Web aplikacija za vođenje evidencije godišnjih odmora zaposlenika

Azenić, Borna

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

Web aplikacija za vođenje evidencije godišnjih odmora zaposlenika

Diplomski rad

Borna Azenić

Osijek, 2020.

Contents

1. UVOD

Interakcija između poslodavaca i zaposlenika u njihovim tvrtkama u današnje je vrijeme svakodnevna. Bilo da je riječ o temama koje se tiču poslovanja ili o temama koje su malo manje vezane za budućnost tvrtke, interakcija je sveprisutna. Socijalni kontakt potrebno je ostvarivati na dnevnoj bazi, a samo sretan, zadovoljan i odmoran radnik može pridonijeti boljoj budućnosti tvrtke.

Gotovo svaki zaposlenik raduje se godišnjem odmoru, koji definitivno ima svoju svrhu, a vašem poslovanju puno će više pomoći odmorna te pouzdana osoba, nego menadžer koji se zbog straha od neobavljenog posla na neki način boji privremeno delegirati radne aktivnosti. S obzirom na prethodno navedene činjenice, uopće ne treba iznenaditi potreba za izradom aplikacije u sklopu ovog diplomskog rada.

Tema diplomskog rada izrada je web aplikacije za vođenje evidencije godišnjih odmora zaposlenika. Izrađena je jednostavna te fluidna aplikacija koja će automatizirati odlazak na godišnji odmor. Diplomski je rad strukturiran na način da su prvo opisane tehnologije potrebne za uspješno funkcioniranje ove web aplikacije. U sklopu ovog dijela predstavljeno je i programsko okruženje u kojem je programirano. Potom, slijedi detaljnije objašnjenje same implementacije svih funkcionalnosti koje aplikacija sadrži. Ovdje će biti opisan front-end aplikacije, kao i back-end. Nakon sistematike, slijedi nam testiranje aplikacije u sklopu kojega će biti provedene provjere uspješnosti prethodno implementiranih funkcionalnosti, kao i prikaz rezultata testiranja.

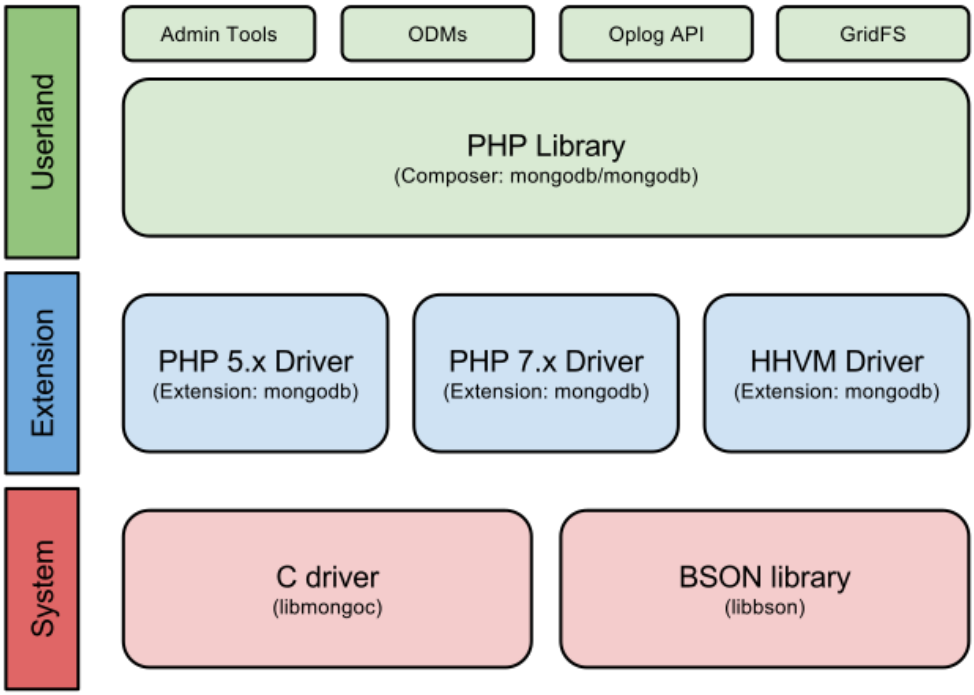
U slučaju apstraktnosti uvoda, za lakše razumijevanje koncepta razmišljanja, kao i same potrebe za aplikacijom, preporučuje se detaljnije proučavanje završnog rada na temu „C# desktop aplikacija za generiranje i pohranu zahtjeva za godišnji odmor“. Spomenuti rad, istog autora, poslužio je kao čvrst temelj za razvoj ovoga rada te je uvelike olakšao razumijevanje godišnjih odmora, plaćenih dopusta i sl.

2. KORIŠTENE TEHNOLOGIJE I OKRUŽENJE

2.1. *PHP*

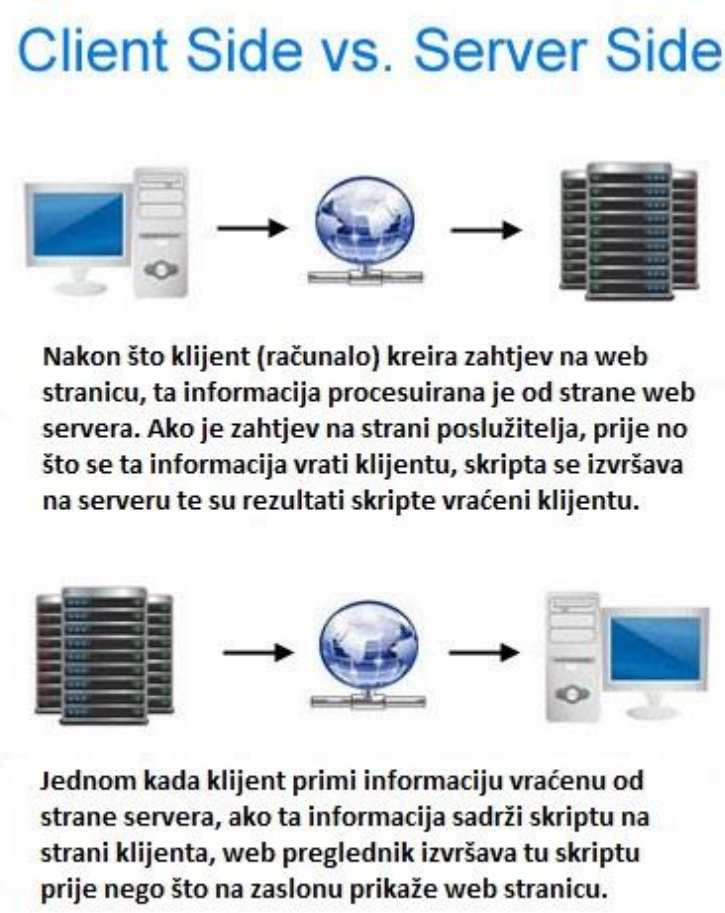
Povijest *PHP* programskog jezika usko je vezana za razvoj *World Wide Web-a*. Iako *HTML* slovi za jezik koji je omogućio „internet“, razvoj *PHP-a* zapravo je učinio web stranice interaktivnima kakve su danas. Pojava *PHP-a* uvelike je doprinjela povećanju popularnosti interneta jer bi, bez pojave istog, internet ostao „samo“ jedan veliki izvor podataka. Ovako, danas je to način za komunikaciju zbog svoje interaktivne prirode.

