrijednosti, čija je sintaksa prikazana na slici 4.4, i padajuća lista s mjernim jedinicama koja nije stala unutar slike, stavljene se unutar tablice radi bolje organizacije.

```
{{Form::label('title', 'Naslov')}}<br/>
{{Form::text('title', '', ['placeholder' => 'Naslov',
    'style' => 'border-radius:25px; border: 2px solid black;'])}}
{{Form::label('rec_category', 'Kategorija')}}
{{Form::select('rec_category',[
    'doručak' => 'Doručak',
    'meso' => 'Meso',
    'riba' => 'Riba',
```

Slika 4.3 Dijelovi obrasca

Navedena polja se stvaraju i imena kategorija i namirnica se ispisuju dinamički pomoću *foreach* petlje koja poslani objekt *\$in*, koji sadržava sve informacije koje se nalaze unutar tablice *ingredients*, razlaže na pojedine objekte tj. uzima jedan red unutar tablice i sprema u varijablu *\$i*. Pomoću informacija jednog reda moguće je ispisati vrijednosti traženog retka kao npr. u varijablu *\$cat* sprema se vrijednost kategorije trenutne namirnice koja se dohvaća pomoću izraza *\$i->category*. Također provjerom ukoliko su kategorija trenutne namirnice i vrijednosti spremljenje u *\$cat* varijabli, sprječava se višestruko ispisivanje kategorija namirnica pomoću *<optgroup>* oznake koja služi za grupaciju srodnih opcija, u ovom slučaju svih namirnica koje spadaju pod istu kategoriju. Za sam ispis vrijednosti "stupca" unutar objekta u HTML-u potrebno je koristiti sljedeću sintaksu, za primjer uzet će se ispis imena namirnice pored potvrdnog okvira: *{{ \$i->food_name }}*. Također kako bi obrazac poslao točno odabrane namirnice potrebno je unutar sintakse za potvrdni okvir dodati dva atributa: *data-id="{{ \$i->id }}"* je atribut podataka koji služi kako bi se na HTML-ov element, u ovom slučaju potvrdni okvir, spremile dodatne informacije te *value="{{ \$i->food_name }}"* kako bi potvrdni okvir poprimio vrijednost odabrane namirnice u obliku njezinog imena.

Poglavlje 4. Opis funkcionalnosti - Dodavanje novog recepta

```
<table name="food_name" id="food_name" style="display:flex; justify-content:center">
  @php
    $cat = null;
  @endphp
  @foreach($in as $i)
    <tr>
      <td>
        @if($i->category != $cat)
          @if($cat != null)
            </optgroup>
          @endif
          @php($cat = $i->category)
          <h2 style="font-size:25px; text-transform:capitalize;">
            <optgroup label = "{{ $i->category }}"></h2>
        @endif
      </td>
      <td>
        <input id= "food_name[]" name="food_name[]" data-id="{{ $i->id }}"
          type="checkbox" value="{{ $i->food_name }}" class="ingredient-enable"
          style = 'border-radius:25px; border: 2px solid black;'>
      </td>
      <td >
        <h3 style="font-size:17px; font-family:'Open Sans'"><b>{{ $i->food_name }}</b> &nbsp;</h3>
      </td>
      <td id="hide" >
        <input name="quantity[]" id="quantity[]" data-id="{{ $i->id }}"
          name="quantity" type="number" class="ingredient-amount dorm-control"
          placeholder="Količina" step="0.01" style = 'border-radius:25px; border: 2px solid black;'>
      </td>
    </tr>
  @endforeach
</table>
```

