

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Računarstvo

ANAMARIJA PAPIĆ

ZAVRŠNI RAD

IZRADA PLATFORME ZA TIMSKI RAD

Split, lipanj 2023.

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Računarstvo

Predmet: Odabrani alati i naredbe u Linuxu

Z A V R Š N I R A D

Kandidat: Anamarija Papić

Naslov rada: Izrada platforme za timski rad

Mentor: Nikola Grgić, viši predavač

Split, lipanj 2023.

Sadržaj

Sažetak	1
1 Uvod	2
2 Korištene tehnologije	3
2.1 Laravel	3
2.2 Docker spremnici kao razvojno okruženje	4
2.2.1 Nginx	4
2.2.2 PHP	4
2.2.3 MySQL	5
2.2.4 phpMyAdmin	5
2.3 Upravitelji paketima i ovisnostima	5
2.3.1 npm	5
2.3.2 Composer	6
2.4 Razvojna okruženja za testiranje	6
3 Opis implementacije praktičnog rada	7
3.1 Instalacija i konfiguracija okruženja	7
3.1.1 Početni komplet Laravel Jetstream	7
3.1.2 .env datoteka	8
3.2 Struktura Laravel aplikacije	9
3.2.1 Struktura početnog direktorija s izvornim kôdom aplikacije	9
3.2.2 Struktura app direktorija	10
3.3 Struktura relacijske baze podataka	11
3.4 Interakcija s bazom podataka	12
3.4.1 Migracije	12
3.4.2 Populacija podatcima	15
3.4.3 Eloquent ORM	16
3.4.3.1 Modeli i tvornice modela	16
3.4.3.2 Relacije	21
3.4.3.3 Kolekcije	28
3.5 Usmjerivanje zahtjeva (engl. <i>routing</i>)	28

3.6	<i>Middleware</i>	32
3.7	Autentikacija	35
3.8	Autorizacija	36
3.8.1	Vrata	36
3.8.2	Politike	37
3.9	Livewire komponente	40
3.9.1	Akcije	43
3.9.2	Događaji	43
3.10	Pohrana datoteka	44
3.10.1	MinIO - AWS S3 kompatibilan servis za pohranu	44
3.11	Spatie Laravel Media Library paket	46
3.12	Lokalizacija	48
3.13	Testiranje i kvaliteta kôda	50
3.13.1	Korišteni alati	50
3.13.2	Pisanje i pokretanje testova	51
3.14	Otvoreni kôd i doprinos zajednice na projektima Laravel ekosustava	53
4	Prezentacija korisničkog sučelja pri korištenju aplikacije	54
5	Zaključak	55
Literatura		56
Dodatci		60

Sažetak

Cilj ovog završnog rada je razviti web aplikaciju koja će služiti kao platforma za timski rad na različitim projektima. Koristeći razvojni okvir Laravel izrađena je aplikacija *Teamstructor*. U navedenoj aplikaciji korisnici mogu formirati timove u kojima članovi tima imaju pristup projektima na kojima tim radi. Unutar svakog projekta članovi tima imaju pristup projektnoj diskusiji - mjestu pogodnom za dijeljenje svih novosti i ažuriranja vezanih uz taj projekt te pristup dijeljenim projektnim resursima - datotekama prenesenim od strane članova tima. Također je implementirano i jednostavno administrativno sučelje. U ovom pisanom radu bit će predstavljene tehnologije koje su korištene i način na koji je aplikacija implementirana.

Ključne riječi: *razvojni okvir Laravel, Livewire, Docker razvojno okruženje, pohrana datoteke, lokalizacija*

Summary

Developing a Teamwork Platform

The goal of this final thesis is to develop a web application that will serve as a platform for teamwork on various projects. The *Teamstructor* application was developed using the Laravel framework. In the mentioned application, users can form teams in which team members have access to the projects the team is working on. Within each project, team members have access to the project discussion - a place suitable for sharing all news and updates related to that project and access to shared project resources - files uploaded by team members. A simple administrative interface is also implemented. This paper will present the technologies that were used and the way in which the application was implemented.

Keywords: *Laravel framework, Livewire, Docker development environment, file storage, localization*

1. Uvod

U današnjem dinamičnom poslovnom svijetu, gotovo je nazamisliv posao koji barem u nekoj mjeri ne uključuje rad unutar tima. Također sve veći dio zaposlenika radi na izdvojenom radnom mjestu tj. *remote* ili hibridno, a sve manji radi isključivo iz ureda tj. *on-site*, pa je potrebno svladati i tu prepreku te nekadašnju komunikaciju isključivo licem u lice sada dijelom zamijeniti novim alatima. Kako bi tim uspješno djelovao i ostvario postavljeni cilj nužno je da su svi članovi tima međusobno usklađeni i mogu neometano surađivati - zato je potrebno osigurati da su članovi tima umreženi te mogu brzo primati novosti vezane uz poslovne zadatke dodijeljene timu te lako međusobno dijeliti resurse.

Kao praktični dio ovog završnog rada izrađena je web aplikacija *Teamstructor*. Registrirani korisnici mogu kreirati timove i pozivati članove u svoj tim. Članovi tima mogu kreirati različite projekte unutar tima. Unutar svakog projekta članovi tima mogu raspravljati i objavljivati novosti relevantne za projekt u formi objava i jednorazinskih komentara. Također, unutar svakog tima članovi tima mogu pregledavati dijeljene resurse te prenositi nove datoteke s osobnog računala među resurse. Prijavljeni korisnici mogu pristupati timovima i timskim projektima (te raspravama i resursima) samo ukoliko su članovi tog tima. Implementirano je i jednostavno administrativno sučelje dostupno korisniku s ulogom administratora u kojem može lako pregledavati postojeće korisnike, timove i projekte.

GitHub repozitorij u kojem je sadržan cijeli izvorni kôd aplikacije *Teamstructor* do supan je na poveznici <https://github.com/anamarijapapic/teamstructor>, a repozitorij s L^AT_EX datotekama korištenima za pisanje ovog rada na poveznici <https://github.com/anamarijapapic/teamstructor-docs>.

U poglavljima koja slijede prvo će se kratko predstaviti tehnologije korištene pri izradi aplikacije, a zatim će se detaljno i pregledno opisati način na koji je sama aplikacija implementirana.

2. Korištene tehnologije

2.1. Laravel

Laravel je PHP (engl. *PHP: Hypertext Preprocessor*) razvojni okvir (engl. *framework*) namijenjen razvoju web aplikacija zasnivanih na MVC ("Model-View-Controller") arhitekturi. Njegove značajke su da je slobodan, besplatan te otvorenog kôda, stoga je cijeli njegov izvorni kôd dostupan na platformi GitHub [1] gdje svatko može dati svoj doprinos pa se naziva i razvojnim okvirom zajednice (engl. *community framework*). Tvorac Laravela je Taylor Otwell, koji je 2011. godine razvio početnu verziju Laravela pokušavajući poboljšati tada popularan razvojni okvir CodeIgniter.

Trenutno je, uz Symfony, Laravel najpopularniji PHP razvojni okvir zahvaljujući svojoj jednostavnoj i izražajnoj sintaksi, detaljnoj dokumentaciji i obilnoj količini (video)vodiča, jasnoj strukturi, iznimnoj skalabilnosti te bogatom Laravel ekostustavu s dostupnom огромnom bibliotekom pomno održavanih paketa.

Pruža elegantna gotova "out-of-the-box" rješenja za česte značajke potrebne web aplikacijama: jednostavno i brzo usmjerivanje zahtjeva, moćno ubrizgavanje ovisnosti, pohranjivanje sesija i predmemorije, autentikaciju, autorizaciju, interakciju s podatcima u bazi, modelima, migracijama i relacijama, validaciju, slanje e-pošte i notifikacija, pohranu datoteke, red čekanja i pozadinsku obradu poslova, zakazivanje zadataka, događaje itd [2].

Dolazi s **Artisan** sučeljem naredbenog retka (engl. *command line interface*) koje pruža mnoge korisne naredbe koje pomažu pri razvoju aplikacije. Artisan naredbe su u formi `php artisan <command>`.

Instalacijom Laravela također se dobije i **Tinker** (REPL - "Read—Eval—Print—Loop") interaktivna ljska koja omogućava testne interakcije s modelima, poslovima, događajima itd. Za početak rada u Tinker razvojnog okruženju pokreće se Artisan naredba `php artisan tinker` [3].

Laravel i njegovi ostali paketi prve strane (engl. *first-party*) prate semantičko verzioniranje, pri čemu se *major* verzija razvojnog okvira Laravel sada izdaje svake godine, dok su *minor* i *patch* izdanja češća. Sva izdanja razvojnog okvira Laravel primaju ispravke pogrešaka do 18 mjeseci od izdavanja, a sigurnosne ispravke do 2 godine od izdavanja. Trenutna verzija razvojnog okvira Laravel je 10.x te zahtijeva minimalnu PHP verziju 8.1 [4].

2.2. Docker spremnici kao razvojno okruženje

Kako bi se web aplikacija jednako uspješno izvodila u različitim okruženjima - na različitim operativnim sustavima i arhitekturama računala, korištena je tehnologija Docker **spremnika** (engl. *containers*) te je aplikacija kontejnerizirana.

Svaki stroj koji ima instaliran Docker Engine može pokretati i stvarati Docker spremnike konstruirane iz Docker **slike** (engl. *image*). Za izgraditi sliku spremnika Docker koristi Dockerfile datoteku u kojoj se nalazi skripta s uputama za kreiranje iste. Za inicijalizaciju i izvođenje aplikacija koje se sastoje od više spremnika potreban je alat **Docker Compose** te su u tom slučaju aplikacijski servisi konfigurirani u docker-compose.yml datoteci. Instalacijom Docker Desktop aplikacije na računalo dobiva se i Docker Engine i Docker Compose V2 te dodatni alati. Upisivanjem naredbe docker compose up u terminal iz direktorija projekta pokreće se aplikacija i svi njeni servisi, a naredbom docker compose down zaustavljaju se svi pokrenuti servisi te se brišu spremnici [5]. Unutar docker-compose.yml datoteke kreirani su imenovani **volumeni** (engl. *volumes*) koji služe za očuvanje podataka koje generiraju i koriste Docker spremnici [6].

U nastavku će kratko biti opisane tehnologije koje su nužne za rad aplikacije, a definirane su kao servisi u docker-compose.yml datoteci.

2.2.1. Nginx

Kao lokalni web poslužitelj (engl. *server*) korišten je Nginx [engine x]. Kao početna točka za konfiguraciju web poslužitelja koristi se datoteka default.conf [7], u čijem se sadržaju unutar server direktive konfigurira virtualni poslužitelj [8].

2.2.2. PHP

PHP (rekurzivni akronim za *PHP: Hypertext Preprocessor*, a prije je kratica označavala *Personal Home Page*) popularni je besplatni skriptni jezik otvorenog kôda (izvorni kôd je dostupan na platformi GitHub [9]) koji se izvršava na poslužitelju (engl. *server-side scripting*). 1994. godine razvio ga je Rasmus Lerdorf, a sintaksa mu je bazirana na C, Java i Perl programskim jezicima. PHP je jezik opće namjene, ali je posebno prikladan za razvijanje web aplikacija [10].

PHP se koristi u najpopularnijem sustavu za upravljanje sadržajem (engl. *CMS - Content Management System*) - WordPressom.

tent Management System) – WordPressu, a postoji i nekoliko PHP razvojnih okvira: Laravel, Symfony, CodeIgniter, Zend, Yii 2, CakePHP, Fuel PHP, FATFree, Aura i dr. [10]

Trenutna verzija PHP jezika je 8.2, a trenutno je aktivno podržana i verzija 8.1. Svaka grana izdanja (engl. *release branch*) PHP-a u potpunosti je podržana dvije godine od svog prvog stabilnog izdanja i tijekom tog razdoblja za nju se objavljuju ispravci pogrešaka i sigurnosnih problema. Nakon tog razdoblja aktivne podrške, grana izdanja dobiva još jednu godinu podrške samo za kritične sigurnosne ispravke i to po potrebi. Nakon što isteknu tri godine podrške, grana dolazi do kraja života (engl. *end of life*) i više nije podržana [11].

2.2.3. MySQL

MySQL je besplatni program otvorenog kôda za upravljanje relacijskim bazama podataka (engl. *RDBMS - Relational Database Management System*). Pokrata "SQL" u imenu stoji za "*Structured Query Language*" - najčešći standardizirani jezik korišten za pristup bazama podataka. Najčešća alternativa MySQL-u su također besplatni programi otvorenog kôda MariaDB i PostgreSQL. Kod MySQL-a osnovni stroj baze podataka i ujedno i najčešće korišteni je InnoDB koji koristi transakcijski mehanizam [10].

2.2.4. phpMyAdmin

phpMyAdmin je besplatni softverski alat otvorenog kôda napisan u PHP-u koji podržava širok spektar operacija unutar MySQL i MariaDB relacijskih baza podataka. Radi se o intuitivnom grafičkom korisničkom sučelju (engl. *GUI - Graphical User Interface*) pa akcije mogu biti izvršene unutar korisničkog sučelja izravno u web pregledniku [10].

Može mu se pristupiti unosom adrese `http://localhost:8080` u web preglednik, pri čemu će se pojaviti phpMyAdmin "Welcome to phpMyAdmin" stranica koja traži unos korisničkog imena i lozničice.