PHP je kreiran sredinom devedesetih godina kako bi bilo olakšano kreiranje dinamičkih web stranica. Originalno, prijevod krilatice *PHP* bio je „osobna početna stranica“, a bio je napisan od strane programera Rasmusa Lerdorfa kako bi lakše ažurirao stranicu na kojoj je bio postavljen njegov životopis. Osim toga, htio je znati koliko se dnevnog prometa odvija na njegovoj stranici. Ispostavilo se da *PHP* pruža brojne prednosti u odnosu na *HTML* te su poduzeti određeni koraci kako bi se od skripte napravio kompletan programski jezik. Korištenjem *HTML-a* za kreiranje web stranica zapravo imamo statičnu stranicu, odnosno, svaki korisnik će svakim posjetom stranici vidjeti isti sadržaj (osim ako administrator ručno ne promijeni sadržaj). Pojavom dinamičkih web stranica, sadržaj se može mijenjati prilikom svakog novog posjeta. S gledišta internet oglašavanja ovo je od velike važnosti jer omogućava ponovne posjete. Upravo zbog ove funkcionalnosti, *PHP* vrlo brzo dobiva na svojoj popularnosti i cijeni. Programski jezik koji se lako i brzo uči, jednostavan je za upotrebu, moćan, brz te se koristi u više od pola svih stranica na webu. Kako bi snaga ovog jezika bila bolje dočarana, jedna od najpopularnijih stranica danas, posebno kod mlađe populacije, Facebook, razvijena je uz pomoć upravo *PHP-a* [1,2]. Grafički prikaz arhitekture *PHP-a* prikazan je na slici 2.1.



Sl. 2.1. *PHP* arhitektura [3]

Razvojem *PHP-a* 3, gdje je *PHP* ponovno napisan iz temelja te su uklonjeni problemi s parsiranjem, olakšano je za druge programere „proširiti“ jezik, a osim toga, omogućeno je dodavanje njihovih osobnih modula. Jezik je dobio ograničenu objektno-orijentiranu podršku, što je samo rasplamsalo razvoj, rast i učestalost korištenja *PHP* programskog jezika. Iako je originalno bio kreiran za lakši i brži razvoj dinamičkih web stranica, *PHP* se danas puno više koristi za skriptiranje na strani poslužitelja. Razlika između skriptiranja na strani poslužitelja i klijenta prikazana je na slici 2.2.



Sl. 2.2. Skriptiranje na strani poslužitelja [4]

Skriptiranje na strani poslužitelja tehnika je najčešće korištena kod razvoja web stranica, odnosno aplikacija, koja uključuje korištenje skripti na web serveru koje šalju odgovor podešen za svaki korisnikov zahtjev poslan na web stranicu. Alternativa je, kao što je već spomenuto, statička web stranica. Skriptiranje na strani poslužitelja omogućava vlasniku web stranice da sakrije izvorni kod koji generira korisničko sučelje, dok kod skriptiranja na strani klijenta korisnik ima pristup cijelom kodu koji je poslan od strane klijenta. Negativna strana ovakvog pristupa je da klijent mora slati daljnje zahtjeve preko mreže na server kako bi nove informacije bile prikazane korisniku u web pregledniku [5].

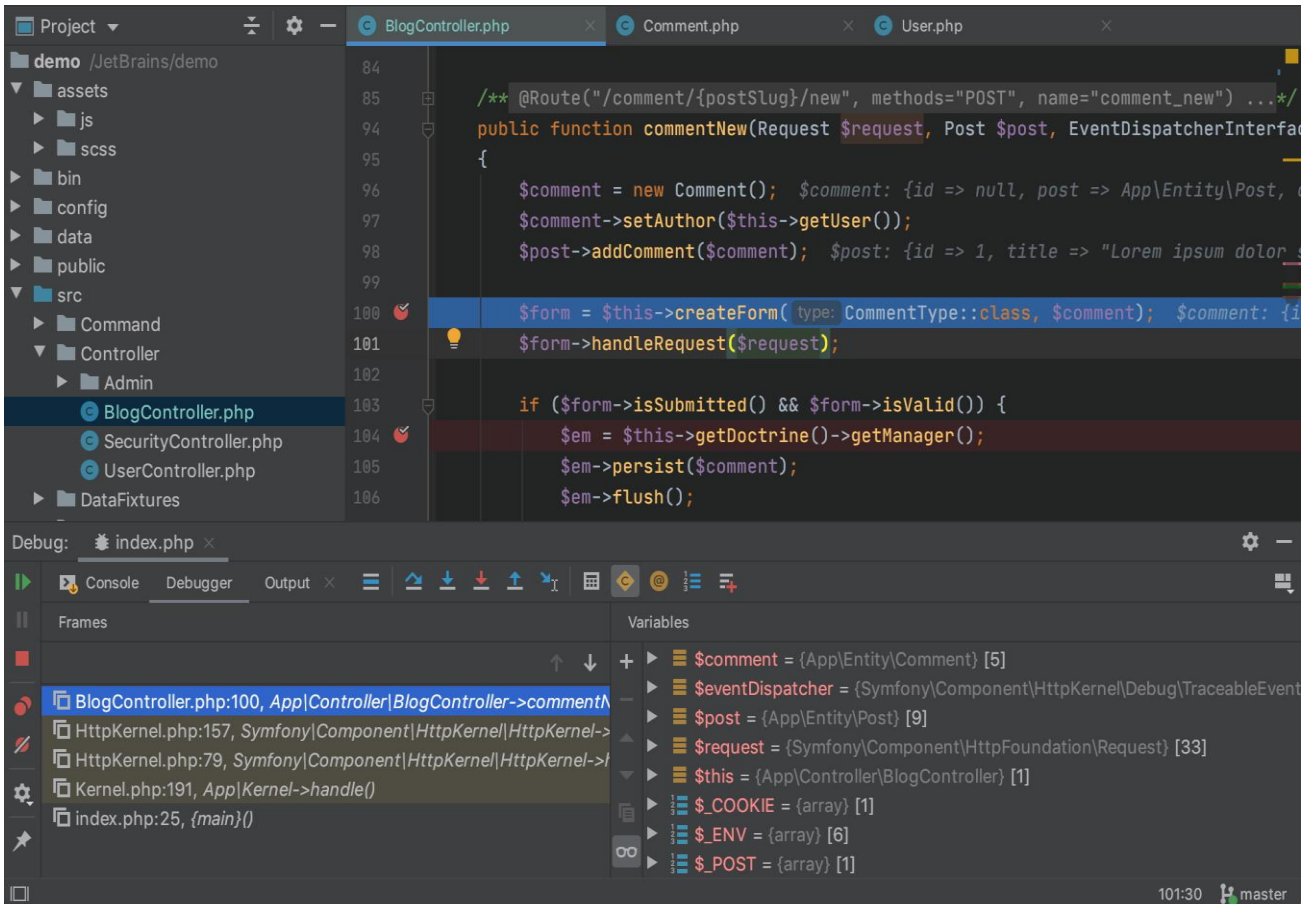
2.1.1. *PhpStorm*

Danas, biti dobar *software developer* zahtijeva duboko znanje programskih jezika u kojima radite. Ako su vaši svakodnevni zadaci primarno razvoj web aplikacija, odnosno stranica,

najvjerojatnije ćete morati biti stručnjak u *PHP-u* i *MySQL-u*. Međutim, samo znanje programskih jezika nije dovoljno. Dobro snalaženje u dopunskim alatima koje koristite za isti taj razvoj jako je važno. Morate biti vješti u upravljanju izvornim kodom, servisi upravljanja virtualnim strojem moraju vam biti u malom prstu te, očigledno, vaše integrirano razvojno okruženje mora biti prilagođeno vašim afinitetima.