Slika 4.4 Ispis svih sastojaka iz tablice *ingredients* unutar obrasca

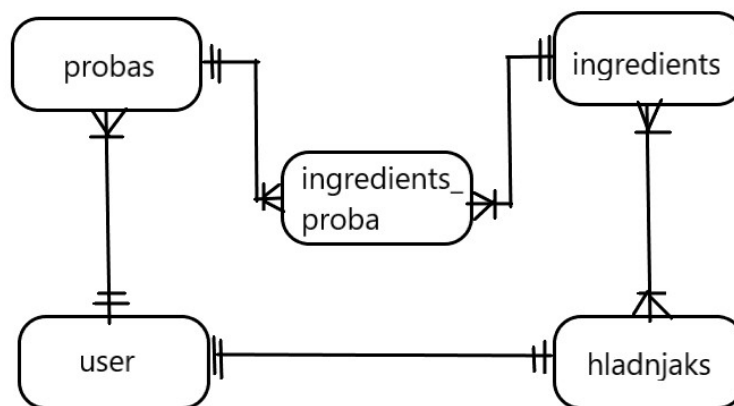
Sintaksa ranije navedenog *Choose file* gumba, čiji je tip *file*, i gumba *Spremi* kao i samo zatvaranje obrasca prikazano je na slici 4.5. Za sam gumb *Spremi* korištena je Bootstrapova klasa *btn btn-primary btn-rounded* koja je dodatno prilagođena temi same aplikacije koristeći *inline CSS* pomoću *style* atributa. Zatvaranje obrasca slično je otvaranju, no umjesto metode *open()* koristi se metoda *close()*.

```
<div class="form-group" style="float:left">
  {{Form::file('cover_image', ['style' => 'font-family:"Open Sans";
    font-weight:bold; width:250px; font-size:20px'])}}
</div><br/><br/>
<br/>
{{Form::submit('Spremi', ['class' => 'btn btn-primary btn-rounded',
  'style' => 'background-color:black; border-color:black; color:white;
  font-family:"Open Sans"; font-weight:bold; width:250px; font-size:20px;'])}}
{!! Form::close() !!}
</div>
```

Slika 4.5 Gumbi *Choose file* i *Spremi* te zatvaranje obrasca

4.2 ER dijagram i modeli

Kako bi se lakše objasnio sam mehanizam spremanja podataka u tablice potrebno je prikazati veze među tablicama. Te veze prikazane su na slici 4.6.



Slika 4.6 ER dijagram tablica aplikacije

Prije daljnjeg objašnjavanja potrebno je pojasniti što se sprema u koju tablicu. Unutar tablice *probas* spremaju se sve informacije o receptu kao npr. naslov recepta, autor, upute za pripremu recepta, slika recepta, itd. Tablica *user* sadrži sve informacije o registriranim korisnicima. *Ingredients* tablica služi kao spremište za sve namirnice i njihove kategorije, a unutar *hladnjaks* tablice spremaju se informacije o odabranim namirnicama, njihovim količinama te kojem korisniku one pripadaju.

Nakon kratkog pojašnjenja čemu koja tablica služi, mogu se objasniti veze između njih te prikazati kako je to implementirano u kodu unutar modela.

Objašnjavanje veza započet će od veze između tablica *probas* i *user*. Pošto svaki korisnik može biti autor više recepata, a svaki recept može imati točno jednog autora,

Poglavlje 4. Opis funkcionalnosti - Dodavanje novog recepta

veza između te dvije tablice je jedan-na-više. Implementacije te veze kodom unutar *User* modela prikazana je na slici 4.7, odnosno unutar *Proba* modela na slici 4.8. Sama funkcija *probas* automatski će odrediti strani ključ za *Proba* model te nakon definicije veze moguće je funkcijom *probas* pristupiti kolekciji recepata koje je taj korisnik napisao.

```
public function probas(){  
    return $this->belongsToMany(Proba::class);  
}
```

Slika 4.7 Veza jedan-na-više na strani modela *User*

Unutar modela *Proba* potrebno je dodatno definirati redove unutar tablice koji se ne popunjavaju ručno već iz sesije s informacijama trenutno prijavljenog korisnika. Metoda *belongsTo* je inverzna veza metodi *belongsToMany* te pomoću nje tj. funkcije *user()* možemo pristupiti svim potrebnim informacijama korisnika.

```
protected $fillable = ['title', 'author', 'author_id'];  
  
public function user(){  
    return $this->belongsTo(User::class);  
}
```

Slika 4.8 Veza jedan-na-više na strani modela *Proba*

Iduća veza je između tablica *user* i *hladnjaks*. Pošto svaki hladnjak pripada točno jednom korisniku te svaki korisnik može imati samo jedan hladnjak, veza između ovih tablica je jedan-na-jedan. Implementacija ove veze unutar modela *User* prikazana je na slici 4.9. Pomoću funkcije *hladnjak()* moguće je dohvatiti hladnjak korisnika.

```
public function hladnjak(){  
    return $this->hasOne(Hladnjak::class);  
}
```

Slika 4.9 Veza jedan-na-jedan na strani modela *User*

Inverzna veza metodi *hasOne* je metoda *belongsTo* koja se nalazi u modelu *Hladnjak* unutar funkcije *user()* s kojom je moguće dohvatiti korisnika te sve njegove informacije. Ta implementacija prikazana je na slici 4.10.