2.3. Upravitelji paketima i ovisnostima

2.3.1. npm

npm (Node package manager) je upravitelj paketa za Node.js nastao 2009. godine kao projekt otvorenog kôda koji bi JavaScript programerima pomogao da jednostavno dijele zapakirane module kôda.

npm je klijent naredbenog retka koji programerima omogućuje instaliranje i objavljenje tih paketa.

Frontend paketi o kojima aplikacija ovisi zapisani su u `package.json` datoteku, a te iste ovisnosti (engl. *dependencies*) instaliraju se pozivom naredbe `npm install` iz terminala. Pokretanjem te naredbe kreira se `node_modules` direktorij koji u sebi sadrži poddirektorije - instalirane module tj. biblioteke potrebne za *frontend* dio aplikacije [12].

2.3.2. Composer

Composer je alat za upravljanje ovisnostima u PHP-u te je osnova modernog PHP razvoja. Omogućava deklariranje "paketa" tj. biblioteka o kojima projekt ovisi te će njima upravljati - voditi brigu o instalaciji, ažuriranju i uklanjanju istih na bazi projekta tako da će ih preuzeti unutar projekta u `vendor` direktorij. Unutar `composer.json` datoteke zapisane su željene ovisnosti projekta. Unutar `composer.lock` datoteke zapisani su svi instalirani paketi i njihove točne verzije, tako da se projekt "zaključava" na te konkretnе verzije paketa [13].

Određeni paket može se zatražiti naredbom `composer require` gdje se navodi naziv isporučitelja paketa, naziv paketa te ograničenje verzije paketa. Naredbom `composer install` instaliraju se sve Composer ovisnosti aplikacije, naredbom `composer update` ažuriraju se paketi, a naredbom `composer remove` i navođenjem naziva isporučitelja paketa i naziva paketa može se ukloniti paket iz liste ovisnosti [14].

Minimalna Composer verzija koju Laravel 10.x zahtjeva je 2.2.0.

2.4. Razvojna okruženja za testiranje

PHPUnit Sebastiana Bergmanna najpopularnije je PHP razvojno okruženje za testiranje. U Laravelu je podrška za testiranje s PHPUnitom zadano uključena te je `phpunit.xml` datoteka unaprijed postavljena. PHPUnit 10 je trenutna stabilna verzija.

Pest je razvojno okruženje za testiranje izgrađeno na PHPUnitu, ali s uključenim novim dodatnim značajkama. Podržava i pokretanje testova pisanih za PHPUnit. Trenutna verzija Pesta je 2.0.

3. Opis implementacije praktičnog rada

3.1. Instalacija i konfiguracija okruženja

3.1.1. Početni komplet Laravel Jetstream

Kako bi *developerima* uštedio vrijeme u samome početku razvijanja nove aplikacije, Laravel nudi autentikacijske i aplikacijske početne komplete (engl. *starter kits*) kao što su Laravel Breeze i Laravel Jetstream koji automatski pružaju rute, kontrolere i poglеде (engl. *views*) potrebne za registraciju i autentikaciju [15].

Laravel Jetstream dizajniran je koristeći **Tailwind CSS**, *utility-first* CSS razvojni okvir. Datoteke `postcss.config.js` i `tailwind.config.js` koriste se pri *buildanju* kompajliranog CSS-a aplikacije.

Značajke koje početni komplet Laravel Jetstream pruža su autentikacija, registracija, upravljanje korisničkim profilima, ponovno postavljanje lozinke, verifikacija e-adrese, dvostruka provjera autentičnosti (engl. *two-factor authentication*), upravljanje aktivnim sesijama u web preglednicima, API podrška te optionalne opcije za upravljanje timovima [16].

Instalira se koristeći Composer, a naredbe za instalaciju prikazane su u ispisu 1.

```
1 composer create-project laravel/laravel teamstructor-app  
2  
3 cd teamstructor-app  
4  
5 composer require laravel/jetstream
```

Ispis 1: Naredbe za instalaciju Jetstream paketa u novi Laravel projekt

Jetstream pruža izbor između korištenja **Livewire** ili Inertia.js *frontend scaffoldinga*. O tom izboru ovisi i odabrani jezik za predloške (engl. *templating language*) jer uz Livewire to je **Blade**, a uz Inertia.js to je Vue.js [16].

Za instaliranje Jetstreama s Livewire *frontend scaffoldingom* i to s uključenom podrškom za timove koristi se naredba

```
php artisan jetstream:install livewire --teams [17].
```

```
php artisan vendor:publish --tag=jetstream-views je naredba čiji se izvršavanjem u app direktoriju kreira direktorij resources/view/components koji sadrži razne generičke Blade komponente čija je svrha da ih se jednostavno može koristiti te pružanje konzistentnog korisničkog sučelja bez potrebe da developer kreira vlastite komponente [18].
```

Nakon instalacije Jetstreama potrebno je instalirati i pokrenuti *build* NPM ovisnosti pomoću naredbi `npm install` i `npm run build` te migrirati bazu naredbom `php artisan migrate` [17].

3.1.2. .env datoteka

.env datoteka nalazi se u aplikacijskom *root* direktoriju i služi kao konfiguracijska datoteka za Laravel web aplikaciju. Pri instalaciji Larabela kreirana je tako da se u nju kopira ogledna konfiguracijska datoteka `.env.example` [19].

Dobra je praksa i zbog sigurnosti i zbog toga što je konfiguracija promjenjiva ovisno o pojedinačnim okruženjima da neenkriptirana .env datoteka nije dio kontrole izvornog kôda aplikacije, dok je to u redu za `.env.example` datoteku te ista može poslužiti kao ogledni primjer s *placeholder* vrijednostima za variable koje je potrebno definirati [19].

Primjer definicije variabile može se vidjeti u ispisu 2.

```
1 APP_NAME=Teamstructor
```

Ispis 2: Definicija variabile u .env datoteci

U Laravel aplikaciji pristup vrijednosti pojedine variabile iz .env datoteke moguće je pomoću `$_ENV` PHP superglobalne variabile ili koristeći funkciju `env`, kao u ispisu 3.

```
1 'name' => env('APP_NAME', 'Laravel'),
```

Ispis 3: Pristup vrijednosti *environment* variabile u Laravelu

3.2. Struktura Laravel aplikacije

3.2.1. Struktura početnog direktorija s izvornim kôdom aplikacije

Osnovna struktura *root* direktorija Laravel aplikacije jest sljedeća [20]:

```
teamstructor-app
├── app
├── bootstrap
├── config
├── database
├── node_modules
├── public
├── resources
├── routes
└── storage
├── tests
└── vendor
```

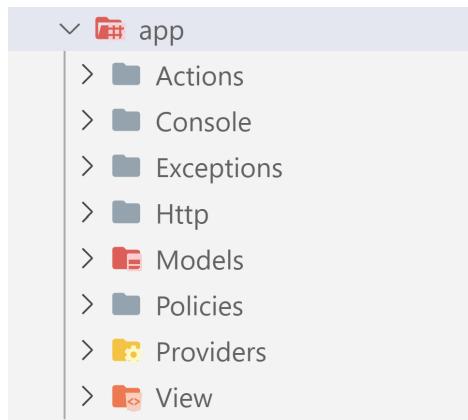
- **app** - Sadrži "jezgru" aplikacije te će biti posebno predstavljen u poglavlju 3.2.2.
- **bootstrap** - Sadrži datoteku `app.php` koja pokreće razvojni okvir te direktorij `cache` koji sadrži datoteke predmemorije za optimizaciju performansi.
- **config** - Sadrži konfiguracijske datoteke.
- **database** - Sadrži migracije, tvornice modela (engl. *model factories*) i *seed*-ove.
- **node_modules** - Nije u osnovnoj strukturi, međutim nastaje instalacijom NPM ovisnosti te sadrži instalirane module tj. biblioteke potrebne za *frontend* dio aplikacije kao poddirektorije.
- **public** - Sadrži datoteku `index.php` koja je polazna točka svih zahtjeva prema aplikaciji te konfigurira automatsko učitavanje (engl. *autoload*). Također se tu nalaze i javna "imovina" aplikacije (engl. *assets*): slike te JavaScript i CSS datoteke.
- **resources** - Sadrži poglede (engl. *views*) i nekompajlirane JavaScript i CSS datoteke koje su *assets* aplikacije.
- **routes** - Sadrži definicije svih ruta u aplikaciji. Zadano su uključene datoteke: `web.php`, `api.php`, `console.php` i `channels.php`.
- **storage** - Služi kao lokalno spremište podataka aplikacije te je podijeljen na direktorije `app`, `framework` i `logs`. Sadrži `.log` datoteke zapisnika, kompajlirane

Blade predloške, sesije, predmemorije datoteka itd.

- `tests` - Sadrži skripte za automatizirane aplikacijske testove te dolazi s po jednim oglednim primjerom *unit* i *feature* testova.
- `vendor` - Sadrži preuzete Composer ovisnosti aplikacije.

3.2.2. Struktura app direktorija

Na slici 1 prikazan je sadržaj app direktorija.



Slika 1: Sadržaj app direktorija

Može se vidjeti da se app direktorij sastoji od sljedećih poddirektorijsa [20]:

- `Actions` - Direktorij koji se kreira pri Jetstream instalaciji, a sadrži `Action` klase koje obično izvode samo jednu akciju i odgovaraju jednoj Jetstream ili Fortify značajki kao npr. kreiranje korisnika ili tima, postavljanje nove lozinke, brisanje korisnika ili tima, dodavanje člana u tim itd. [21]
- `Console` - Sadrži datoteku `Kernel.php` u kojoj se registriraju *custom* Artisan naredbe.
- `Exceptions` - Sadrži datoteku `Handler.php` koja upravlja svim iznimkama u aplikaciji te je moguće registrirati nove *custom* iznimke.
- `Http` - Gotovo sva logika aplikacije smještena je u ovaj direktorij. Sadrži kontrolere, klase Livewire komponenti te *middleware*.
- `Models` - Sadrži klase svih Eloquent modela ove web aplikacije.
- `Policies` - Nastaje izvođenjem `make:policy` Artisan naredbe te sadrži klase u

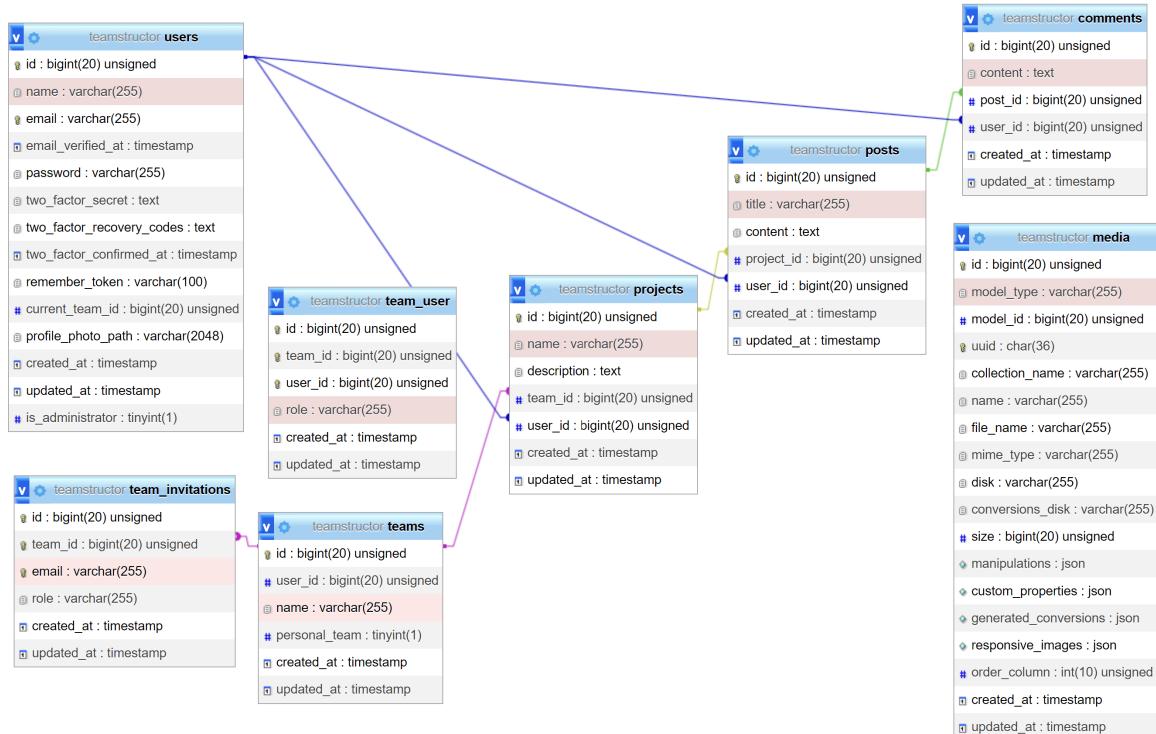
kojima su definirana pravila autorizacije unutar aplikacije.

- **Providers** - Sadrži sve davatelje usluga (engl. *service providers*) aplikacije.
- **View** - Direktorij se kreira pri Jetstream instalaciji, a sadrži klase Livewire *layout* komponenti - "application" *layout* te "guest" *layout* [22].

Još nekolicina direktorija može biti generirana unutar `app` direktorija izvršavanjem Artisan make naredbi za generiranje klasa [20].

3.3. Struktura relacijske baze podataka

Na slici 2 prikazan je ER (engl. *Entity Relationship*) dijagram relacijske baze podataka u phpMyAdmin sučelju na kojemu se može vidjeti tablice u koje su podatci spremani te njihove međusobne relacije.