PhpStorm komercijalna je, *cross* platforma integriranog razvojnog okruženja ponajprije za *PHP* programski jezik. Inicijalni razvoj dogodio se krajem 2009. godine od strane češke kompanije *JetBrains*. *PhpStorm* pruža svojim korisnicima alate za *PHP*, *HTML* te *JavaScript* u obliku *on-the-fly* analize koda te prevencije grešaka. Pisan je u *Java* programskom jeziku, a korisnici mogu vrlo lako proširiti razvojno okruženje instalirajući razne dodatke koji su originalno kreirani za *PhpStorm*, ili pak mogu sami napisati svoje dodatke. Softver komunicira s vanjskim izvorima kao što je primjerice *Xdebug*.

Brojne su prednosti korištenja upravo ovog razvojnog okruženja te je prisutna podrška za razne programske okvire kao što su *Symfony*, *Laravel*, *Drupal*, *WordPress*, *Zend Framework*, *Magento*, *Joomla!*, *CakePHP* i mnogi drugi. Jedan je od rijetkih editora koji zapravo „razumiju“ strukturu koda, pritom pružajući podršku za sve moderne i naslijeđene projekte. Pruža veliki broj olakšanja u radu kao što su kompletiranje koda, *refactoring* koda, integracija *version controla* i *command prompt-a*, *debuging*... Izgled projekta u *PhpStormu* prikazan je na slici 2.3.



Sl. 2.3. Primjer projekta u *PhpStormu*

2.2. *MySQL*

SQL (***S**tructured **Q**uery **L**anguage*) jezik je korišten za provođenje operacija nad bazom podataka. Radi se o osnovnom jeziku koji se koristi kod gotovo svih baza podataka. Postoje minorne razlike u sintaksi kod drukčijih baza podataka, ali generalno gledano, sintaksa je identična i vrlo intuitivna. *SQL* se koristi u pristupu, ažuriranju kao i u manipulaciji s podacima u radu s bazama podataka. Dizajn jezika dozvoljava upravljanje i sustavima koji se koriste kod baza podataka, kao što je primjerice *MySQL*. Programeri ga često upotrebljavaju i za kreiranje te brže mijenjanje sheme baze podataka (EER dijagrami i sl.).

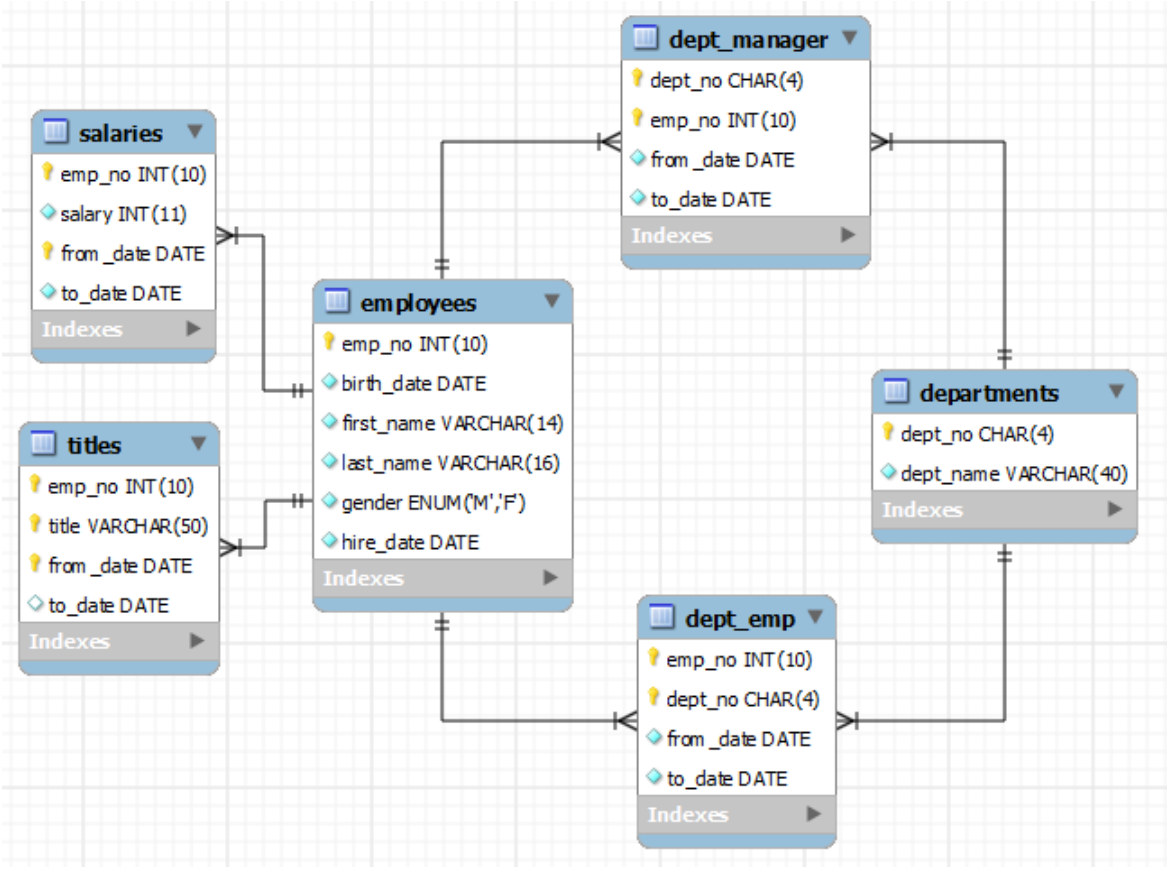
S druge pak strane, *MySQL* jedan je od najpopularnijih sustava za upravljanje bazama podataka. To je aplikacijska podrška pomoću koje se bezbolnije izgrađuje i obrađuje baza podataka. Većina sustava u današnje vrijeme zasniva svoj rad na relacijskom modelu uvedenom od strane E. F. Codd-a, a osim relacijskog modela, postoji još i sustav baziran na objektnom modelu te stari *ISAM* sustav. Radi se o besplatnom, *open source* sustavu koji je, uz *PostgreSQL*

čest izbor baze za projekte otvorenog koda, a osim toga, obično je distribuiran kao sastavni dio Linux distribucija. Naravno, postoje i inačice za ostale operacijske sustave poput Windowsa, macOS-a i drugih. Na samom početku svojeg razvoja, *MySQL* se suočavao s brojnim kritikama zbog nedostatka osnovnih funkcija prethodno definiranih *SQL* standardom. Sustav funkcionira na način da je optimiziran kako bi bio brz, pod cijenu lošije funkcionalnosti. Unatoč tome, iznimno je stabilan, moduli i ekstenzije su dobro dokumentirani te ima podršku brojnih programskih jezika poput *PHP-a, Jave, Python-a* i drugih. Baze su relacijskog tipa, koji se kroz povijest etablirao kao najbolji način za skladištenje i rad s većim količinama podataka. Takve baze, odnosno sustavi, predstavljaju temelj poslovnih subjekata koji u današnje vrijeme svoje poslovanje najčešće baziraju na brzoj dostupnosti kvalitenih i provjerenih informacija.