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

Slika 4.10 Veza jedan-na-jedan na strani modela *Hladnjak*

Najteža veza za implementirati je veza između tablica *probas* i *ingredients* koja je više-na-više. Iz tog razloga bilo je potrebno uvesti novu tablicu koja će rastaviti tu vezu na dvije jedan-na-više veze. Takva tablica u Laravelu se naziva *pivot* tablica te njezino ime mora biti spojeno ime dviju tablica između kojih se nalazi veza više-na-više u alfabetskom poretku, tako je nastala tablica *ingredients_proba*. Sam sadržaj tablice *ingredients_proba* te kako se on definira unutar Laravel migracija prikazan je na slici 4.11. Potrebno je primarne ključeve tablica *probas* i *ingredients* pretvoriti u strane ključeve unutar *ingredients_proba* tablice te njihov skup proglasiti primarnim ključem tablice *ingredients_proba*.

Poglavlje 4. Opis funkcionalnosti - Dodavanje novog recepta

```
*/
public function up()
{
    Schema::create('ingredients_proba', function (Blueprint $table) {
        $table->unsignedBigInteger('ingredients_id')->index();
        $table->foreign('ingredients_id')->references('id')->on('ingredients')->onDelete('cascade');
        $table->unsignedBigInteger('proba_id')->unsigned()->index();
        $table->foreign('proba_id')->references('id')->on('probas')->onDelete('cascade');
        $table->primary(['ingredients_id', 'proba_id']);
        $table->float('quantity', 8, 2);
        $table->string('measurement_unit');
        $table->timestamps();
    });
}
```

Slika 4.11 Kreiranje tablice *ingredients_proba* unutar Laravel migracija

Pivot tablica može imati i dodatne stupce, no potrebno ih je definirati prilikom implementacije veza unutar modela. Dodatni stupci definiraju se metodom *withPivot()* što je vidljivo iz slika 4.12 i 4.13. Veza više-na-više definira se metodom *belongsToMany* na obje strane. Unutar navedene metode prvo se specificira s kojom klasom se stvara navedena veza, zatim ide ime pivot tablice, nakon toga ime stranog ključa modela unutar kojeg definiramo vezu i na kraju strani ključ modela s kojim spajamo trenutni model pomoću zadane veze.

```
public function probas(){
    return $this->belongsToMany(Proba::class, 'ingredients_proba',
        'ingredients_id', 'proba_id')
        ->withPivot(['quantity', 'measurement_unit']);
}
```

Slika 4.12 Veza više-na-više unutar *Ingredients* modela

Funkcijom *ingredients*, koja je prikazana na slici 4.13, moguće je pristupiti pivot tablici te njezinim stupcima definirajući unutar *using()* metode korištenje pivot modela *IngredientsProba*.

Poglavlje 4. Opis funkcionalnosti - Dodavanje novog recepta

```
public function ingredients(){  
    return $this->belongsToMany(Ingredients::class, 'ingredients_proba',  
        'proba_id', 'ingredients_id')->withPivot(['quantity', 'measurement_unit'])  
        ->using(IngredientsProba::class);  
}
```

Slika 4.13 Veza više-na-više unutar *Proba* modela

Unutar modela pivot tablice potrebno je definirati sve stupce koji će se popunjavati, kao i primarni ključ. Osim navedenog definirana je i funkcija *food* kojom se iz pivot tablice može pristupiti *ingredients* tablici kako bi se dohvatila imena namirnica i njihova kategorija. Slika 4.14 prikazuje gore navedene informacije.

```
class IngredientsProba extends Pivot{  
    |  
    |  
    protected $fillable = ['ingredients_id', 'proba_id', 'quantity', 'measurement_unit'];  
    protected $primaryKey = ['ingredients_id', 'proba_id'];  
    public function food(){  
        return $this->belongsTo(Ingredients::class, 'ingredients_id');  
    }  
}
```