Slika 2: Dijagram relacijske baze podataka

Radi preglednosti su na dijagramu sa slike 2 prikazane samo tablice koje se odnose na modele, a izostavljene su tablice:

- `failed_jobs` - Automatski prisutna u Laravel aplikacijama, služi za pohranu poslova iz reda čekanja koji se nisu uspjeli izvršiti.

- `migrations` - Automatski prisutna u Laravel aplikacijama, služi za pohranu migracija.
- `password_reset_tokens` - Automatski prisutna u Laravel aplikacijama, služi za pohranu tokena za ponovno postavljanje lozinki u aplikaciji.
- `personal_access_tokens` - Kreira je Laravel Sanctum pri instalaciji Laravel Jetstreama, a služi za pohranu API tokena u aplikaciji.
- `sessions` - Kada se koristi database *driver* za sesije, onda se u ovu tablicu one pohranjuju.
- `telescope_entries`, `telescope_entries_tags`,
`telescope_monitoring` - Nastaju pri instalaciji Laravel Telescopea, a služe za pohranu podataka za Telescope.

U poglavlju 3.4 će biti više rečeno o načinima na koje se ostvaruje interakcija s bazom podataka te o samim modelima i relacijama.

3.4. Interakcija s bazom podataka

3.4.1. Migracije

Migracije služe kao kontrola verzije baze podataka. U Laravelu migracije koriste fasadu (engl. *facade*) Schema koja pruža *database agnostic* podršku za sve sustave baza podataka koje Laravel podržava [23].

Migracija se može kreirati pozivom `make:migration` Artisan naredbe te će se ista smjestiti u `database/migrations` direktorij. Nazivu svake migracije dodaje se pripadajući *timestamp* prema kojem Laravel prati redoslijed migracija [23].

Sama migracijska klasa sadrži dvije metode: `up` koja kreira ili modifcira tablice, stupce ili indekse u bazi podataka te `down` koja bi trebala poništiti promjene učinjene u `up` metodi inverznim operacijama [23].

U ispisu 4 može se vidjeti kôd migracije za kreiranje tablice `projects` u kojoj se u `up` metodi kreira tablica navodeći njezin naziv i stupce koje sadrži, a u `down` metodi poništavaju operacije izvedene u `up` metodi na način da se odbacuje novokreirana tablica tj. izvodi `drop` tablice.

```

1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 return new class extends Migration
8 {
9     /**
10      * Run the migrations.
11     */
12     public function up(): void
13     {
14         Schema::create('projects', function (Blueprint $table) {
15             $table->id();
16             $table->string('name');
17             $table->text('description');
18             $table->foreignId('team_id')->constrained();
19             $table->foreignId('user_id')->constrained();
20             $table->timestamps();
21         });
22     }
23
24     /**
25      * Reverse the migrations.
26     */
27     public function down(): void
28     {
29         Schema::dropIfExists('projects');
30     }
31 };

```

Ispis 4: Migracija za kreiranje tablice projects

U ispisu 5 prikazana je migracija koja modifcira tablicu `users` tako da joj dodaje stupac `is_administrator`. U `up` metodi definira se stupac koji će se dodati, a u `down` metodi te će promjene biti poništene odbacivanjem novododanog stupca metodom `dropColumn`.

```

1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 return new class extends Migration
8 {
9     /**
10      * Run the migrations.
11     */
12     public function up(): void
13     {
14         Schema::table('users', function (Blueprint $table) {
15             $table->boolean('is_administrator')->default(false);
16         });
17     }
18
19     /**
20      * Reverse the migrations.
21     */
22     public function down(): void
23     {
24         Schema::table('users', function (Blueprint $table) {
25             $table->dropColumn('is_administrator');
26         });
27     }
28 };

```

Ispis 5: Migracija za dodavanje stupca is_administrator u tablicu users

Za izvršiti sve migracije koje se još nisu primijenile na bazu podataka pokreće se Artisan naredba `migrate`, a za povratak na prethodnu migraciju tj. poništavanje promjena posljednjih *n* migracija Artisan naredba `migrate:rollback` [23].

3.4.2. Populacija podatcima

Populacija baze podataka (engl. *seeding*) u Laravelu se može obaviti koristeći *seed* klase koje se nalaze u direktoriju database/seeds. Ondje se zadano nalazi DatabaseSeeder klasa koja sadrži metodu run koja se izvršava pozivom db:seed Artisan naredbe. Unutar metode run koristeći se metodom call pozivaju se dodatni ostali u aplikaciji definirani *seederi*. Dobra je praksa *seeding* kôd podijeliti u više dijelova tj. *seed* klasa jer se tako ne "zatrپava" isključivo jedan *seeder* kôdom. U ispisu 6 može se vidjeti *seedere* pozvane koristeći metodu call [24].

```
1 <?php
2
3 namespace Database\Seeders;
4
5 // use Illuminate\Database\Console\Seeds\WithoutModelEvents;
6 use Illuminate\Database\Seeder;
7
8 class DatabaseSeeder extends Seeder
9 {
10     /**
11      * Seed the application's database.
12     */
13     public function run(): void
14     {
15         $this->call([
16             UserSeeder::class,
17             TeamSeeder::class,
18             ProjectSeeder::class,
19             PostSeeder::class,
20             CommentSeeder::class,
21             ResourceSeeder::class,
22         ]);
23     }
24 }
```

Ispis 6: Sadržaj DatabaseSeeder klase

Samo ubacivanje podataka u bazu podataka koristeći *seedere* može se ostvariti ručno koristeći *query builder* ili koristeći Eloquent tvornice modela (engl. *model factories*) o čemu će biti više riječi u nastavku.

3.4.3. Eloquent ORM

Eloquent ORM (engl. *object-relational mapper*) je Laravelova biblioteka za objektno-relacijsko mapiranje čija je uloga olakšati interakcije s bazom podataka jer obavlja konverziju kôda napisanog u objektno orijentiranom jeziku u jezik relacijske baze podataka i obrnuto. Omogućava stvaranje virtualne objektne baze podataka koja je prilagođena korištenju unutar programskog jezika [25].

3.4.3.1. Modeli i tvornice modela

Eloquent modeli nalaze se u direktoriju `app/Models` te se za stvaranje novog modela koristi Artisan naredba `make:model`. Svaka tablica baze podataka odgovara jednom Eloquent modelu te će on biti korišten za operacije nad istom. Prema konvenciji, ako drugačije nije definirano u samoj klasi, naziv odgovarajuće tablice je naziv klase modela u množini u *snake caseu*. Tako su u tablici 1 prikazani nazivi klasa modela te odgovarajući nazivi tablica (koje su prikazane i na slici 2).

Tablica 1: Nazivi klasa modela i pripadajuće tablice

Naziv klase modela	Naziv odgovarajuće tablice
Comment	comments
Membership	team_user
Post	posts
Project	projects
Resource	media
Team	teams
TeamInvitation	team_invitations
User	users

U klasi modela definiraju se njegovi atributi (*fillable* - one koji su *mass assignable* tj. dozvoljeno je da ih dodijele korisnici aplikacije te *hidden* - one koji su skriveni pri serijalizaciji), relacije (o kojima će biti više riječi u poglavljju 3.4.3.2), *traits* tj. osobine koje koristi,

događaje koje emitira itd.

U ispisu 7 prikazan je dio kôda iz klase modela `User` gdje je vidljivo da koristi neko-licinu osobina (engl. *traits*) te definiranje atributa modela.

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Relations\hasMany;
7 use Illuminate\Foundation\Auth\User as Authenticatable;
8 use Illuminate\Notifications\Notifiable;
9 use Laravel\Fortify\TwoFactorAuthenticatable;
10 use Laravel\Jetstream\HasProfilePhoto;
11 use Laravel\Jetstream\HasTeams;
12 use Laravel\Sanctum\HasApiTokens;
13
14 class User extends Authenticatable
15 {
16     use HasApiTokens;
17     use HasFactory;
18     use HasProfilePhoto;
19     use HasTeams;
20     use Notifiable;
21     use TwoFactorAuthenticatable;
22
23     /**
24      * The attributes that are mass assignable.
25      *
26      * @var array<int, string>
27      */
28     protected $fillable = [
29         'name', 'email', 'password',
30     ];
31
32     /**
33      * The attributes that should be hidden for serialization.
```

```

34     *
35     * @var array<int, string>
36     */
37 protected $hidden = [
38     'password',
39     'remember_token',
40     'two_factor_recovery_codes',
41     'two_factor_secret',
42     'is_administrator',
43 ];
44
45 /**
46 * The attributes that should be cast.
47 *
48 * @var array<string, string>
49 */
50 protected $casts = [
51     'email_verified_at' => 'datetime',
52 ];
53
54 /**
55 * The accessors to append to the model's array form.
56 *
57 * @var array<int, string>
58 */
59 protected $appends = [
60     'profile_photo_url',
61 ];
62
63 // ...

```

Ispis 7: Dio kôda model klase User

U ispisu 7 može se primjetiti da model `User` koristi osobinu `HasFactory`. Pomoću tvornica modela (engl. *model factories*) može se definirati skup zadanih atributa za određeni Eloquent model pa je olakšano populiranje baze podataka sa testnim zapisima. Tvornica modela može se generirati Artisan naredbom `make:factory` te će se nalaziti u `database/factories` direktoriju [26].

U ispisu 8 prikazan je dio kôda iz tvornice modela `UserFactory` gdje je vidljiva metoda `definition` koja vraća skup atributa koji bi se trebali primijeniti na model pri njegovom kreiranju. Za generiranje testnih lažnih podataka koristi se Faker [27] PHP biblioteka čija je instanca u `$this->faker` [26].

```
1 <?php
2
3 namespace Database\Factories;
4
5 use App\Models\Team;
6 use App\Models\User;
7 use Illuminate\Database\Eloquent\Factories\Factory;
8 use Illuminate\Support\Str;
9 use Laravel\Jetstream\Features;
10
11 class UserFactory extends Factory
12 {
13     /**
14      * The name of the factory's corresponding model.
15      *
16      * @var string
17      */
18     protected $model = User::class;
19
20     /**
21      * Define the model's default state.
22      *
23      * @return array<string, mixed>
24      */
25     public function definition(): array
26     {
27         return [
28             'name' => $this->faker->name(),
29             'email' => $this->faker->unique()->safeEmail(),
30             'email_verified_at' => now(),
31             'password' => '...', // password
32             'two_factor_secret' => null,
33             'two_factor_recovery_codes' => null,
```

```

34     'remember_token' => Str::random(10),
35     'profile_photo_path' => null,
36     'current_team_id' => null,
37     'is_administrator' => false,
38 ];
39 }
40
41 // ...

```

Ispis 8: Dio kôda tvornice modela UserFactory

Također, tvornicama modela mogu se definirati stanja (engl. *factory states*) koja predstavljaju način da se primijene diskretne izmjene tj. modificiraju atributi na nekoj tvornici modela. U ispisu 9 prikazan je dio kôda iz tvornice modela UserFactory gdje se definira stanje tvornice modela za stvaranje administrator korisnika. Vidljivo je da to stanje vrši izmjene samo na odabranim atributima, a ostali će atributi biti postavljeni kao u izvornoj definiciji tvornice modela [26].

```

1 /**
2  * Indicate that the user is administrator.
3 */
4 public function administrator(): static
5 {
6     return $this->state(function (array $attributes) {
7         return [
8             'name' => 'Admin Admin',
9             'email' => 'admin@teamstructor.test',
10            'is_administrator' => true,
11        ];
12    });
13 }

```

Ispis 9: Dio kôda tvornice modela UserFactory koji prikazuje definiciju stanja tvornice modela

3.4.3.2. Relacije

Odnosi tj. relacije između modela definiraju se unutar samih klasa Eloquent modela kao metode. Može se ponovno pogledati sliku 2 i obratiti pozornost na veze među modelima.

Model `User` definira veze prema modelima `Project`, `Post` i `Comments` kao što je vidljivo u ispisu 10, a u osobini `HasTeams` koju model koristi definirane su veze prema modelima `Team` i `Membership` te su te metode prikazane u ispisu 11.

```
1  /**
2   * Get the projects created by user.
3   */
4  public function projects(): HasMany
5  {
6      return $this->hasMany(Project::class);
7  }
8
9 /**
10 * Get the user's posts.
11 */
12 public function posts(): HasMany
13 {
14     return $this->hasMany(Post::class);
15 }
16
17 /**
18 * Get the user's comments.
19 */
20 public function comments(): HasMany
21 {
22     return $this->hasMany(Comment::class);
23 }
```

Ispis 10: Relacije modela `User`

```
1 /**
2 * Get all of the teams the user owns.
3 *
```

```

4     * @return \Illuminate\Database\Eloquent\Relations\HasMany
5     */
6 public function ownedTeams()
7 {
8     return $this->hasMany(Jetstream::teamModel());
9 }
10
11 /**
12 * Get all of the teams the user belongs to.
13 *
14 * @return \Illuminate\Database\Eloquent\Relations\BelongsToMany
15 */
16 public function teams()
17 {
18     return $this->belongsToMany(
19         Jetstream::teamModel(),
20         Jetstream::membershipModel()
21     )->withPivot('role')
22     ->withTimestamps()
23     ->as('membership');
24 }

```

Ispis 11: Relacije modela User koje koristi iz osobine HasTeams

Model Team definira veze prema modelu Project, kao što je vidljivo u ispisu 12, a nasljeđujući model JetstreamTeam definirane su veze prema modelima User, Membership i TeamInvitation te su te metode prikazane u ispisu 13.