ACID načelo jedno je od temeljnih pravila koja moraju biti zadovoljena u radu s relacijskim bazama podataka, a *MySQL* poštuje upravo to načelo pri izvođenju transakcija i operacija nad podacima. Kratica *ACID* (engl. *atomicity*, *consistency*, *isolation*, *durability*) u računalnim znanostima odnosi se na skup pravila koja garantiraju valjanost podataka čak i u slučaju slijeda grešaka, nestanka struje, kvarova i sl. Kratka definicija transakcije bila bi da je to niz slijednih operacija nad bazom podataka, koji se iz ugla promatrača mogu gledati kao jedna logička operacija nad podacima. Theo Haerder i Andreas Reuter osmislili su ovaj akronim još davne 1983. godine, a nezaobilazan je i danas. Detaljniji opis pravila prikazan je na slici 2.4.

Sl. 2.4. *ACID* načelo

Prije početka rada s bilo kojim sustavom za upravljanje bazama podataka, pa tako i *MySQL-om*, potrebno je detaljno osmisliti odgovarajući izgled baze podataka. To zapravo znači napraviti shemu baze, koja će kasnije biti prikazana u obliku odgovarajućeg broja tablica koje se koriste za pohranu podataka. Elementi koji se pohranjuju u bazu podataka nazivaju se entiteti, koji predstavljaju stvari iz svakodnevnog života za koje se informacije pohranjuju. Uz entitet, drugi vrlo važan pojam koji se učestalo koristi jest relacija. Između različitih entiteta postoje određeni odnosi, takozvane relacije koje se na specifičan način predstavljaju unutar samog sustava. Relacije mogu biti jedan prema jedan, jedan prema više te više prema više. Podaci su unutar tablica pohranjeni u obliku kolona i redova [6]. Primjer kreirane sheme baze podataka nalazi se na slici 2.5.



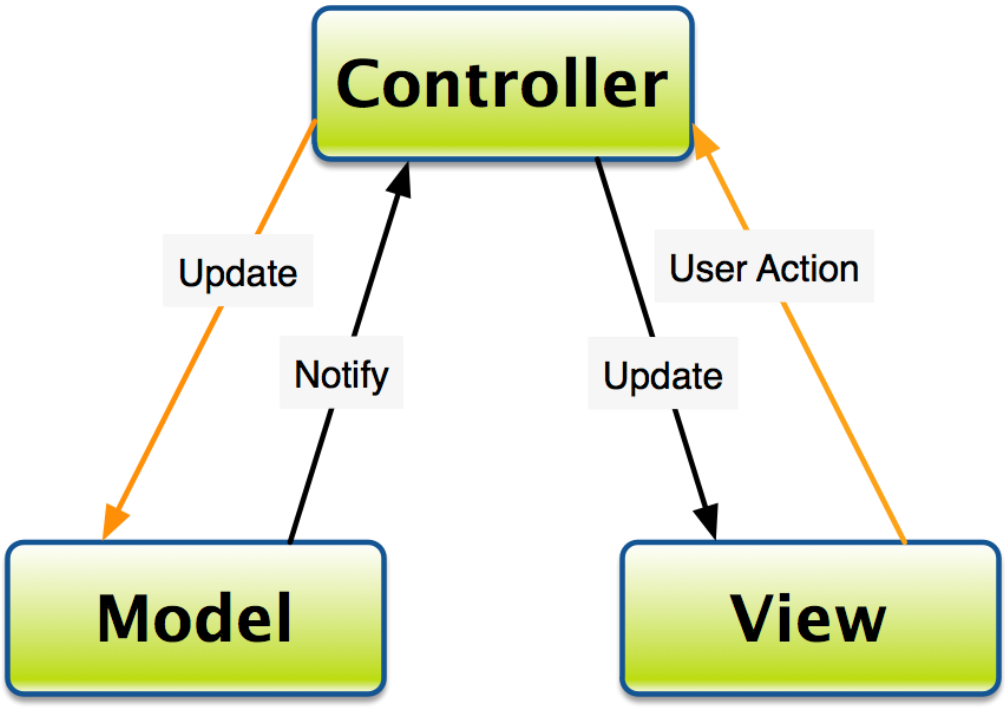
Sl. 2.5. Primjer EER modela baze podataka [7]

2.3. *Laravel*

Jedan od najpopularnijih i najčešće korištenih programskih okvira danas. *Laravel* je zasnovan na *MVC* (*model – view - controller*) arhitekturi, a razvijen je od strane Taylora Otwella krajem 2011. godine. Programski okvir ima javno dostupan izvorni kod koji se nalazi na GitHub-u te je licenciran od strane MIT-a. Taylor Otwell kreirao je *Laravel* pokušavajući zapravo razviti napredniju verziju *CodeIgniter-a*, koji nije imao ugrađene funkcionalnosti za autentikaciju te autorizaciju korisnika.

Model – view – controller obrazac je softverske arhitekture korišten za razvoj korisničkih sučelja koja će podijeliti aplikaciju na više manjih komponenti koje su međusobno povezane. Svaka ova komponenta pritom postoji kako bi lakše rukovala sa specifičnim problemima koje aplikacija ima. *MVC* jedan je najčešće korištenih industrijski standardiziranih obrazaca za razvoj skalabilnih i lako proširivih projekata. Tradicionalno upotrebljavan za desktop grafička korisnička sučelja, danas je postao popularan za dizajniranje web aplikacija. *PHP*, *JavaScript*, *Python*, *Ruby*, *C#* te *Swift* samo su neki od programskih jezika zasnovani na *MVC* arhitekturi. Kao što sam naziv

može sugerirati, sastoji se od tri komponente: modela, pogleda i kontrolora. Model je centralna komponenta obrasca, to je dinamička struktura podataka, neovisna o korisničkom sučelju aplikacije direktno zadužena za manevriranje podacima, logikom kao i pravilima same aplikacije. Pogled je bilo koji prikaz informacija kao što su primjerice dijagrami, grafikoni ili tablice. Višestruki pogledi na iste informacije su naravno mogući, a primjer toga imamo u stupčanom grafikonu menadžmenta s jedne strane te tabličnom prikazu istih podataka za računovođe. Kontrolor prihvaća ulazne podatke i pretvara ih u naredbe za model, odnosno pogled. Kontrolor je zapravo sučelje između modela i pogleda. Ovakva arhitektura uvelike olakšava posao programerima na način da ubrzava neovisan razvoj, testiranje, ali i održavanje željene aplikacije [8]. Shematski prikaz ove arhitekture prikazan je na slici 2.6.



Sl. 2.6. Shema *MVC* arhitekture [9]

U *software* industriji, modularnost se odnosi na podjelu nekog velikog sustava ili aplikacijskih modula na manje komponente koje se kasnije opet mogu spojiti i upotrijebiti. Kod modularnosti, poslovna logika može biti podijeljena na više dijelova od kojih svaki pripada jedinstvenom entitetu. *Laravel* omogućuje modularni pristup u obliku paketa. Autentifikacija je jedna od najvažnjih funkcionalnosti gotovo svake web aplikacije, a programerima puno dragocjenog vremena odlazi na pisanje koda za implementaciju autentifikacije. *Laravel* pruža

svojim korisnicima jednostavnu, ugrađenu autentifikaciju koja vrlo lako može biti integrirana u web aplikaciji koristeći se sa svega nekoliko jednostavnih *artisan* naredbi. Važna prednost korištenja *Laravel* programskog okvira je i poprilično jednostavno korištenje *cache* memorije. *Cache* memorija koristi se za kratkotrajnu pohranu podataka koji vrlo brzo mogu biti dohvaćeni, a ta se memorija koristi kako bi opterećenje servera bilo smanjeno. *Laravel* sadrži jedinstveni *API* pomoću kojega programeri uspostavljaju interakciju s raznom *cache* memorijom. *Memcached* i *Redis* lako su konfigurirani uz pomoć *Laravela*. Stavka koja se lako zanemari, a iznimno je važna za aplikaciju je sigurnost. *Laravel* koristi *salt* i *hash* mehanizme prilikom spremanja lozinke kako ona nikad ne bi bila dostupna kao čisti tekst u bazi podataka. Osim toga, koristi i „*Bcrypt Hashing*“ algoritam za generiranje enkriptirane lozinke. Dodatno, ovaj programski okvir koristi predefinirane *SQL* naredbe koje sprječavaju napade na bazu podataka [8].