Slika 4.14 Pivot model *IngredientsProba*

4.3 Opis funkcija *create()* i *store()*

Objekti u naslovu navedene funkcije nalaze se unutar *ProbaControllera*. Prvo će biti opisana funkcija *create()* jer je kraća i poprilično jasna funkcija. Na slici 4.15 prikazana je funkcija *create()*. Unutar funkcije u varijablu *\$ing* spremaju se svi recepti koji postoje te u varijablu *\$in* se spremaju sve namirnice iz tablice *ingredients* **poredani** po kategorijama, a zatim po imenu abecednim redom. Na kraju funkcija vraća pogled *proba_create* u koji su još poslane i varijable *\$ing* i *\$in* kako bi se mogle koristiti unutar pogleda, npr. (varijabla *\$in* koristila se za ispis svih namirnica unutar *foreach* petlje).

```
public function create()
{
    $ing = Proba::all();
    $in = Ingredients::orderBy('category', 'asc')
        ->orderBy('food_name', 'asc')->get();
    return view('proba.proba_create')
        ->with('ing', $ing)->with('in', $in);
}
```

Slika 4.15 Funkcija *create()*

Funkcija *store()* biti će opisana u pet dijelova radi lakšeg razumijevanja rada funkcije.

Prvi dio funkcije sačinjava se od validacije unosa koji je prikazan na slici 4.16. U varijablu *\$val* spremaju se validirane vrijednosti koje su poslane funkciji *store()* prilikom pritiska na gumb *Spremi* nakon završetka popunjavanja obrasca za kreiranje novog recepta unutar pogleda *proba_create*. **Validacija većine polja je samo provjera ukoliko su unesene vrijednosti jer za skoro svako polje je potrebno unijeti vrijednost inače se podaci neće spremati.** Za polja *time* i *quantity* definirane su još i brojčane vrijednosti unutar kojih korisnik može unijeti vrijednost u navedena polja. **Također** polje *cover_image* može ostati prazno (vrijednost *nullable*) jer nije definirano da je potrebno učitati sliku, no ukoliko se učita **slika** maksimalna veličina joj može biti 2

MB.

```
public function store(StoreRecipeRequest $request)
{
    $val = $this->validate($request, [
        'title' => 'required',
        'rec_category' => 'required',
        'time' => 'required|between:0,99.99',
        'measurement' => 'required',
        'servings' => 'required',
        'food_name' => 'required',
        'quantity' => 'required|between:0,99.99|nullable',
        'measurement_unit' => 'required',
        'instructions' => 'required',
        'cover_image' => 'image|nullable|max:1999',
    ]);
}
```

Slika 4.16 Funkcija *store()*, prvi dio - validacija

Drugi dio funkcije prikazan je na slici 4.17. Pošto se za sve namirnice, koje se nalaze unutar obrasca za kreiranje novog recepta, bile one odabrane ili ne, se spremaju vrijednosti količine i mjerne jedinice bilo je potrebno počistiti podatke. Definirana su dva polja: *\$v1* unutar kojeg su spremene sve vrijednosti količina koje nisu *null* i *\$v2* unutar kojeg su spremene sve vrijednosti mjernih jedinica koje su različite od '---'.

```
$v1 = [];  
foreach($val['quantity'] as $v){  
    if($v !== null){  
        array_push($v1, $v);  
    }  
}  
  
$v2 = [];  
foreach($val['measurement_unit'] as $v){  
    if($v !== '---'){  
        array_push($v2, $v);  
    }  
}
```