```

1 <?php
2
3 namespace App\Models;
4
5 // ...
6 use Illuminate\Database\Eloquent\Relations\HasMany;
7 // ...
8 use Laravel\Jetstream\Team as JetstreamTeam;
9
10 class Team extends JetstreamTeam

```

```

11  {
12
13 // ...
14
15     /**
16      * Get the projects for the team.
17     */
18
19     public function projects(): HasMany
20     {
21         return $this->hasMany(Project::class)->latest();
22     }
23 // ...

```

Ispis 12: Relacije modela Team

```

1     /**
2      * Get the owner of the team.
3      *
4      * @return \Illuminate\Database\Eloquent\Relations\BelongsTo
5     */
6
7     public function owner()
8     {
9         return $this->belongsTo(Jetstream::userModel(), 'user_id');
10    }
11
12    /**
13     * Get all of the users that belong to the team.
14     *
15     * @return \Illuminate\Database\Eloquent\Relations\BelongsToMany
16    */
17
18    public function users()
19    {
20        return $this->belongsToMany(
21                    Jetstream::userModel(),
22                    Jetstream::membershipModel()
23                )
24                ->withPivot('role')
25                ->withTimestamps()

```

```

23             ->as('membership');
24     }
25
26     /**
27      * Get all of the pending user invitations for the team.
28      *
29      * @return \Illuminate\Database\Eloquent\Relations\HasMany
30      */
31     public function teamInvitations()
32     {
33         return $this->hasMany(Jetstream::teamInvitationModel());
34     }

```

Ispis 13: Relacije modela Team koje nasljeđuje od modela JetstreamTeam

Model Membership je pivotni model (između modela Team i User), a model TeamInvitation definira vezu prema modelu Team kao što je prikazano u ispisu 14:

```

1 /**
2  * Get the team that the invitation belongs to.
3 */
4 public function team(): BelongsTo
5 {
6     return $this->belongsTo(Jetstream::teamModel());
7 }

```

Ispis 14: Relacije modela TeamInvitation

Model Project definira veze prema modelima Team, User i Post vidljivo u ispisu 15, te implementirajući *interface* HasMedia i koristeći osobinu InteractsWithMedia definira i vezu prema modelu Resource vidljivo u ispisu 16.

```

1 <?php
2
3 namespace App\Models;
4
5 // ...

```

```

6 use Illuminate\Database\Eloquent\Model;
7 use Illuminate\Database\Eloquent\Relations\BelongsTo;
8 use Illuminate\Database\Eloquent\Relations\HasMany;
9 use Spatie\MediaLibrary\HasMedia;
10 use Spatie\MediaLibrary\InteractsWithMedia;
11
12 class Project extends Model implements HasMedia
13 {
14     // ...
15     use InteractsWithMedia;
16
17     // ...
18
19     /**
20      * Get the team that owns the project.
21     */
22     public function team(): BelongsTo
23     {
24         return $this->belongsTo(Team::class);
25     }
26
27     /**
28      * Get the user that created the project.
29     */
30     public function user(): BelongsTo
31     {
32         return $this->belongsTo(User::class);
33     }
34
35     /**
36      * Get the posts for the project.
37     */
38     public function posts(): HasMany
39     {
40         return $this->hasMany(Post::class)
41             ->with(['user', 'comments'])
42             ->latest();
43     }
44

```

```
45 // ...
```

Ispis 15: Relacije modela Project

```
1 public function media(): MorphMany
2 {
3     return $this->morphMany(config('media-library.media_model'), 'model')
4 }
```

Ispis 16: Relacije modela Project koje koristi iz osobine InteractsWithMedia

Model Post definira veze prema modelima Project, User i Comment kao što je prikazano u ispisu 17, a model Comment definira veze prema modelima Post i User, prikazano u ispisu 18.

```
1 /**
2  * Get the project that owns the post.
3 */
4 public function project(): BelongsTo
5 {
6     return $this->belongsTo(Project::class);
7 }
8
9 /**
10 * Get the author of the post.
11 */
12 public function user(): BelongsTo
13 {
14     return $this->belongsTo(User::class);
15 }
16
17 /**
18 * Get the comments for the post.
19 */
20 public function comments(): HasMany
21 {
```

```
22         return $this->hasMany(Comment::class)->latest();  
23     }
```

Ispis 17: Relacije modela Post

```
1  /**
2   * Get the post that owns the comment.
3   */  
4  public function post(): BelongsTo  
5  {  
6      return $this->belongsTo(Post::class);  
7  }  
8  
9  /**
10  * Get the author of the comment.
11  */  
12 public function user(): BelongsTo  
13 {  
14     return $this->belongsTo(User::class);  
15 }
```

Ispis 18: Relacije modela Comment

Model Resource, nasljeđujući model Media iz Spatie Media Library paketa, definira vezu prema modelu Project, prikazano u ispisu 19.

```
1 public function model(): MorphTo  
2 {  
3     return $this->morphTo();  
4 }
```

Ispis 19: Relacije modela Resource koje nasljeđuje od modela Media

U ovim prethodno navedenim odnosima među modelima može se primijetiti da su korištene Eloquent relacije HasMany i inverzna joj BelongsTo - koje označavaju vezu jedan prema više (1:N) odnosno više prema jedan (N:1). Za ostvarivanje veze više prema više

(N:M) korištena je Eloquent relacija BelongsToMany. Također su korištene i polimorfne Eloquent relacije MorphMany i inverzna joj MorphTo - koje označavaju polimofnu vezu (neovisnu o tipu modela) jedan prema više (1:N) odnosno više prema jedan (N:1) [28].

3.4.3.3. Kolekcije

U Laravelu postoji bazna klasa Illuminate\Support\Collection koja pruža praktičan *wrapper* (omotač) za rad s nizovima podataka. Eloquent upiti tj. sve Eloquent metode koje vraćaju više od jednog rezultata vraćaju instancu klase Illuminate\Database\Eloquent\Collection. Objekt Eloquent kolekcija nasljeđuje baznu Laravel kolekciju pa tako nasljeđuje i sve metode dosupne u baznoj klasi [29].

Neke od najkorištenijih metoda su all, count, dd, dump, except, find, first, firstOrFail, get, only, pop, push, sortBy, sortByDesc, toArray, toJson, where, whereBetween, whereIn itd. [30]

3.5. Usmjerivanje zahtjeva (engl. *routing*)

Sve rute u Laravel aplikaciji definirane su unutar datoteka koje se nalaze unutar direktorija routes. Te su datoteke automatski učitane zahvaljujući

App\Providers\RouteServiceProvider. U datoteci routes/web.php definirane su rute korištene na aplikacijskom web sučelju, koje se mogu dohvatiti unosom URL-a u web preglednik. Rutama iz routes/web.php dodijeljena je web middleware grupa koja pruža značajke kao što su stanje sesija i CSRF (engl. Cross-Site Request Forgery) zaštita [31].

Ispis 20 prikazuje sadržaj datoteke routes/web.php gdje je vidljiv način na koji su rute definirane. Podijeljene su prvenstveno u dvije grupe - prva dostupna svim prijavljenim korisnicima (grupa koristi middleware koji zahtjeva autentikaciju), te druga dostupna samo admin korisnicima (korišten custom middleware can:admin-privileges), a zadnja definirana ruta je ona koju koristi kontoler pri promjeni jezika aplikacije o čemu će biti riječi u poglavljju 3.12. Svakoj ruti dodijeljeno je jedinstveno ime koristeći metodu name.

```
1 <?php  
2  
3 use App\Http\Controllers\LanguageController;
```

```

4 use App\Http\Livewire\Admin>ShowProjects as AdminShowProjects;
5 use App\Http\Livewire\Admin>ShowTeams;
6 use App\Http\Livewire\Admin>ShowUsers;
7 use App\Http\Livewire\Media>ShowResources;
8 use App\Http\Livewire\Posts>ShowPosts;
9 use App\Http\Livewire\Projects>ShowProjects;
10 use Illuminate\Support\Facades\Route;
11
12 /**
13 ----- Web Routes
14 -----
15 Here is where you can register web routes for your application. These
   routes are loaded by the RouteServiceProvider and all of them will
16 be assigned to the "web" middleware group. Make something great!
17 */
18
19 Route::middleware([
20     'auth:sanctum',
21     config('jetstream.auth_session'),
22     'verified',
23 ])>group(function () {
24     Route::get('/', function () {
25         return view('dashboard');
26     })>name('dashboard');
27
28     Route::get('teams/{team}/projects', ShowProjects::class)
29         >name('teams.projects');
30
31     Route::get('teams/{team}/projects/{project}/discussion', ShowPosts::
32         class)
33         >name('teams.projects.discussion');
34
35     Route::get('teams/{team}/projects/{project}/resources', ShowResources
36         ::class)
37         >name('teams.projects.resources');
38 });
39
40 Route::group([
41     'prefix' => 'admin',

```

```

40     'middleware' => ['auth:sanctum', 'can:admin-privileges'],
41 ], function () {
42     Route::get('/', function () {
43         return view('admin-dashboard');
44     })->name('admin.dashboard');
45
46     Route::get('/users', ShowUsers::class)
47         ->name('admin.dashboard.users');
48
49     Route::get('/teams', ShowTeams::class)
50         ->name('admin.dashboard.teams');
51
52     Route::get('/projects', AdminShowProjects::class)
53         ->name('admin.dashboard.projects');
54 });
55
56 Route::get('locale/{locale}', [LanguageController::class, 'switchLocale'])
57     ->name('locale.switch');

```

Ispis 20: Sadržaj datoteke routes/web.php

Izvršavanjem Artisan naredbe `route:list` mogu se izlistati sve rute definirane u aplikaciji te je rezultat izvršavanja iste prikazan na slici 3.