2.4. *Vue.js*

Vue.js JavaScript je programski okvir najčešće upotrebljavan za lakši dizajn korisničkog sučelja, ali i za *single-page* aplikacije. *Laravel* kao back-end te *Vue.js* kao front-end savršena su kombinacija za bilo koji web projekt. Glavna prednost ovog *JavaScript* programskog okvira je u njegovoj mogućnosti da se kreiraju kompletne i skalabilne aplikacije koje će svojim korisnicima pružiti komponente koje će se lako uskladiti jedna s drugom, a izuzev te činjenice, brzina te odlične performanse samo su neke od odlika ovog programskog okvira. Integracija ovih dvaju programskih okvira na zavidnoj je razini, a o popularnosti spomenutog programskog okvira dovoljno svjedoči njegovih 148 tisuća zvjezdica na GitHubu, što samo po sebi doprinosi njegovom kredibilitetu. Iznenađujuće je koliko glatko može biti integriran u ogromne projekte te pojednostaviti sam proces razvoja web aplikacije [10].

Virtualni predmetni model dokumenta (engl. *DOM* – ***D****ocument* ***O****bject* ***M****odel*) jedna je od najvažnijih funkcionalnosti *Vue.js-a*. Bilo kakve promjene nisu direktno prikazane u *DOM-u* već je kreirana kopija u obliku *Javascript* strukture podataka. Ako su potrebne neke promjene, one su tada prikazane u *Javascriptu* upravo te kreirane strukture podataka. Potom je uspoređena s inicijalnom strukturom podataka, što samo po sebi uštedi jako puno vremena. Povezivanje podataka sljedeća je od funkcionalnosti, a radi se o procesu u kojemu korisnik može manipulirati nad elementima web stranice koristeći web preglednik. Koristi dinamički *HTML* te ne zahtijeva nikakve kompleksne skripte niti programiranje. Pomoću ove funkcionalnosti, lako je manevrirati

vrijednostima ili ih dodijeliti *HTML* atributima. Sve ovo, a i mnogo više, u kombinaciji s *Laravelom* idealno je za kreiranje ove web aplikacije [10].

2.5. *Bootstrap*

Besplatan programski okvir, s javno dostupnim izvornim kodom koji je usmjeren na responzivan front-end razvoj web stranica i aplikacija. Najčešće sadrži predloške bazirane na *CSS-u* te *JavaScriptu* za tipografiju, forme, navigaciju i ostale komponente koje se nalaze u sučelju. *Bootstrap* je sedmi po redu projekt na stranici GitHub, gdje ima preko 142 000 zvjezdica, te je zajedno s *Vue.js* programskim okvirom, jedan od najčešće korištenih u Sjedinjenim Američkm državama.

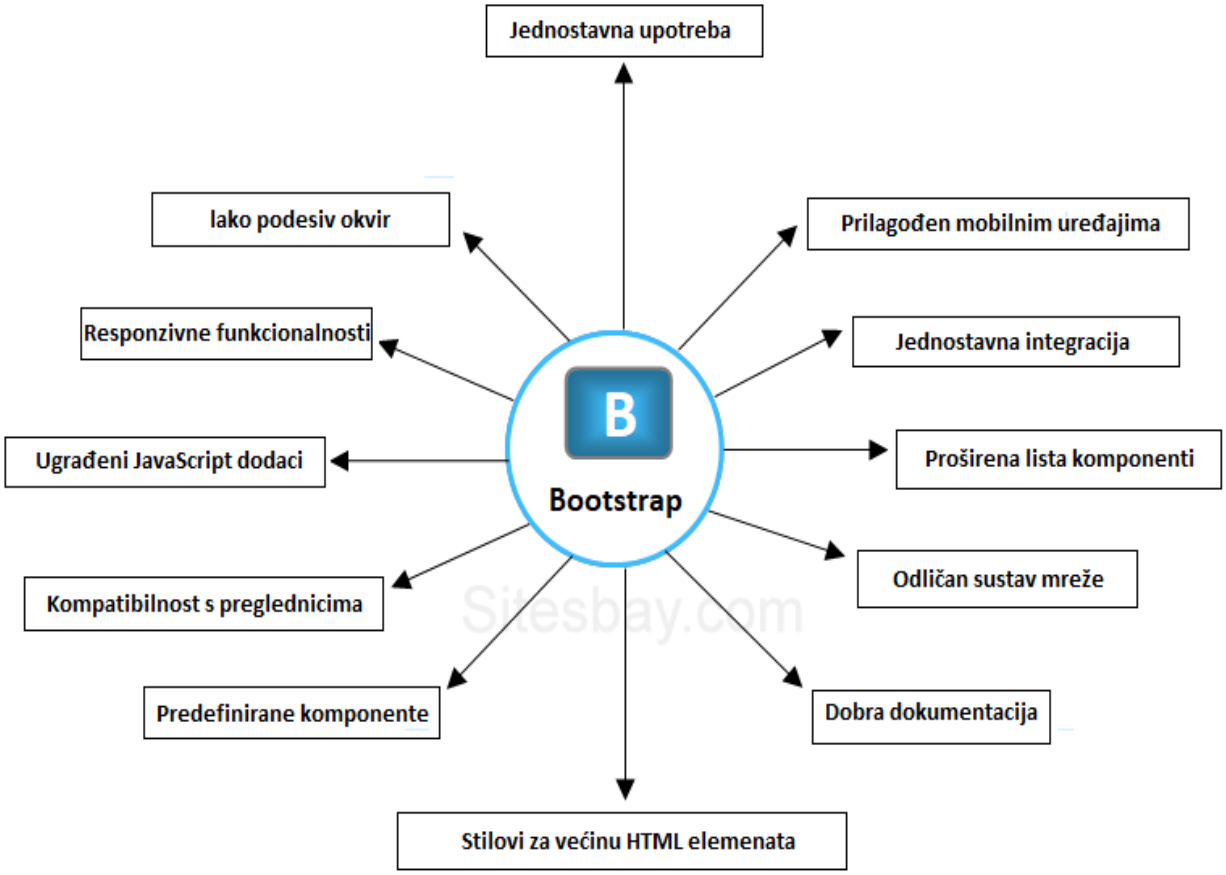
Bootstrap, prvotno imenovan *Twitter Blueprint*, razvijali su Mark Otto i Jacob Thornton unutar, zaključno prema imenu, tvrtke Twitter gdje su pokušali razviti programski okvir koji će osigurati dosljednost među unutarnjim alatima koji su se koristili na dnevnoj bazi. Prije razvoja *Bootstrapa*, mnoge biblioteke upotrebljavane su za razvoj korisničkog sučelja, što je dovodilo do učestalih nedosljednosti, a zatim i do problema visokog opterećenja. Mark Otto složio se s manjom grupom programera unutar tvrtke oko ideje da će dizajnirati i razviti novi alat koji će poslužiti kao odličan i brz način za podjelu zajedničkih dizajnerskih obrazaca unutar kompanije.