Slika 4.17 Funkcija *store()*, drugi dio - čišćenje podataka

Treći dio funkcije rezerviran je za spremanje slika. Sam proces spremanja slika u tablicu nije toliko jednostavan kao npr. spremanje imena. Prvo je potrebno provjeriti je li korisnik uopće učitao sliku te **ukoliko je** prvo se unutar varijable *\$filenameWithExt* sprema slika s originalnim imenom i ekstenzijom. Zatim se izvlači ime slike pomoću metode *pathinfo()* koja prima putanju i zastavicu koja određuje što će biti vraćeno. U ovom slučaju vratiti će se samo ime slike. Nakon toga potrebno je izvući ekstenziju slike koja se dobiva korištenjem metode *getClientOriginalExtension()* na učitanoj slici. Potom je stvoreno novo ime slike, kako bi se izbjegli duplikati unutar tablice *probas*, spajanjem vrijednosti varijabli *\$filename* i *\$extension* stavljajući između njih vrijednost metode *time()*. Na kraju je potrebno definirati putanju na koju će se učitana slika spremiti te ju tamo i spremiti. Kako bi se slike prikazivale na web pregledniku, potrebno je da su spremljene unutar *public* mape. Kod za opisani proces prikazan je na slici 4.18.

Poglavlje 4. Opis funkcionalnosti - Dodavanje novog recepta

```
if($request->hasFile('cover_image')){
    $filenameWithExt = $request->file('cover_image')
    ->getClientOriginalName();

    $filename = pathinfo($filenameWithExt, PATHINFO_FILENAME);

    $extension = $request->file('cover_image')
    ->getClientOriginalExtension();

    $fileNameToStore = $filename.'_'.time().'.'.$extension;

    $path = $request->file('cover_image')
    ->storeAs('public/cover_images', $fileNameToStore);
}else{
    $fileNameToStore = 'noimage.jpg';
}
```

Slika 4.18 Funkcija *store()*, treći dio - spremanje slika

Četvrti dio funkcije sastoji se od koda za spremanje recepta u tablicu *probas* kao što prikazuje slika 4.19. Prvo se definira novi objekt *Proba* unutar varijable *\$recept*. Zatim se svaki stupac *probas* tablice ispunjava s potrebnim vrijednostima, bile one uzete direktno iz zahtjeva ili dohvaćene korištenjem dodatnih metoda, npr. *auth()->user()->name* metoda dohvaća ime trenutno prijavljenog korisnika te ga sprema pod *author* stupac u tablici. Da bi se novo učitane vrijednosti spremile u tablicu potrebno je pozvati metodu *save()* na varijablu *\$recept*. Također kako će se unutar ove funkcije spremati i podaci u pivot tablicu potrebno je prvo spremiti recept pomoću kojega će se moći pristupiti *id* vrijednosti samog recepta jer je ona strani ključ te dio skupa primarnog ključa pivot tablice.

Poglavlje 4. Opis funkcionalnosti - Dodavanje novog recepta

```
$recept = new Proba;
$recept->title = $request->input('title');
$recept->author = auth()->user()->name;
$recept->author_id = auth()->user()->id;
$recept->rec_category = $request->input('rec_category');
$recept->instructions = $request->input('instructions');
$recept->time = $request->input('time');
$recept->measurement = $request->input('measurement');
$recept->servings = $request->input('servings');
$recept->cover_image = $fileNameToStore;
$recept->save();
```

Slika 4.19 Funkcija `store()`, četvrti dio - spremanje recepta

Zadnji, peti dio, funkcije prikazuje spremanje izabраниh namirnica kao i njihovih upisanih količina i mjernih jedinica u pivot tablicu *ingredients_proba*. Potrebno je bilo definirati pomoćnu varijablu *\$i* kako bi bilo omogućeno iteriranje po *\$v1* i *\$v2* poljima unutar kojih su spremljene vrijednosti za količinu tj. mjernu jedinicu. Koristeći *foreach* petlju po polju *'food_name'* moguće je za svaku posebnu namirnicu izvući sve potrebne podatke. Unutar varijable *\$hrana* sprema se prvi red tablice *ingredients* kod kojeg se podudara ime namirnice s trenutnom namirnicom koja se nalazi unutar varijable *\$v*. Zatim se definira novi objekt tipa *IngredientsProba* unutar kojeg će biti spremljeni svi potrebni podaci kako bi se popunio cijeli red *ingredients_proba* tablice. Opisani postupak prikazan je na slici 4.20.