```

GET|HEAD / ..... dashboard
GET|HEAD _debugbar/assets/javascript ..... debugbar.assets.js > Barryvdh\Debugbar > AssetController@js
GET|HEAD _debugbar/assets/stylesheets ..... debugbar.assets.css > Barryvdh\Debugbar > AssetController@css
DELETE _debugbar/cache/{key}/{tags} ..... debugbar.cache.delete > Barryvdh\Debugbar > CacheController@delete
GET|HEAD _debugbar/clockwork/{id} ..... debugbar.clockwork > Barryvdh\Debugbar > OpenHandlerController@clockwork
GET|HEAD _debugbar/open ..... debugbar.openhandler > Barryvdh\Debugbar > OpenHandlerController@handle
GET|HEAD _debugbar/telescope/{id} ..... debugbar.telescope > Barryvdh\Debugbar > TelescopeController@show
POST ignition/execute-solution ..... ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolutionController
GET|HEAD _ignition/health-check ..... ignition.healthCheck > Spatie\LaravelIgnition > HealthCheckController
POST _ignition/update-config ..... ignition.updateConfig > Spatie\LaravelIgnition > UpdateConfigController
GET|HEAD admin ..... admin.dashboard
GET|HEAD admin/projects ..... admin.dashboard.projects > App\Http\Livewire\Admin>ShowProjects
GET|HEAD admin/teams ..... admin.dashboard.teams > App\Http\Livewire\Admin>ShowTeams
GET|HEAD admin/users ..... admin.dashboard.users > App\Http\Livewire\Admin>ShowUsers
GET|HEAD api/user ..... 
PUT current-team ..... current-team.update > Laravel\Jetstream > CurrentTeamController@update
GET|HEAD forgot-password ..... password.request > Laravel\Fortify > PasswordResetLinkController@create
POST forgot-password ..... password.email > Laravel\Fortify > PasswordResetLinkController@store
GET|HEAD livewire/livewire.js ..... Livewire\Controllers > LivewireJavaScriptAssets@source
GET|HEAD livewire/livewire.js.map ..... Livewire\Controllers > LivewireJavaScriptAssets@maps
POST livewire/message/{name} ..... livewire.message > Livewire\Controllers > HttpConnectionHandler
GET|HEAD livewire/preview-file/{filename} ..... livewire.preview-file > Livewire\Controllers > FilePreviewHandler@handle
POST livewire/upload-file ..... livewire.upload-file > Livewire\Controllers > FileUploadHandler@handle
GET|HEAD locale/{locale} ..... locale.switch > LanguageController@switchLocale
GET|HEAD login ..... login > Laravel\Fortify > AuthenticatedSessionController@create
POST login ..... logout > Laravel\Fortify > AuthenticatedSessionController@store
POST logout ..... logout > Laravel\Fortify > AuthenticatedSessionController@destroy
GET|HEAD register ..... register > Laravel\Fortify > RegisteredUserController@create
POST register ..... Laravel\Fortify > RegisteredUserController@store
POST reset-password ..... password.update > Laravel\Fortify > NewPasswordController@store
GET|HEAD reset-password/{token} ..... password.reset > Laravel\Fortify > NewPasswordController@create
GET|HEAD sanctum/csrf-cookie ..... sanctum.csrf-cookie > Laravel\Sanctum > CsrfCookieController@show
GET|HEAD team-invitations/{invitation} ..... team-invitations.accept > Laravel\Jetstream > TeamInvitationController@accept
GET|HEAD teams/create ..... teams.create > Laravel\Jetstream > TeamController@create
GET|HEAD teams/{team} ..... teams.show > Laravel\Jetstream > TeamController@show
GET|HEAD teams/{team}/projects ..... teams.projects > App\Http\Livewire\Projects>ShowProjects
GET|HEAD teams/{team}/projects/{project}/discussion ..... teams.projects.discussion > App\Http\Livewire\Posts>ShowPosts
GET|HEAD teams/{team}/projects/{project}/resources ..... teams.projects.resources > App\Http\Livewire\Media>ShowResources
POST telescope/telescope-api/batches ..... Laravel\Telescope > QueueBatchesController@index
GET|HEAD telescope/telescope-api/batches/{telescopeEntryId} ..... Laravel\Telescope > QueueBatchesController@show
POST telescope/telescope-api/cache ..... Laravel\Telescope > CacheController@index
GET|HEAD telescope/telescope-api/cache/{telescopeEntryId} ..... Laravel\Telescope > CacheController@show
POST telescope/telescope-api/client-requests ..... Laravel\Telescope > ClientRequestController@index
GET|HEAD telescope/telescope-api/client-requests/{telescopeEntryId} ..... Laravel\Telescope > ClientRequestController@show
POST telescope/telescope-api/commands ..... Laravel\Telescope > CommandsController@index
GET|HEAD telescope/telescope-api/commands/{telescopeEntryId} ..... Laravel\Telescope > CommandsController@show
POST telescope/telescope-api/dumps ..... Laravel\Telescope > DumpController@index
DELETE telescope/telescope-api/entries ..... Laravel\Telescope > EntriesController@destroy
POST telescope/telescope-api/events ..... Laravel\Telescope > EventsController@index
GET|HEAD telescope/telescope-api/events/{telescopeEntryId} ..... Laravel\Telescope > EventsController@show
POST telescope/telescope-api/exceptions ..... Laravel\Telescope > ExceptionController@index
GET|HEAD telescope/telescope-api/exceptions/{telescopeEntryId} ..... Laravel\Telescope > ExceptionController@show
PUT telescope/telescope-api/exceptions/{telescopeEntryId} ..... Laravel\Telescope > ExceptionController@update
POST telescope/telescope-api/gates ..... Laravel\Telescope > GatesController@index
GET|HEAD telescope/telescope-api/gates/{telescopeEntryId} ..... Laravel\Telescope > GatesController@show
POST telescope/telescope-api/jobs ..... Laravel\Telescope > QueueController@index
GET|HEAD telescope/telescope-api/jobs/{telescopeEntryId} ..... Laravel\Telescope > QueueController@show
POST telescope/telescope-api/logs ..... Laravel\Telescope > LogController@index
GET|HEAD telescope/telescope-api/logs/{telescopeEntryId} ..... Laravel\Telescope > LogController@show
POST telescope/telescope-api/mail ..... Laravel\Telescope > MailController@index
GET|HEAD telescope/telescope-api/mail/{telescopeEntryId} ..... Laravel\Telescope > MailController@show
GET|HEAD telescope/telescope-api/mail/{telescopeEntryId}/download ..... Laravel\Telescope > MailImController@show
GET|HEAD telescope/telescope-api/mail/{telescopeEntryId}/preview ..... Laravel\Telescope > MailHtmlController@show
POST telescope/telescope-api/models ..... Laravel\Telescope > ModelsController@index
GET|HEAD telescope/telescope-api/models/{telescopeEntryId} ..... Laravel\Telescope > ModelsController@show
GET|HEAD telescope/telescope-api/monitored-tags ..... Laravel\Telescope > MonitoredTagController@index
POST telescope/telescope-api/monitored-tags ..... Laravel\Telescope > MonitoredTagController@store
POST telescope/telescope-api/monitored-tags/delete ..... Laravel\Telescope > MonitoredTagController@destroy
POST telescope/telescope-api/notifications ..... Laravel\Telescope > NotificationsController@index
GET|HEAD telescope/telescope-api/notifications/{telescopeEntryId} ..... Laravel\Telescope > NotificationsController@show
POST telescope/telescope-api/queries ..... Laravel\Telescope > QueriesController@index
GET|HEAD telescope/telescope-api/queries/{telescopeEntryId} ..... Laravel\Telescope > QueriesController@show
POST telescope/telescope-api/redis ..... Laravel\Telescope > RedisController@index
GET|HEAD telescope/telescope-api/redis/{telescopeEntryId} ..... Laravel\Telescope > RedisController@show
POST telescope/telescope-api/requests ..... Laravel\Telescope > RequestsController@index
GET|HEAD telescope/telescope-api/requests/{telescopeEntryId} ..... Laravel\Telescope > RequestsController@show
POST telescope/telescope-api/schedule ..... Laravel\Telescope > ScheduleController@index
GET|HEAD telescope/telescope-api/schedule/{telescopeEntryId} ..... Laravel\Telescope > ScheduleController@show
POST telescope/telescope-api/toggle-recording ..... Laravel\Telescope > RecordingController@toggle
POST telescope/telescope-api/views ..... Laravel\Telescope > ViewsController@index
GET|HEAD telescope/telescope-api/views/{telescopeEntryId} ..... Laravel\Telescope > ViewsController@show
GET|HEAD {view?} ..... telescope > Laravel\Telescope > HomeController@index
GET|HEAD two-factor-challenge ..... two-factor.login > Laravel\Fortify > TwoFactorAuthenticatedSessionController@create
POST two-factor-challenge ..... Laravel\Fortify > TwoFactorAuthenticatedSessionController@store
GET|HEAD user/confirm-password ..... Laravel\Fortify > ConfirmablePasswordController@show
POST user/confirm-password ..... password.confirm > Laravel\Fortify > ConfirmablePasswordController@store
GET|HEAD user/confirmed-password-status ..... password.confirmation > Laravel\Fortify > ConfirmedPasswordStatusController@show
POST user/confirmed-two-factor-authentication two-factor.confirm > Laravel\Fortify > ConfirmedTwoFactorAuthenticationController@store
PUT user/password ..... user.password.update > Laravel\Fortify > PasswordController@update
GET|HEAD user/profile ..... profile.show > Laravel\Jetstream > UserProfileController@show
PUT user/profile-information ..... user-profile-information.update > Laravel\Fortify > ProfileInformationController@update
POST user/two-factor-authentication ..... two-factor.enable > Laravel\Fortify > TwoFactorAuthenticationController@store
DELETE user/two-factor-authentication ..... two-factor.disable > Laravel\Fortify > TwoFactorAuthenticationController@destroy
GET|HEAD user/two-factor-qr-code ..... two-factor.qr-code > Laravel\Fortify > TwoFactorQrCodeController@show
GET|HEAD user/two-factor-recovery-codes ..... two-factor.recovery-codes > Laravel\Fortify > RecoveryCodeController@index
POST user/two-factor-recovery-codes ..... Laravel\Fortify > RecoveryCodeController@store
GET|HEAD user/two-factor-secret-key ..... two-factor.secret-key > Laravel\Fortify > TwoFactorSecretKeyController@show
POST {locale}/livewire/message/{name} ..... livewire.message-localized > Livewire\Controllers > HttpConnectionHandler

```

Showing [98] routes

Slika 3: Izlist svih ruta definiranih u aplikaciji pomoću Artisan naredbe `route:list`

3.6. *Middleware*

Middleware, srednji sloj Laravel web aplikacije, mehanizam je za pregled, filtriranje i kontrolu korisničkih HTTP zahtjeva koji dolaze aplikaciji. Nekoliko *middlewarea* zadano je uključeno u razvojni okvir Laravel, uključujući onaj za autentikaciju i CSRF zaštitu. Sve *middleware* klase nalaze se u direktoriju app/Http/Middleware te se novi *middleware* može stvoriti Artisan naredbom make:middleware [32].

U ispisu 21 prikazan je ConfigureLocale *middleware* napisan za potrebe implementacije lokalizacije.

```
1 <?php
2
3 namespace App\Http\Middleware;
4
5 use Closure;
6 use Illuminate\Http\Request;
7 use Illuminate\Support\Facades\App;
8 use Illuminate\Support\Facades\Session;
9 use Symfony\Component\HttpFoundation\Response;
10
11 class ConfigureLocale
12 {
13     /**
14      * Handle an incoming request.
15      *
16      * @param \Closure(\Illuminate\Http\Request): (\Symfony\Component\
17      * HttpFoundation\Response) $next
18      */
19
20     public function handle(Request $request, Closure $next): Response
21     {
22         if ($request->has('locale')) {
23             $locale = $request->get('locale');
24             Session::put('locale', $locale);
25         }
26
27         if (Session::has('locale') && array_key_exists(Session::get('
28             locale'), config('locales'))) {
```

```

26         App::setLocale(Session::get('locale'));
27     } else {
28
29         App::setLocale(config('app.fallback_locale'));
30
31     return $next($request);
32 }
33 }
```

Ispis 21: Middleware ConfigureLocale

Pojedini *middleware* treba se registrirati u aplikaciji - ako se želi da je aktivan globalno dodaje ga se među vrijednosti podatkovnog člana `$middleware` klase `app/Http/Kernel.php`, a ako se želi da djeluje na pojedinoj ruti na nju ga se dodaje pomoću metode `middleware`. Radi jednostavnosti i praktičnosti, unutar klase `app/Http/Kernel.php` u podatkovnom članu `$middlewareAliases` pojedinom `middlewareu` se može nadjenuti *alias* ime [32] (pogledati ispis 22).

```

1 /**
2  * The application's middleware aliases.
3  *
4  * Aliases may be used to conveniently assign middleware to routes
5  * and groups.
6  *
7  * @var array<string, class-string|string>
8 */
9 protected $middlewareAliases = [
10     'auth' => \App\Http\Middleware\Authenticate::class,
11     'auth.basic' => \Illuminate\Auth\Middleware\
12     AuthenticateWithBasicAuth::class,
13     'auth.session' => \Illuminate\Session\Middleware\
14     AuthenticateSession::class,
15     'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::
16     class,
17     'can' => \Illuminate\Auth\Middleware\Authorize::class,
18     'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
19     'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword
```

```

16     ::class,
17         'signed' => \App\Http\Middleware\ValidateSignature::class,
18         'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::
19             class,
20                 'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::
21                     class,
22                     'locale' => \App\Http\Middleware\ConfigureLocale::class,
23             ];

```

Ispis 22: Definiranje *middleware aliasa*

Različite *middleware* klase može se međusobno grupirati pod istim ključem (engl. *key*) u prethodno navedenoj `Kernel` klasi koristeći podatkovni član `$middlewareGroups` [32] (pogledati ispis 23).