Pokrenut 2011. godine, *Bootstrap* je postepeno evoluirao u vrlo moćno te pouzdano rješenje koje programeri koriste kod raznih web stranica diljem svijeta, ali i u iOS te Android mobilnim aplikacijama. Usporedbe radi, ovaj programski okvir može zamijeniti tim od nekolicine programera te UI/UX dizajnera koji bi skupo naplatili svoje usluge razvoja običnog predloška od samog početka. Ovako, *Bootstrap* predstavlja veliku uštedu po pitanju vremena, ali i novca. Jednom kada donesete odluku koje veće funkcionalnosti vaša buduća stranica treba sadržavati, možete početi s odabirom odgovarajuće teme idealne za projekt [11, 12]. Preporuka je svakako proučiti glavne značajke ovog programskog okvira, od kojih treba istaknuti:

- Jednostavan početak – vrlo je lako započeti s *Bootstrapom*, čak i ako prethodno niste imali puno iskustva s programiranjem.
- Moćna skalabilnost – opremljen odlično osmišljenim i visoko responzivnim sustavom mreže od 12 stupaca, *Bootstrap* omogućava stranicama prilagodbu rezoluciji zaslona, ovisno o uređaju koji koristite. Shodno tome, iznimno je lako prikazati ili sakriti sadržaj po potrebi.

- Responzivne strukture i stilovi – uzimajući u obzir sve veći porast potrebe korisnika za pristupom web stranicama s mobilnih uređaja te tableta, važno je da stranica izgleda jednako dobro na različitim rezolucijama zaslona. *Bootstrap* organizira sve elemente zaslona u odnosu na dostupan prostor.
- Jednostavna integracija – ovaj programski okvir neće stvarati probleme u povezivanju s drugim programskim okvirima ili platformama na vašoj web stranici.
- Podudarnost s web preglednicima – *Bootstrap* je u potpunosti kompatibilan s najnovijim verzijama desktop i mobilnih web preglednika, što znači da će web stranica jednako dobro raditi neovisno o tome koristite li *Firefox, Chrome, Safari* ili pak *Internet Explorer*.
- Prilagođavanje – velik je odabir prilagodbi koje se mogu odabrati neovisno o projektu na kojem radite, što je jedan od glavnih razloga zašto se programeri većinom odlučuju koristiti upravo ovaj programski okvir. Dolazi s kompletno upakiranim komponentama koje su potrebne za razvoj profesionalnog dizajna. Međutim, važno je naglasiti da prilikom odabira odgovarajućeg predloška budu uklonjene značajke koje kasnije neće biti potrebne.
- Tehnička podrška – *Bootstrap* kao programski okvir nudi velik broj ažuriranja ponajprije zahvaljujući potpori ogromne zajednice koja ga koristi u svom radu. Budući je pohranjen na GitHubu, velik je broj onih koji svojim savjetima, ali i iskustvima pomažu kada god dođe do problema. Odlična dokumentacija također je prisutna s primjerima i demo verzijama što uvelike olakšava posao početnicima.
- Širok izbor predefiniranih elemenata – kod dizajniranja web stranica postoje neki elementi kojima je potrebno posvetiti malo više pažnje, kao što su recimo zaglavlje, tablice, tipkala, linkovi i slično. Svi ovi elementi imaju važnu ulogu u kreiranju web stranice, a Bootstrap pokriva sve ove opcije koje su lake za razumjeti i još lakše za upotrijebiti.
- Mnoštvo dodataka – ponuđen je velik broj dodataka s kojima je početak bezbolan. Ako su potrebne funkcionalnosti kao primjerice padajući izbornik, klizači, karusel i mnogi drugi, samo je potrebno integrirati ih unutar dizajna web stranice [13].

Grafički, sve gore navedene funkcionalnosti prikazane su na slici 2.7.



Sl. 2.7. Glavne funkcionalnosti *Bootstrapa*

3. PROGRAMSKO OSTVARENJE WEB APLIKACIJE

3.1. Baza podataka

Važan korak u izradi bilo koje web aplikacije predstavlja uspješno posložena i dizajnirana baza podataka. Za potrebe ove web aplikacije korišten je *MySQL*, verzija 5.7. Podaci unutar baze podataka trebaju konstantno biti dostupni klijentu, odnosno korisniku, a za bilo kakve promjene nad istim tim podacima koriste se odgovarajući alati. Baza podataka u ovom radu temelji se na relacijskom modelu koji je zapravo zasnovan na pojmu relacije i podrazumijeva da su entiteti i veze između istih tih entiteta prikazani tablicama.

Kao što je već u teorijskom dijelu rada napomenuto, *Laravel* funkcionira na temeljima *MVC* arhitekture. Najvažniji dio te arhitekture koji se veže uz bazu podataka je model. Podaci se iz baze podataka učitavaju u memoriju gdje su oni prikazani upravo pomoću modela. Za potrebe ove aplikacije kreirano je nekoliko modela, a to su *Company*, *User*, *Vacation* i *Location*. Unutar *User* modela važno je spomenuti *location()* metodu koja nam zapravo pomaže u odnosu između korisnika i lokacije, odnosno adrese. Središnji model aplikacije predstavlja *Vacation* čija će definicija biti malo detaljnije objašnjena. U modelu su prvo definirana polja koja se mogu ispunjavati (engl. *fillable*), a datumi koji se isto tako koriste, parsirani su od strane samog *Laravela*. Oni se u bazi podataka zapisuju u *timestamp* formatu, a potom se iz stringa pretvaraju u datum kako bi se nad njima lakše provodile odgovarajuće operacije i akcije. Definicija klase za ovaj model prikazana je na slici 3.1.

```php
class Vacation extends Model
{
    protected $table = 'vacations';
    protected $dates = ['start_date', 'end_date'];
    protected $fillable = [
        'user_id',
        'start_date',
        'end_date',
        'type',
        'company_id',
        'request_pdf',
        'approval_pdf',
    ];
```