Poglavlje 4. Opis funkcionalnosti - Dodavanje novog recepta

```
$i = 0;
foreach($val['food_name'] as $v){
    $hrana = Ingredients::where('food_name', '=', $v)->first();
    $novo = new IngredientsProba;
    $novo->ingredients_id = $hrana->id;
    $novo->proba_id = $recept->id;
    $novo->quantity = $v1[$i];
    $novo->measurement_unit = "$v2[$i]";
    $novo->save();
    $i++;
}

return redirect('/proba')->with('success', 'Dodan recept!');
```

Slika 4.20 Funkcija *store()*, peti dio - spremanje u pivot tablicu

Na samom kraju funkcija *store()* preusmjerava korisnika na stranicu *Moji recepti* gdje, ukoliko je sve uspješno spremljeno, se pojavljuje poruka *'Dodan recept!'*

Poglavlje 5

Zaključak

Aplikacija *Virtualni hladnjak* namijenjena je svima, a pogotovo ljudima koji su, ili neiskusni u kuhanju pa im je potrebna pomoć, ili svima onima koji jednostavno nemaju nikakve ideje što bi danas mogli napraviti za jest ili kako iskoristiti određene namirnice.

Sami temelji aplikacije za njezinu namijenjenu uporabu su napravljeni, no uvijek ima mjesta za implementaciju dodatnih funkcionalnosti kao npr. ocjenjivanje recepata te preporuka recepata za koje nedostaju možda jedna ili dvije namirnice.

Sve korištene tehnologije poprilično su jednostavne za upotrebu i iako su neke od njih potpuno nove (*Laravel 8*, *Jetstream* i *Livewire* su objavljeni u rujnu 2020. godine) na Internetu je moguće pronaći velik broj dokumentacije koja će olakšati samu izradu web aplikacije te je većina problema dokumentirano i riješeno na *Stack Overflowu* stoga razlog za brigu nema.

Dobra strana svih korištenih tehnologija je ta što u kratkom vremenu i s minimalnim naporom moguće je stvoriti potpune web aplikacije koje sadrže osnovne funkcionalnosti (CRUD metode), imaju elegantano i uniformirano korisničko sučelje te se učinkovito renderiraju na uređajima različitih veličina zaslona.

Bibliografija

- [1] Laravel Wikipedia. , s Interneta, <https://en.wikipedia.org/wiki/Laravel> , rujan 2021.
- [2] Laravel. , s Interneta, <https://laravel.com/> , srpanj 2021.
- [3] Laravel Jetstream. , s Interneta, <https://jetstream.laravel.com/2.x/introduction.html> , srpanj 2021.
- [4] Livewire. , s Interneta, <https://jetstream.laravel.com/2.x/stacks/livewire.html> , srpanj 2021.
- [5] Bootstrap. , s Interneta, <https://getbootstrap.com/> , kolovoz 2021.
- [6] MVC Wikipedia. , s Interneta, <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller> , rujan 2021.
- [7] MVC Architecture. , s Interneta, <https://medium.com/@emirveledar/mvc-architecture-819c5918f160> , rujan 2021.
- [8] Tailwind CSS vs. Bootstrap: Is it time to ditch UI kits? , s Interneta, <https://blog.logrocket.com/tailwind-css-vs-bootstrap-ui-kits/> , rujan 2021.
- [9] Jetstream concept overview. , s Interneta, <https://jetstream.laravel.com/2.x/concept-overview.html> , rujan 2021.
- [10] Blade Templates. , s Interneta, <https://laravel.com/docs/8.x/blade> , rujan 2021.
- [11] Livewire Making Components. , s Interneta, <https://laravel-livewire.com/docs/2.x/making-components> , rujan 2021.
- [12] Livewire Properties. , s Interneta, <https://laravel-livewire.com/docs/2.x/properties> , rujan 2021.