```

1      /**
2       * The application's route middleware groups.
3       *
4       * @var array<string, array<int, class-string|string>>
5       */
6
7 protected $middlewareGroups = [
8     'web' => [
9         \App\Http\Middleware\EncryptCookies::class,
10        \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::
11            class,
12            \Illuminate\Session\Middleware\StartSession::class,
13            \App\Http\Middleware\ConfigureLocale::class,
14            \Illuminate\View\Middleware\ShareErrorsFromSession::class,
15            \App\Http\Middleware\VerifyCsrfToken::class,
16            \Illuminate\Routing\Middleware\SubstituteBindings::class,
17        ],
18
19     'api' => [
20         // \Laravel\Sanctum\Http\Middleware\
21         EnsureFrontendRequestsAreStateful::class,
22         \Illuminate\Routing\Middleware\ThrottleRequests::class.':api'
23     ,
24         \Illuminate\Routing\Middleware\SubstituteBindings::class,

```

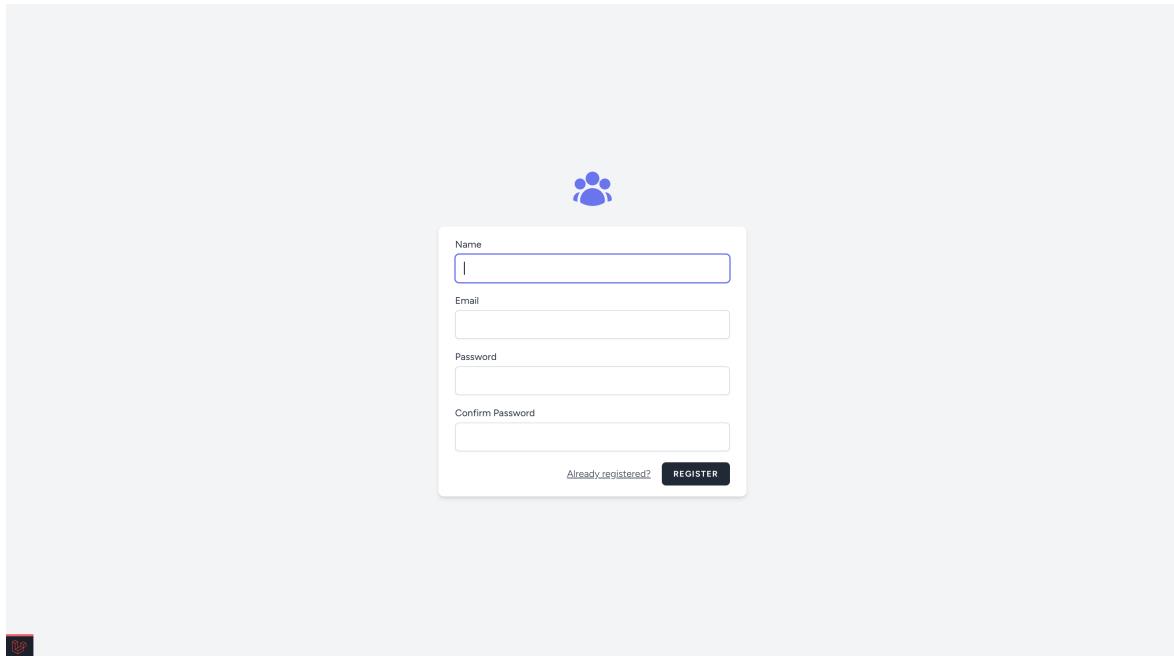
```
21      ] ,  
22      ] ;
```

Ispis 23: Definiranje *middleware* grupa

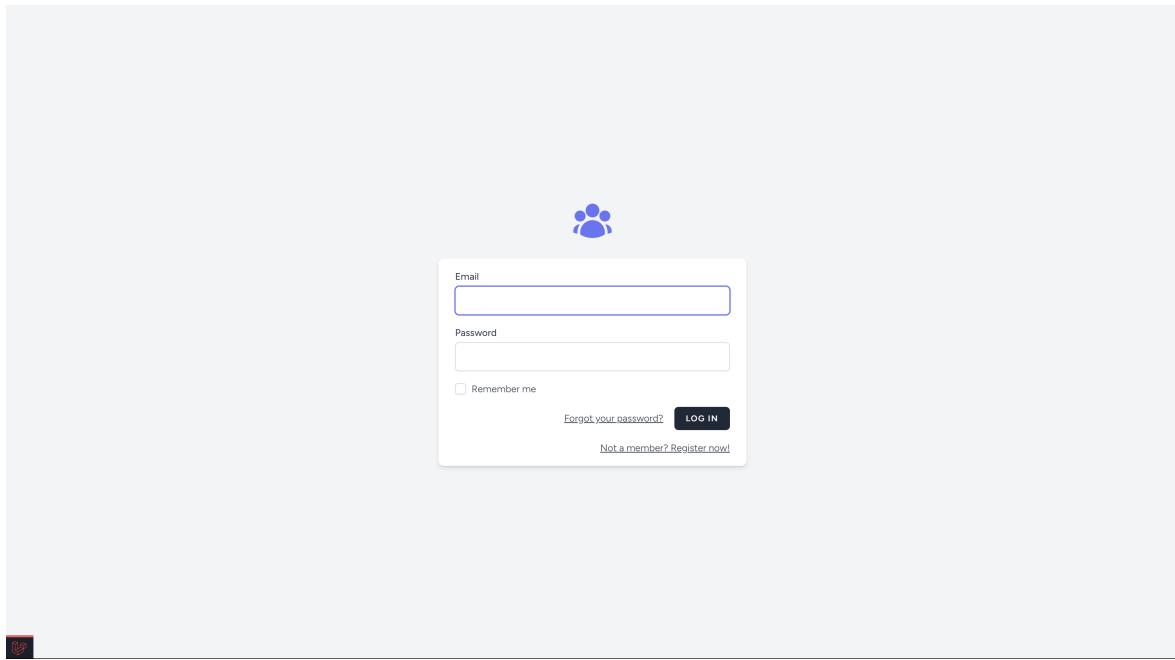
3.7. Autentikacija

Početni komplet Laravel Jetstream u potpunosti se pobrinuo za cjelokupan autentikacijski sustav aplikacije na način da koristi autentikacijske servise Laravel Fortifyja, uz to pruživši robustan i moderan *scaffolding* korisničkog sučelja. Laravel autentikacijskim servisima može se pristupiti putem fasade (engl. *facade*) Auth [33].

Na slici 4 prikazano je pristupanje ruti `register` unutar web preglednika, a na slici 5 ruti `login`.



Slika 4: /register



Slika 5: /login

3.8. Autorizacija

U Laravelu postoje dva osnovna načina na koji se autoriziraju akcije - koristeći vrata (engl. *gates*) ili koristeći politike (engl. *policies*). Vrata je najbolje primjenjivati za autorizaciju akcija koje nisu vezane uz model ili resurs već su općenite za cijelu aplikaciju, a politike su namijenjene za autorizaciju akcija nad određenim modelom/resursom [34].

3.8.1. Vrata

Vrata (engl. *gates*) određuju ima li korisnik ovlasti za izvršavanje određene radnje. Obično se definiraju unutar `boot` metode u klasi `App\Providers\AuthServiceProvider` koristeći fasadu `Gate` [34].

U ispisu 24 prikazana je `boot` metoda u kojoj se definiraju vrata 'admin-privileges' te se koristeći metodu `before` definira *closure* koji će biti provjeravan prije svih ostalih autorizacijskih provjera.

```
1  /**
2   * Register any authentication / authorization services.
3   */
4  public function boot(): void
```

```

5      {
6          $this->registerPolicies();
7
8          Gate::define('admin-privileges', function ($user) {
9              return $user->isAdministrator()
10             ? Response::allow()
11             : Response::deny(__('You must be an administrator.'));
12         });
13
14         Gate::before(function (User $user, string $ability) {
15             if ($user->isAdministrator()) {
16                 return true;
17             }
18         });
19     }

```

Ispis 24: Vrata korištena za implementaciju admin korisnika i administratorskog sučelja

3.8.2. Politike

Politike (engl. *policies*) su klase čija je zadaća organizacija autorizacijske logike pojedinog modela/resursa. Generiraju se pozivom Artisan naredbe `make:policy` te se nalaze unutar direktorija `app/Policies`. Kreirane politike potrebno je registrirati unutar podatkovnog člana `$policies` klase `App\Providers\AuthServiceProvider` (prikazano u ispisu 25) gdje se mapira pojedine Eloquent modele s odgovarajućom politikom [34].

```

1  /**
2   * The policy mappings for the application.
3   *
4   * @var array<class-string, class-string>
5   */
6
protected $policies = [
    Team::class => TeamPolicy::class,
    Project::class => ProjectPolicy::class,
    Post::class => PostPolicy::class,
    Comment::class => CommentPolicy::class,
    Resource::class => ResourcePolicy::class,
]

```

Ispis 25: Registriranje politika

Kao primjer politike u ispisu 26 prikazana je datoteka `PostPolicy.php` tj. politika koja definira autorizaciju akcija nad modelom `Post`. Tu je definirano da pojedinu objavu mogu vidjeti samo članovi tima kojem objava pripada, također da objavu može urediti i ažurirati samo onaj korisnik koji ju je i objavio, a ovlasti za izbrisati objavu imaju vlasnik tima, vlasnik/tvorac projekta kojem objava pripada te autor objave.

```

1 <?php
2
3 namespace App\Policies;
4
5 use App\Models\Post;
6 use App\Models\User;
7
8 class PostPolicy
9 {
10     /**
11      * Determine whether the user can view any models.
12      */
13     public function viewAny(User $user): bool
14     {
15         return true;
16     }
17
18     /**
19      * Determine whether the user can view the model.
20      */
21     public function view(User $user, Post $post): bool
22     {
23         return $user->belongsToTeam($post->project->team);
24     }
25
26     /**
27      * Determine whether the user can create models.

```

```

28     */
29     public function create(User $user): bool
30     {
31         return true;
32     }
33
34 /**
35 * Determine whether the user can update the model.
36 */
37     public function update(User $user, Post $post): bool
38     {
39         return $post->user->id == $user->id;
40     }
41
42 /**
43 * Determine whether the user can delete the model.
44 */
45     public function delete(User $user, Post $post): bool
46     {
47         return $user->ownsTeam($post->project->team) ||
48             $user->ownsProject($post->project) ||
49             $post->user->id == $user->id;
50     }
51
52 /**
53 * Determine whether the user can restore the model.
54 */
55     public function restore(User $user, Post $post): bool
56     {
57         return false;
58     }
59
60 /**
61 * Determine whether the user can permanently delete the model.
62 */
63     public function forceDelete(User $user, Post $post): bool
64     {
65         return false;
66     }

```

Ispis 26: Politika za autorizaciju akcija nad modelom Post

Za provjeru korisnikovih ovlasti za izvršenje pojedine akcije nad modelom/resursom koristi se metoda `authorize`, a unutar Blade predložaka koriste se još i direktive `@can`, `@cannot`, `@canany` [34].

3.9. Livewire komponente

Livewire je *full-stack* razvojni okvir za Laravel koji olakšava razvijanje dinamičkog korisničkog sučelja. Razvio ga je Caleb Porzio, otvorenog je kôda te je kôd dostupan na platformi GitHub [35]. Radi na način da se prvo inicijalno prikazuje Livewire komponenta na stranici, a zatim kada korisnik s njom interaktira Livewire šalje AJAX zahtjev prema serveru s ažuriranim podatcima. Server zatim iznova prikazuje tu komponentu i vraća novi HTML, a Livewire mutira DOM (engl. *Document Object Model*) tako da se učinjene promjene očitaju na stranici [36].

Livewire komponenta kreira se pozivom Artisan naredbe `make:livewire` te će se tada kreirati sljedeće dvije datoteke - u direktoriju `app/Http/Livewire` kreirat će se klasa Livewire komponente, a u direktoriju `resources/views/livewire.blade.php` datoteka Livewire komponente. U ispisu 27 prikazana je klasa Livewire komponente `ShowPost`.

```

1 <?php
2
3 namespace App\Http\Livewire\Posts;
4
5 // use ... -> required imports
6
7 class ShowPost extends Component
8 {
9     use AuthorizesRequests;
10    use InteractsWithBanner;
11
12    public Post $post;
13    public $postId;
```

```

14     public $title;
15     public $content;
16     public $openEditModal;
17     public $openDeleteModal;
18
19     protected $listeners = ['postUpdated' => '$refresh'];
20
21     protected $rules = [
22         'title' => 'required',
23         'content' => 'required',
24     ];
25
26     public function render()
27     {
28         return view('livewire.posts.show-post');
29     }
30
31     public function edit($id)
32     {
33         $post = Post::findOrFail($id);
34
35         $this->authorize('update', $post);
36
37         $this->postId = $id;
38         $this->title = $post->title;
39         $this->content = $post->content;
40
41         $this->openEditModal = true;
42     }
43
44     public function update()
45     {
46         $this->validate();
47
48         if ($this->postId) {
49             Post::find($this->postId)->update([
50                 'title' => $this->title,
51                 'content' => $this->content,
52             ]);

```

```

53
54     $this->emitSelf('postUpdated');
55     $this->banner(__('Post updated successfully.'));
56
57     $this->openEditModal = false;
58
59     $this->reset(['title', 'content']);
60 }
61 }
62
63 public function delete($id)
64 {
65     $post = Post::findOrFail($id);
66
67     $this->authorize('delete', $post);
68
69     $this->postId = $id;
70
71     $this->openDeleteModal = true;
72 }
73
74 public function destroy()
75 {
76     if ($this->postId) {
77         Comment::where('post_id', $this->postId)->delete();
78         Post::where('id', $this->postId)->delete();
79
80         $this->emit('postDeleted');
81         $this->banner(__('Post deleted successfully.'));
82
83         $this->openDeleteModal = false;
84     }
85 }
86 }
```

Ispis 27: ShowPost.php - klasa Livewire komponente

Livewire komponenta može biti uključena u Blade predložak koristeći direktivu @livewire.

Validacija atributa komponente u Livewireu obavlja se pozivom metode `$this->validate()` koja zatim provjerava pravila definirana unutar podatkovnog člana `$rules` za pojedini atribut [37].

Kako bi se unutar Livewire komponente koristila paginacija potrebno je da komponenta koristi osobinu (engl. *trait*) `WithPagination` [38].

3.9.1. Akcije

Akcije u Livewireu "osluškuju" interakcije na stranici te pozivaju odgovarajuću metodu na Livewire komponenti [39].

Tako, gledajući prethodni ispis, metode `edit`, `update`, `delete`, `destroy` u klasi Livewire komponente izvršavaju se nakon interakcije korisnika nad određenim elementom na stranici - primjerice korisnik interaktira s gumbom (engl. *button*) koji "osluškuje" određeni događaj putem direktive `wire` kao npr. `wire:click="edit"`, `wire:click="update"`, `wire:click="delete"`, `wire:click="destroy"` u pripadajućem Blade predlošku.

Kako bi se akcije u Livewireu mogle autorizirati potrebno je da Livewire komponenta koristi osobinu (engl. *trait*) `AuthorizesRequests` te je zatim moguće pozvati metodu `$this->authorize()` unutar Livewire komponente [40].

3.9.2. Događaji

Livewire komponente koje "žive" na istoj stranici međusobno mogu komunicirati putem globalnog sustava događaja [41].

Događaji se mogu pokrenuti na više načina: iz Blade predloška, iz komponente te iz globalnog JavaScripta pozivom neke od sljedećih metoda: `emit`, `emitUp`, `emitTo`, `emitSelf`, ovisno o `scope` unutar kojega će emitirani događaj biti dostupan [41].

Unutar klase Livewire komponente i to u podatkovnom članu `$listeners` potrebno je registrirati koje će događaje komponenta "slušati" te koju metodu će komponenta pozvati kad se određeni događaj emitira [41].

3.10. Pohrana datoteka

Laravel integrira Flysystem [42] PHP paket Franka de Jongea te tako pruža apstrakciju datotečnog sustava - nudi *drivere* za rad s lokalnim datotečnim sustavom, SFTP-om (engl. *Secure File Transfer Protocol*) te Amazon S3 pohranom objekata u oblaku [43].

Konfiguracijska datoteka datotečnog sustava u Laravelu jest `filesystems.php` unutar direktorija `config` te se u njoj konfiguriraju tzv. diskovi Laravel datotečnog sustava od kojih svaki predstavlja jedinstveni *driver* te lokaciju za pohranu. *Driver local* upravlja datotekama pohranjenima na lokalnom serveru na kojem se pokreće Laravel aplikacija, a *driver s3* koristi se za pohranu datoteka na Amazon S3 pohranu objekata u oblaku [43].

3.10.1. MinIO - AWS S3 kompatibilan servis za pohranu

U aplikaciji *Teamstructor* za pohranu timskih resursa korišten je AWS (Amazon Web Services) S3 kompatibilan servis za pohranu naziva **MinIO** [44]. On koristi `s3` disk u konfiguracijskoj datoteci.

Prije korištenja `s3` *drivera* potrebno je instalirati Flysystem S3 paket putem Composer pozivom naredbe: `composer require league/flysystem-aws-s3-v3 "^3.0" --with-all-dependencies` [43]. Također je potrebno dodati MinIO servis u `docker-compose.yml` datoteku kao što je prikazano u ispisu 28 te definirati potrebne *environment* varijable unutar `.env` datoteke kao u ispisu 29.