Sl. 3.1. Definicija klase *Vacation* unutar modela

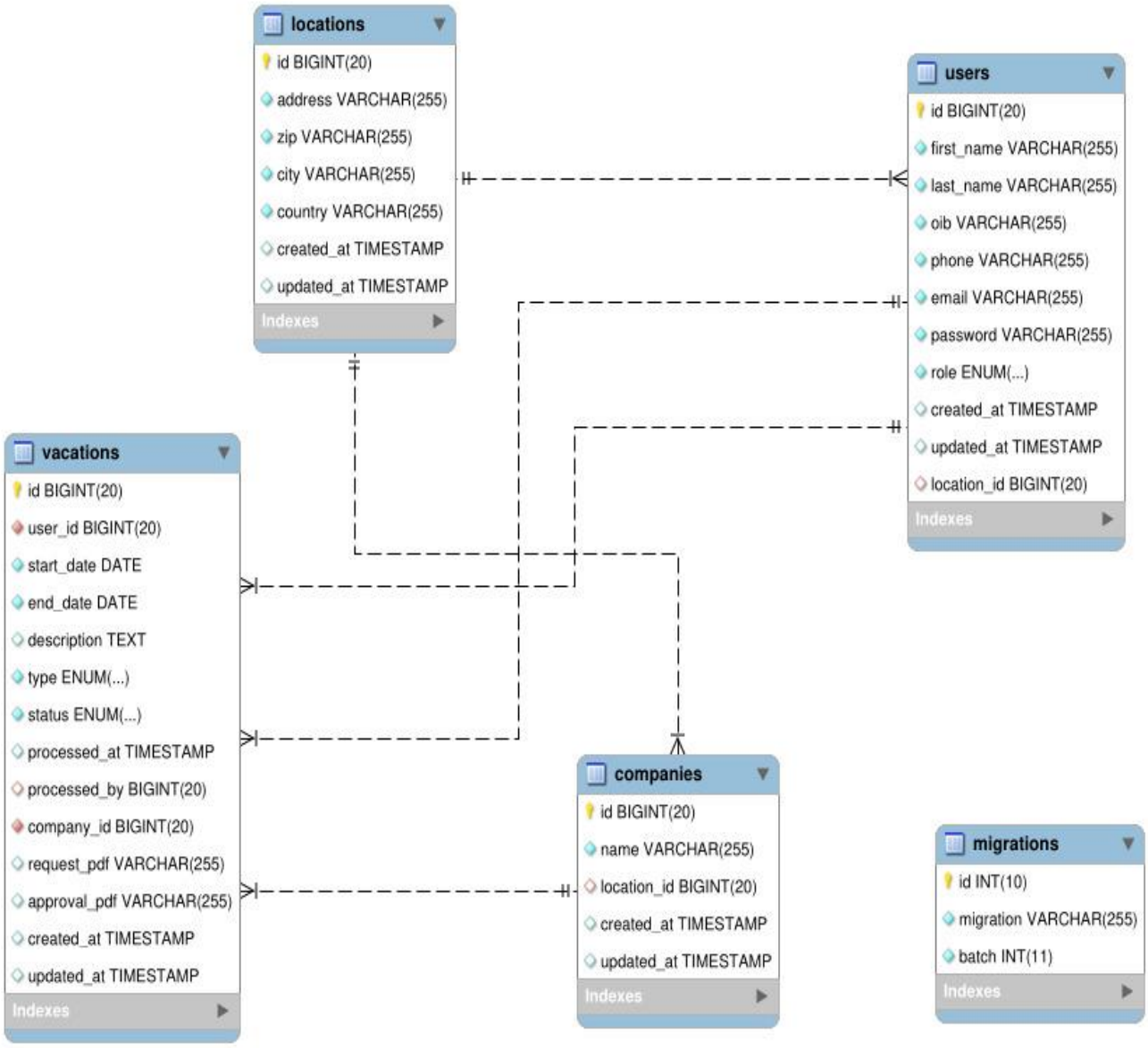
Laravel obično kontrolira rad baze podataka pomoću migracija. Migracije se koriste za kreiranje struktura ili tablica, a vrlo su korisne jer je pomoću njih vrlo jednostavno „migrirati“, odnosno prebaciti tablicu direktno u bazu podataka koristeći jednu jedinu naredbu. Za kreiranje migracije, koristi se *make:migration* naredba unutar *artisan* konzolnog sučelja. Novo kreirana migracija smještena je unutar *database/migrations* direktorija, a svaka migracija obično sadrži i datum, što omogućava Laravelu da lakše odredi redoslijed kreiranih migracija. Migracije su strukturirane na način da svaka klasa sadrži dvije glavne metode, *up* i *down* metodu. *Up* metoda koristi se za dodavanje novih tablica, stupaca ili indeksa u bazu podataka, dok *down* metoda treba obrnuti operacije koje je provela *up* metoda. Unutar obje ove metode preporuča se korištenje *Laravel schema buildera* za brže kreiranje i modifikaciju tablica. Primjer ovih dvaju metoda prikazan je na slici 3.2. Za kreiranje nove tablice baze podataka koristi se *create* metoda koja prima dva argumenta, prvi je ime tablice, a drugi je nacrt objekta koji kasnije može biti iskorišten za lakšu definiciju nove tablice.

```
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('first_name');
            $table->string('last_name');
            $table->string('oib');
            $table->string('phone');
            $table->string('email')->unique();
            $table->string('password');
            $table->enum('role', ['user', 'admin'])->default('user');
            $table->rememberToken();
            $table->timestamps();
        });
    }

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

Sl. 3.2. *Up* i *down* metode

Na slici iznad, osim spomenutih metoda, prikazan je i postupak kreiranja tablice *users*, gdje je metodi *create* predano ime tablice te nacrt objekta. Potom su definirani stupci tablice poput imena, prezimena, osobnog identifikacijskog broja, telefona, maila, lozinke, tipa korisnika i slično. Grafički prikaz kreiranih tablica koje su potrebne za bazu podataka nalazi se na slici 3.3., a ovaj EER dijagram kreiran je u programu *MySQL Workbench* 8.0., gdje je vrlo jednostavno dodati već postojeću bazu, napraviti konekciju na server, a program tada izrađuje dijagram na kojemu se jasno vide atributi i entiteti, te veze među pojedinim tablicama.



Sl. 3.3. EER dijagram baze podataka

3.2. Back-end

U ovom poglavlju rada bit će opisana implementacija back-enda, odnosno oni dijelovi koji su zaduženi za ispravno funkcioniranje osnovne logike same web aplikacije.

3.2.1. Prijava i registracija

Osnovni dio gotovo svake web aplikacije su registracija i prijava u sustav, a tako je bilo i u izradi ove. Prilikom definicije korisnika, bilo je potrebno kreirati model u bazi u kojem će se nalaziti parametri te kontrolor koji je zapravo zadužen za samu logiku registracije, prijave, ali i procesa koji se odvijaju u pozadini prilikom spremanja podataka u bazu. Kontrolori zaduženi za ove procese su *RegisterController* te *LoginController*, kreirani uz pomoć *Laravela*. Kontrolor zadužen za registraciju ima dvije metode koje je važno objasniti. Metoda *validator* vraća polja i pravila koja ista ta polja moraju zadovoljavati te potom uspoređuje podatke koje smo primili s prethodnom definicijom kako uneseni podaci trebaju izgledati. Ako provjera zadovolji, program nastavlja dalje, a u protivnom, neće se nastaviti izvršavati. Potom, u metodi *create* kreirana je instanca novog korisnika nakon provedene registracije, a sama logika ovih dvaju metoda prikazana je na slici 3.4.

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

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

Sl. 3.4. Funkcije *validator* i *create* unutar *RegisterController*

Unutar *LoginController* dodana je najobičnija provjera uvjeta da li je korisnik prijavljen u sustav, te ako je, da li je on tipa *admin* ili *user*, a logika tog kontrolora nalazi se na slici 3.5.