Bibliografija

- [13] Livewire Rendering Components. , s Interneta, <https://laravel-livewire.com/docs/2.x/rendering-components> , rujan 2021.
- [14] Laravel Homestead. , s Interneta, <https://laravel.com/docs/8.x/homestead> , rujan 2021.
- [15] Vagrant Tutorial: Everything a Beginner Needs To Know. , s Interneta, <https://phoenixnap.com/kb/vagrant-beginner-tutorial> , rujan 2021.
- [16] MariaDB. , s Interneta, <https://searchdatamanagement.techtarget.com/definition/MariaDB> , rujan 2021.
- [17] What is Bootstrap? A Short Bootstrap Tutorial on the What, Why, and How. , s Interneta, <https://www.toptal.com/front-end/what-is-bootstrap-a-short-tutorial-on-the-what-why-and-how> , rujan 2021.
- [18] Bootstrap Introduction. , s Interneta, <https://getbootstrap.com/docs/5.1/getting-started/introduction/> , rujan 2021.
- [19] Build Laravel Livewire CRUD Application using Jetstream. , s Interneta, <https://www.positronx.io/build-laravel-livewire-crud-application-using-jetstream/> , rujan 2021.
- [20] Download Php. , s Interneta, <https://www.php.net/downloads.php> , rujan 2021.
- [21] Download VirtualBox. , s Interneta, <https://www.virtualbox.org/wiki/Downloads> , rujan 2021.
- [22] Download Vagrant. , s Interneta, <https://www.vagrantup.com/downloads> , rujan 2021.
- [23] Download Git. , s Interneta, <https://git-scm.com/downloads> , rujan 2021.
- [24] Setting up Laravel 8.x with JetStream Auth. , s Interneta, <https://blog.devgenius.io/setting-up-laravel-8-x-with-jetstream-auth-84bbeafc0cd3> , rujan 2021.
- [25] Bootstrap breakpoints. , s Interneta, <https://getbootstrap.com/docs/5.1/layout/breakpoints/> , rujan 2021.
- [26] Bootstrap Containers. , s Interneta, <https://getbootstrap.com/docs/5.1/layout/containers/> , rujan 2021.
- [27] Bootstrap containers - W3Schools. , s Interneta, https://www.w3schools.com/bootstrap4/bootstrap_containers.asp , rujan 2021.

Bibliografija

- [28] Bootstrap Grid. , s Interneta, <https://getbootstrap.com/docs/5.1/layout/grid/> , rujan 2021.
- [29] Bootstrap Grid - W3Schools. , s Interneta, https://www.w3schools.com/bootstrap/bootstrap_grid_system.asp , rujan 2021.
- [30] Bootstrap Buttons. , s Interneta, <https://getbootstrap.com/docs/5.1/components/buttons/> , rujan 2021.
- [31] CKEditor. , s Interneta, <https://ckeditor.com> , rujan 2021.
- [32] Laravel Collective. , s Interneta, <https://laravelcollective.com> , rujan 2021.

Sažetak

Jedan od učestalih problema ljudske svakodnevice je konstantno razmišljanje što danas kuhati ili kako iskoristiti namirnice koje su dostupne kod kuće. U ovom radu predstavljena je web aplikacija *Virtualni hladnjak* čiji je glavni zadatak riješiti upravo te probleme. Neke od funkcionalnosti web aplikacije su skladištenje korisnikovih namirnica te pomoću istih uspješno preporučiti recepte koje je moguće napraviti odmah. Web aplikacija napravljena je u *Laravel 8* i *Laravel Jetstream* tehnologijama uz primjenu ostalih opisanih tehnologija koje se nalaze u samom radu. Sve funkcionalnosti web aplikacije također su prikazane vizualno uz detaljne opise, te je potpuno opisana funkcionalnost kreiranja novog recepta na razini koda.

Ključne riječi — Virtualni hladnjak, Laravel 8, Laravel Jetstream

Abstract

One of the most recurring problems in our day-to-day life is constantly thinking about what to cook, or how to use ingredients that are available at home, in the best way possible. This thesis presents *Virtual refrigerator*, a web application whose main task is to solve previously mentioned problems. Some of the web application's functionalities are storing user's food items and the ability to recommend recipes that can be made now, with the already stored ingredients. The web application is written in *Laravel 8* and *Laravel Jetstream* technologies with the usage of other technologies that are mentioned inside this thesis. All web application's functionalities are also presented visually with detailed descriptions, while the functionality of creating a new recipe is completely described on a code level.

Keywords — Virtual refrigerator, Laravel 8, Laravel Jetstream