```
1 version: "3.9"
2 services:
3   # ...
4
5   minio:
6     image: minio/minio
7     ports:
8       - 9000:9000
9       - 9001:9001
10    environment:
11      MINIO_ROOT_USER: minioadmin
12      MINIO_ROOT_PASSWORD: minioadmin
13    volumes:
```

```

14      - minio-data:/data
15      command: server /data --console-address :9001
16
17      # ...
18
19 volumes:
20      # ...
21      minio-data:

```

Ispis 28: minio servis unutar docker-compose.yml datoteke

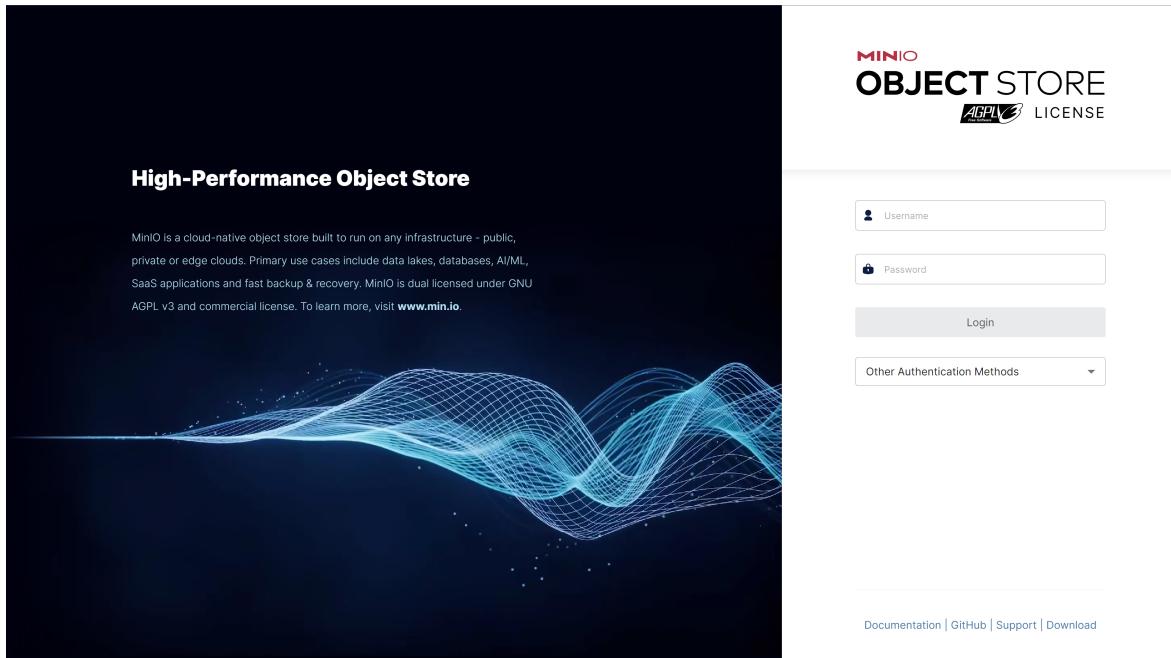
```

1 AWS_ACCESS_KEY_ID=XXXXXXXXXXXXXXXXXXXX
2 AWS_SECRET_ACCESS_KEY=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
3 AWS_DEFAULT_REGION=eu-south-1
4 AWS_BUCKET=teamstructor-bucket
5 AWS_URL=http://teamstructor.test:9000/teamstructor-bucket
6 AWS_ENDPOINT=http://minio:9000
7 AWS_USE_PATH_STYLE_ENDPOINT=true

```

Ispis 29: Definiranje potrebnih AWS/MinIO *environment* varijabli

Pristupajući URL-u `http://localhost:9000` unutar web preglednika prikazat će se MinIO Object Store zaslon za prijavu kao na slici 6, a uspješnom prijavom koristeći korisničko ime i lozinku definirane za `minio` Docker spremnik prikazat će se Object Browser zaslon prikazan na slici 7 na kojemu je moguće pregledati i pristupiti pojedinim *bucketima* [45].



Slika 6: MinIO Console - MinIO Object Store *login* zaslon

The screenshot shows the MinIO Object Browser page. The left sidebar is identical to the login page, showing the "Object Browser" link is selected. The main content area is titled "Object Browser" and displays a table of buckets. The table has columns: "Name", "Objects", "Size", and "Access". There is one entry: "teamstructor-bucket" with 150 objects, a size of 765.0 KiB, and R/W access. Above the table is a search bar with a magnifying glass icon and a refresh/circular arrow icon. The entire interface has a dark theme.

Slika 7: MinIO Console - MinIO Object Browser zaslon i stvorenji *bucket*

3.11. Spatie Laravel Media Library paket

Za implementaciju timskih resursa korišten je Laravel Media Library ([spatie/laravel-medialibrary](#)) paket razvijen od strane Spatie tima - belgijske *development* agencije koja razvija mnogo poznatih paketa otvorenog kôda za PHP i Laravel ekosustav.

Media Library paket instalira se putem Composera naredbom `composer require "spatie/laravel-medialibrary:^10.0.0"` te je potom potrebno objaviti migraciju za kreiranje tablice media pomoću Artisan naredbe `vendor:publish` te primijeniti tu migraciju na bazu podataka [46].

Za ostvariti željenu vezu između modela Project i Resource potrebno je da model Project implementira *interface* HasMedia i osobinu (engl. *trait*) `InteractsWithMedia`, a model Resource nasljeđuje bazni model Media iz Media Library paketa. Također je potrebno u konfiguracijskoj datoteci `config/media-library.php` navesti naziv klase *custom* medijskog modela kao u ispisu 30.

```
1      /*
2       * The fully qualified class name of the media model.
3       */
4      'media_model' => App\Models\Resource::class,
```

Ispis 30: Specificiranje naziva klase *media* modela u `config/media-library.php`

U ispisu 31 vidljiv je dio kôda zaslužan za asociranje učitanog resursa s pripadajućim projektom te pohranjivanje učitanog resursa na s3 disk. U prikazanoj metodi `save` iz klase Livewire komponente `UploadResource` prvo se validira *input* tj. učitana datoteka, zatim se koristeći metodu `addMedia` učitani resurs asocira s pripadajućim projektom. Pozivom metode `toMediaCollection` resurs se pohranjuje na s3 disk u zadalu medijsku kolekciju, a pozivom metode `withCustomProperties` na resurs se dodaje i *custom* svojstvo `'user_id'` gdje spremamo identifikator korisnika koji je izvršio *upload* te datoteke.

```
1      public function save()
2      {
3          $this->validate([
4              'resource' => 'required|file|max:12288', // (12MB max)
5          ]);
6
7          $this->project
8              ->addMedia($this->resource)
```

```

9         ->withCustomProperties(['user_id' => Auth::user()->id])
10        ->toMediaCollection('default', 's3');
11
12        $this->emit('resourceAdded');
13        $this->banner(__('Resource uploaded successfully.'));
14
15        $this->reset(['resource']);
16    }

```

Ispis 31: Asociranje resursa s pripadajućim projektom i njegovo pohranjivanje koristeći Media Library metode

3.12. Lokalizacija

Zadano Laravel aplikacija ne sadrži direktorij lang u kojem bi bile pohranjene potrebne jezične datoteke, ali ga se može stvoriti izvođenjem Artisan naredbe lang:publish [47].

Lokalizacija *Teamstructor* aplikacije izvedena je na način da su podržani engleski i hrvatski jezik što je jasno naznačeno u datoteci config/locales.php čiji je sadržaj prikazan u ispisu 32.

```

1 <?php
2
3 return [
4     /*
5     | -----
6     | Locales
7     | -----
8     |
9     | List of application locales (locale_code => language_name).
10    |
11    */
12
13    'en' => 'English',
14    'hr' => 'Hrvatski',
15];

```

Ispis 32: Sadržaj datoteke config/locales.php

Jezične JSON datoteke s prevedenim *stringovima* nalaze se u aplikacijskom direktoriju resources/lang:

```
resources/lang
├── en.json
└── hr.json
```

Kako bi se doista u Blade predlošku i prikazali odgovarajući prevedeni stringovi koristi se *echo* sintaksa {{ }} i *helper* funkcija — npr. {{ __('Welcome to your Teamstructor application!') }} [47].

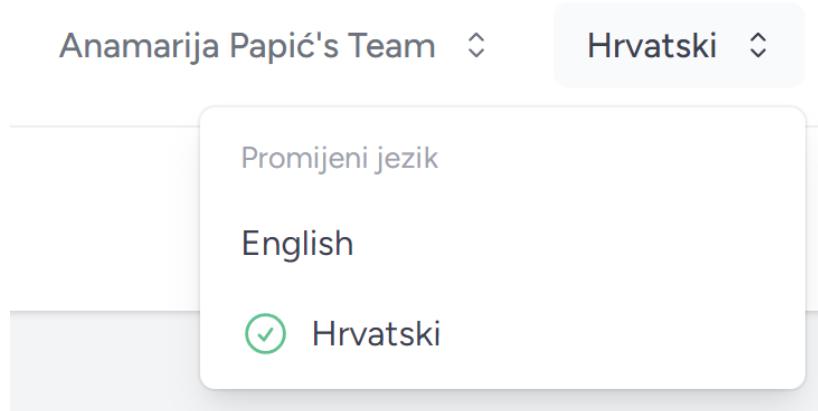
Implementiran je LanguageController prikazan u ispisu 33, dodana ruta za promjenu trenutnog jezika (prikazano u ispisu 20 u liniji 56), dodan ConfigureLocale middleware prikazan u ispisu 21.

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Support\Facades\Redirect;
6 use Illuminate\Support\Facades\Session;
7
8 class LanguageController extends Controller
9 {
10     public function switchLocale($locale)
11     {
12         if (array_key_exists($locale, config('locales'))) {
13             Session::put('locale', $locale);
14         }
15
16         return Redirect::back();
17     }
18 }
```

Ispis 33: LanguageController.php

U konfiguracijsku datoteku za NGINX server default.conf dodana je linija
rewrite ^/(en|hr) /?(.*)? https://teamstructor.test/\$2?locale=\$1
permanent; te u .htaccess datoteku linija
RedirectMatch "^(en|hr) /?(.*)?" "/\$2?locale=\$1" što omogućava da
se u web pregledniku URL unesen u formi https://teamstructor.test/hr tumači
kao https://teamstructor.test/?locale=hr.

Naposlijetku je implementiran i *language switcher* tj. gumb za promjenu jezika prikazan na slici 8.



Slika 8: Gumb u navigacijskoj traci koji služi za promjenu trenutnog jezika aplikacije

3.13. Testiranje i kvaliteta kôda

3.13.1. Korišteni alati

Pri *developmentu* radi bržeg otkrivanja i otklanjanja pogrešaka te praćenja performansi i ponašanja u aplikaciji korišteni su paketi **Laravel Debug Bar** [48] i **Laravel Telescope** [49].

Također je korišten i *code style fixer* **Laravel Pint** [50], izgrađen na PHP-CS-Fixeru, koji je automatski dostupan za korištenje uz Laravel instalaciju. Pomaže da kôd ostane "čist" i konzistentan te se popravak *code stylea* inicira izvođenjem naredbe
.vendor/bin/pint iz terminala.

Radi pretpregleda testne elektroničke pošte u aplikaciji (poslane i primljene) dodan je servis **MailHog** [51] u datoteku docker-compose.yml (prikazano u ispisu 34) te se MailHog web sučelju može pristupiti unosom URL-a http://localhost:8025 u web

preglednik.

```
1 mailhog:
2   platform: linux/x86_64
3   image: mailhog/mailhog
4   ports:
5     - 1025:1025 # smtp server
6     - 8025:8025 # web ui
```

Ispis 34: mailhog servis unutar docker-compose.yml datoteke

3.13.2. Pisanje i pokretanje testova

U aplikacijskom `test` direktoriju zadano se nalaze dva poddirektorija - `Feature` i `Unit`. *Unit* testovi testiraju mali izolirani dio kôda, a *feature* testovi testiraju veće dijelove kôda tj. pojedine funkcionalnosti u cijelosti [52].

Kako se testiranje ne bi provodilo nad glavnom "radnom" bazom podataka i tako se vršile promjene nad stvarnim zapisima u bazi, za potrebe testiranja definirano je posebno izolirano `sqlite_testing` okruženje baze podataka u memoriji (prikazano u ispisu 35).

```
1   'sqlite_testing' => [
2     'driver' => 'sqlite',
3     'database' => ':memory;',
4   ],
```

Ispis 35: Testno okruženje baze podataka definirano u config/database.php

Laravel automatski dolazi s podrškom za testiranje s **PHPUnit** te se novi *feature* test može kreirati pozivom Artisan naredbe `make:test`, a testovi se mogu pokrenuti koristeći `phpunit` naredbu `./vendor/bin/phpunit` ili Artisan naredbu `test` [52].

U ispisu 36 prikazan je primjer *feature* testa pisanog u PHPUnitu.

```
1 <?php
2
```

```

3 namespace Tests\Feature;
4
5 // use ... -> imports
6
7 class CreateCommentFormTest extends TestCase
8 {
9     use RefreshDatabase;
10
11     public function test_the_component_can_render(): void
12     {
13         $this->actingAs($user = User::factory()->withPersonalTeam()->
14         create());
15
16         $project = Project::factory()->create([
17             'team_id' => $user->currentTeam,
18             'user_id' => $user,
19         ]);
20
21         $post = Post::factory()->create([
22             'project_id' => $project,
23             'user_id' => $user,
24         ]);
25
26         $component = Livewire::test(CreateCommentForm::class, ['post' =>
27             $post]);
28
29         $component->assertStatus(200);
30     }
31 }

```

Ispis 36: CreateCommentFormTest - test_the_component_can_render

Pest PHP razvojno okruženje za testiranje potrebno je instalirati putem Composera:
 composer require pestphp/pest --dev --with-all-dependencies,
 a zatim inicijalizirati Pest u trenutnom projektu koristeći naredbu
 ./vendor/bin/pest --init [53].

Moguće je instalirati Pest dodatke (engl. *plugins*) za Laravel i Livewire putem Com-

```
posera: composer require pestphp/pest-plugin-laravel --dev
composer require pestphp/pest-plugin-livewire --dev [54].
```

Testovi se nakon instalacije Pesta mogu pokretati naredbom ./vendor/bin/pest, a Pest test može se kreirati pozivom Artisan naredbe pest:test [53].

U ispisu 37 prikazan je primjer *feature* testa pisanog u Pestu.

```
1 <?php
2
3 // use ... -> imports
4
5 uses() -> group('feature', 'resources');
6
7 uses(RefreshDatabase::class);
8
9 test('the component can render', function () {
10     actingAs($user = User::factory() -> withPersonalTeam() -> create());
11
12     $project = Project::factory() -> create([
13         'team_id' => $user -> currentTeam,
14         'user_id' => $user,
15     ]);
16
17     livewire(UploadResource::class, ['project' => $project])
18         -> assertStatus(200);
19 }) ;
```

Ispis 37: UploadResourceTest - test 'the component can render'

3.14. Otvoreni kôd i doprinos zajednice na projektima Laravel ekosustava

4. Prezentacija korisničkog sučelja pri korištenju aplikacije

5. Zaključak

U ovom tekstu dan je pregled izrade završnog rada. Na temelju iznesenog, studenti bi trebali izraditi korektno sastavljeni završni rad. Naravno, najteži dio izrade je sadržaj rada, koji studenti moraju sami osmisliti. Ovaj tekst pomoći će im da izbjegnu zamke plagiranja i da uoče vrijednost samostalnog i autorskog rada.

Literatura

- [1] <https://github.com/laravel/laravel>, posjećeno 12.6.2023.
- [2] Laravel LLC, “Laravel - The PHP Framework For Web Artisans,” <https://laravel.com/>, posjećeno 12.6.2023.
- [3] ——, “Laravel Documentation - Artisan Console,” <https://laravel.com/docs/10.x/artisan>, posjećeno 12.6.2023.
- [4] ——, “Laravel Documentation - Release Notes,” <https://laravel.com/docs/10.x/releases>, posjećeno 12.6.2023.
- [5] Docker Inc., “Docker Docs - Try Docker Compose,” <https://docs.docker.com/compose/gettingstarted/>, posjećeno 14.6.2023.
- [6] ——, “Docker Docs - Volumes,” <https://docs.docker.com/storage/volumes/>, posjećeno 14.6.2023.
- [7] Laravel LLC, “Laravel Documentation - Deployment,” <https://laravel.com/docs/10.x/deployment>, posjećeno 14.6.2023.
- [8] NGINX - Part of F5, Inc., “NGINX Docs - Configuring NGINX and NGINX Plus as a Web Server,” <https://docs.nginx.com/nginx/admin-guide/web-server/web-server/>, posjećeno 14.6.2023.
- [9] <https://github.com/php/php-src>, posjećeno 14.6.2023.
- [10] S. Brekalo, *Uvod u PHP programiranje*. Međimursko vеleučilište u Čakovcu, 2018., https://www.mev.hr/wp-content/uploads/2019/01/Uvod_u_PHP_programiranje.pdf, posjećeno 14.6.2023.
- [11] The PHP Group, “php.net - Supported Versions,” <https://www.php.net/supported-versions.php>, posjećeno 14.6.2023.
- [12] npm, Inc., “npm Docs - About npm,” <https://docs.npmjs.com/about-npm>, posjećeno 14.6.2023.
- [13] Composer, “Composer Documentation - Getting Started,” <https://getcomposer.org/doc/00-intro.md>, posjećeno 14.6.2023.

- [14] ——, “Composer Documentation - Basic Usage,” <https://getcomposer.org/doc/01-basic-usage.md>, posjećeno 14.6.2023.
- [15] Laravel LLC, “Laravel Documentation - Starter Kits,” <https://laravel.com/docs/10.x/starter-kits>, posjećeno 16.6.2023.
- [16] “Laravel Jetstream - Introduction,” <https://jetstream.laravel.com/3.x/introduction.html>, posjećeno 16.6.2023.
- [17] “Laravel Jetstream - Installation,” <https://jetstream.laravel.com/3.x/installation.html>, posjećeno 16.6.2023.
- [18] “Laravel Jetstream - Livewire,” <https://jetstream.laravel.com/3.x/stacks/livewire.html>, posjećeno 16.6.2023.
- [19] Laravel LLC, “Laravel Documentation - Configuration,” <https://laravel.com/docs/10.x/configuration>, posjećeno 16.6.2023.
- [20] ——, “Laravel Documentation - Directory Structure,” <https://laravel.com/docs/10.x/structure>, posjećeno 16.6.2023.
- [21] “Laravel Jetstream - Concept Overview: Actions,” <https://jetstream.laravel.com/2.x/concept-overview.html#actions>, posjećeno 16.6.2023.
- [22] “Laravel Jetstream - Concept Overview: Layouts,” <https://jetstream.laravel.com/2.x/concept-overview.html#layouts>, posjećeno 16.6.2023.
- [23] Laravel LLC, “Laravel Documentation - Database: Migrations,” <https://laravel.com/docs/10.x/migrations>, posjećeno 18.6.2023.
- [24] ——, “Laravel Documentation - Database: Seeding,” <https://laravel.com/docs/10.x/seeding>, posjećeno 19.6.2023.
- [25] ——, “Laravel Documentation - Eloquent: Getting Started,” <https://laravel.com/docs/10.x/eloquent>, posjećeno 19.6.2023.
- [26] ——, “Laravel Documentation - Eloquent: Factories,” <https://laravel.com/docs/10.x/eloquent-factories>, posjećeno 20.6.2023.
- [27] <https://github.com/FakerPHP/Faker>, posjećeno 20.6.2023.
- [28] Laravel LLC, “Laravel Documentation - Eloquent: Relationships,” <https://laravel.com/>

docs/10.x/eloquent-relationships, posjećeno 21.6.2023.

[29] ——, “Laravel Documentation - Eloquent: Collections,” <https://laravel.com/docs/10.x/eloquent-collections>, posjećeno 21.6.2023.

[30] ——, “Laravel Documentation - Collections,” <https://laravel.com/docs/10.x/collections>, posjećeno 21.6.2023.

[31] ——, “Laravel Documentation - Routing,” <https://laravel.com/docs/10.x/routing>, posjećeno 26.6.2023.

[32] ——, “Laravel Documentation - Middleware,” <https://laravel.com/docs/10.x/middleware>, posjećeno 26.6.2023.

[33] ——, “Laravel Documentation - Authentication,” <https://laravel.com/docs/10.x/authentication>, posjećeno 27.6.2023.

[34] ——, “Laravel Documentation - Authorization,” <https://laravel.com/docs/10.x/authorization>, posjećeno 27.6.2023.

[35] <https://github.com/livewire/livewire>, posjećeno 27.6.2023.

[36] Livewire, “Laravel Livewire,” <https://laravel-livewire.com/>, posjećeno 27.6.2023.

[37] ——, “Laravel Livewire - Validation,” <https://laravel-livewire.com/docs/2.x/input-validation>, posjećeno 27.6.2023.

[38] ——, “Laravel Livewire - Pagination,” <https://laravel-livewire.com/docs/2.x/pagination>, posjećeno 27.6.2023.

[39] ——, “Laravel Livewire - Actions,” <https://laravel-livewire.com/docs/2.x/actions>, posjećeno 27.6.2023.

[40] ——, “Laravel Livewire - Authorization,” <https://laravel-livewire.com/docs/2.x/authorization>, posjećeno 27.6.2023.

[41] ——, “Laravel Livewire - Events,” <https://laravel-livewire.com/docs/2.x/events>, posjećeno 27.6.2023.

[42] <https://github.com/theophileleague/flysystem>, posjećeno 28.6.2023.

[43] Laravel LLC, “Laravel Documentation - File Storage,” <https://laravel.com/docs/10.x/file-storage>

filesystem, posjećeno 28.6.2023.

- [44] MinIO, Inc., “MinIO - High Performance Object Storage for AI,” <https://min.io/>, posjećeno 28.6.2023.
- [45] ——, “MinIO Documentation - MinIO Object Storage for Container,” <https://min.io/docs/minio/container/index.html>, posjećeno 28.6.2023.
- [46] Spatie, “Docs - Laravel-medialibrary - Introduction,” <https://spatie.be/docs/laravel-medialibrary/v10/introduction>, posjećeno 28.6.2023.
- [47] Laravel LLC, “Laravel Documentation - Localization,” <https://laravel.com/docs/10.x/localization>, posjećeno 28.6.2023.
- [48] <https://github.com/barryvdh/laravel-debugbar>, posjećeno 28.6.2023.
- [49] Laravel LLC, “Laravel Documentation - Laravel Telescope,” <https://laravel.com/docs/10.x/telescope>, posjećeno 28.6.2023.
- [50] ——, “Laravel Documentation - Laravel Pint,” <https://laravel.com/docs/10.x/pint>, posjećeno 28.6.2023.
- [51] <https://github.com/mailhog/MailHog>, posjećeno 28.6.2023.
- [52] Laravel LLC, “Laravel Documentation - Testing: Getting Started,” <https://laravel.com/docs/10.x/testing>, posjećeno 28.6.2023.
- [53] Pest, “Pest Docs - Installation,” <https://pestphp.com/docs/installation>, posjećeno 28.6.2023.
- [54] ——, “Pest Docs - Plugins,” <https://pestphp.com/docs/plugins>, posjećeno 28.6.2023.

Dodatci

Popis slika

1	Sadržaj app direktorija	10
2	Dijagram relacijske baze podataka	11
3	Izlist svih ruta definiranih u aplikaciji pomoću Artisan naredbe <code>route:list</code>	31
4	<code>/register</code>	35
5	<code>/login</code>	36
6	MinIO Console - MinIO Object Store <code>login</code> zaslon	46
7	MinIO Console - MinIO Object Browser zaslon i stvoreni <code>bucket</code>	46
8	Gumb u navigacijskoj traci koji služi za promjenu trenutnog jezika aplikacije	50

Popis tablica

1	Nazivi klasa modela i pripadajuće tablice	16
---	---	----

Popis ispisa kôda

1	Naredbe za instalaciju Jetstream paketa u novi Laravel projekt	7
2	Definicija varijable u <code>.env</code> datoteci	8
3	Pristup vrijednosti <i>environment</i> varijable u Laravelu	8
4	Migracija za kreiranje tablice <code>projects</code>	13
5	Migracija za dodavanje stupca <code>is_administrator</code> u tablicu <code>users</code>	14
6	Sadržaj <code>DatabaseSeeder</code> klase	15
7	Dio kôda model klase <code>User</code>	17
8	Dio kôda tvornice modela <code>UserFactory</code>	19
9	Dio kôda tvornice modela <code>UserFactory</code> koji prikazuje definiciju stanja tvornice modela	20
10	Relacije modela <code>User</code>	21
11	Relacije modela <code>User</code> koje koristi iz osobine <code>HasTeams</code>	21
12	Relacije modela <code>Team</code>	22

13	Relacije modela Team koje nasljeđuje od modela JetstreamTeam	23
14	Relacije modela TeamInvitation	24
15	Relacije modela Project	24
16	Relacije modela Project koje koristi iz osobine InteractsWithMedia	26
17	Relacije modela Post	26
18	Relacije modela Comment	27
19	Relacije modela Resource koje nasljeđuje od modela Media	27
20	Sadržaj datoteke routes/web.php	28
21	<i>Middleware</i> ConfigureLocale	32
22	Definiranje <i>middleware aliasa</i>	33
23	Definiranje <i>middleware</i> grupa	34
24	Vrata korištena za implementaciju admin korisnika i administratorskog sučelja	36
25	Registriranje politika	37
26	Politika za autorizaciju akcija nad modelom Post	38
27	ShowPost.php - klasa Livewire komponente	40
28	minio servis unutar docker-compose.yml datoteke	44
29	Definiranje potrebnih AWS/MinIO <i>environment</i> varijabli	45
30	Specificiranje naziva klase media modela u config/media-library.php	47
31	Asociranje resursa s pripadajućim projektom i njegovo pohranjivanje koristeći Media Library metode	47
32	Sadržaj datoteke config/locales.php	48
33	LanguageController.php	49
34	mailhog servis unutar docker-compose.yml datoteke	51
35	Testno okruženje baze podataka definirano u config/database.php .	51
36	CreateCommentFormTest - test_the_component_can_render	51
37	UploadResourceTest - test 'the component can render'	53