```
    use AuthenticatesUsers;

    /**
     * Create a new controller instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware('guest')->except('logout');
    }

    protected function redirectTo()
    {
        if (auth()->user() && auth()->user()->isAdmin()) {
            return route('admin.vacations');
        }

        return route('vacations');
    }
}
```

Sl. 3.5. Logika kod *LoginController*

3.2.2. Kontrolori

Kontrolori su klase koje mogu grupirati relacijsku logiku zahtjeva unutar jedne klase, a korisne su jer je alternativa pisati cijeli ovaj kod unutar *route* direktorija. Svaki kontrolor nasljeđuje glavnu kontrolor klasu koja dolazi kreirana od strane *Laravela*, a ta, glavna klasa osigurava aplikaciji neke važne metode koje se kasnije koriste na nekoliko mjesta. Za potrebe ove aplikacije kreirano je tri glavna kontrolora, uz one predefinirane koji dolaze s *Laravelom*, a to su *VacationController*, *ProfileController* te *HomeController*. Uzmimo za primjer *ProfileController* koji ima dvije glavne metode, a to su *show* i *update*. Metoda *show* zapravo vraća pogled unutar kojega smo prethodno definirali kako će naša *HTML* forma izgledati, a pomoću *get* rute, unutar *web.php* datoteke predajemo kontroloru koje podatke on treba vratiti. *Update* metoda samo izvršava ažuriranje podataka koje je korisnik prethodno unio u bazu. Obje metode prikazane su na slici 3.6.

```php
class ProfileController extends Controller
{
    public function show()
    {
        return view('profile');
    }

    public function update(UpdateProfileRequest $request)
    {
        $user = $request->user();

        $user->first_name = $request->input('first_name');
        $user->last_name = $request->input('last_name');
        $user->oib = $request->input('oib');

        $location = $user->location;
        if (!$location) {
            $location = new Location;
        }

        $location->address = $request->input('location.address');
        $location->zip = $request->input('location.zip');
        $location->city = $request->input('location.city');
        $location->country = $request->input('location.country');
        $location->save();

        $user->location()->associate($location);

        $user->save();

        return redirect()->route('profile');
    }
}
```

Sl. 3.6. Prikaz *show* i *update* metode

Osim spomenutih kontrolora, unutar admin mape postoje i *VacationController* te *CompanyController*. Prvi sadrži logiku za kreiranje i spremanje PDF dokumenata, kao i neke pomoćne metode, dok *CompanyController* rukuje podacima o kompaniji.

3.2.3. *Middleware*

Middleware osigurava prikladan mehanizam filtriranja *HTTP* zahtjeva koji ulaze u aplikaciju. Laički rečeno, to su sve klase koje se izvršavaju prije no što zahtjev dođe do neke od metoda. *Laravel* ih obično sam generira za svaki projekt, a primjer takve klase je recimo ona za verifikaciju korisnika. Ukoliko korisnik nije prijavljen, *middleware* će ga automatski proslijediti na zaslon s prijavom. S druge pak strane, ukoliko je prijavljen, *middleware* će dozvoliti nastavak zahtjeva te će otvoriti zaslon koji slijedi, obično neki početni zaslon. Dodatno, *middleware* klase mogu biti i ručno napisane unutar aplikacije, a za potrebe ovoga rada napisane su dvije, *UserMiddleware* te *AdminMiddleware*, koje su potrebne kako bi se ovisno o tipu korisnika, ukoliko se radi o *adminu*, omogućile neke dodatne opcije. Na slici 3.7. prikazan je kod za

AdminMiddleware klasu, koja zapravo funkcionira na način da u svojoj metodi provjerava je li korisnik prijavljen, i ako je, da li je tipa *admin*.

```php
class AdminMiddleware
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        if (!$request->user() || !$request->user()->isAdmin()) {
            return redirect('/');
        }

        return $next($request);
    }
}
```

Sl. 3.7. *AdminMiddleware* klasa

3.2.4. Rute

Laravel rute obično se definiraju u *routes* direktoriju unutar projekta, a ove datoteke također su generirane automatski od strane programskog okvira. Datoteka *routes/web.php* sadrži rute zadužene za sučelje naše web aplikacije. Ove rute pridružene su grupi *middleware*, koja omogućava funkcionalnosti poput kontroliranja sesije, ali i *CSRF* zaštite koja se odnosi na sprječavanje davanja neovlaštenih naredbi korisniku. Metode koje se mogu koristiti unutar rute najčešće su *get, post, put, patch, delete* i *options*, dok su za potrebe ovog rada upotrijebljene *get* i *post*, a *web.php* datoteka prikazana je na slici 3.8. Parametri koje ruta obično sadržava pišu se unutar {} zagrada, a moguće je definirati onoliko parametara koliko ruta zahtijeva.

```php
Route::middleware(['admin'])->group(function () {
    Route::get('vacations', 'Admin\VacationController@index')->name('admin.vacations');
    Route::get('vacations/{vacation}/pdf', 'Admin\VacationController@getRequestPdf')->name('admin.vacations.show');
    Route::get('vacations/{vacation}/approval/pdf', 'Admin\VacationController@getApprovalPdf')->name('admin.approval.show');

    Route::post('vacations/{vacation}/accept', 'Admin\VacationController@accept')->name('admin.vacations.accept');
    Route::post('vacations/{vacation}/reject', 'Admin\VacationController@reject')->name('admin.vacations.reject');

    Route::get('company', 'Admin\CompanyController@edit')->name('admin.company');
    Route::post('company', 'Admin\CompanyController@update')->name('admin.company.update');
});

Route::middleware(['user'])->group(function() {
    Route::get('profile', 'ProfileController@show')->name('profile');
    Route::post('profile', 'ProfileController@update');

    Route::get('my-vacations', 'VacationController@index')->name('vacations');
    Route::post('my-vacations', 'VacationController@store')->name('vacations.store');
    Route::get('my-vacations/create', 'VacationController@create')->name('vacations.create');
    Route::get('my-vacations/{vacation}', 'VacationController@show');

    Route::get('my-vacations/{vacation}/pdf', 'VacationController@getRequestPdf')->name('vacations.request.pdf');
    Route::get('my-vacations/{vacation}/approval/pdf', 'VacationController@getApprovalPdf')->name('vacations.approval.pdf');
});

Auth::routes();

Route::get('/', 'HomeController@index')->name('home');
```

Sl. 3.8. Rute unutar aplikacije

3.2.5. Generiranje dokumenata – zahtjev i odobrenje

Postoje dva dokumenta koja se generiraju, a to su zahtjev i odobrenje. Nakon što korisnik popuni formu sa zahtjevom te klikne na 'predaj zahtjev', izvršava se *VacationController*, u sklopu kojega je najvažnija *store* metoda. Pomoću *post* metode unutar *web.php* direktorija pozvana je *store* metoda unutar *VacationController*. Metoda zapravo sprema trenutno prijavljenog korisnika, datume i tip, te podatke o tvrtki koju već imamo u sustavu, a *admin* ju po potrebi može ažurirati. Model se sprema u bazu, a potom se izvršava logika prikazana na slici 3.9.

```
    // Create and store PDF.
    $filename = 'Zahtjev-' . time() . rand(10,99). '.pdf';
    $path = storage_path($filename);
    PDF::loadView('vacations.view', ['vacation' => $vacation])->save($path);

    $vacation->request_pdf = $path;
    $vacation->save();

    session()->flash('success', 'Zahtjev je uspjesno kreiran!');

    return redirect()->route('vacations');
}
```

Sl. 3.9. Logika spremanja PDF dokumenata

Storage_path funkcija je od *Laravela* koja vraća gdje će se točno datoteka spremiti, a nakon što je kreiran zahtjev u bazi podataka, metoda *loadView* učitava *vacations.view* pogled te popunjava potrebne podatke, a iz tog pogleda se zatim generira PDF dokument. Taj dokument sprema se na odgovarajuću lokaciju, a ta se lokacija potom dostavlja bazi podataka na prethodno poslani zahtjev. Unutar sesije, imamo poruku da je zahtjev uspješno kreiran te se korisnik na samom kraju preusmjerava na listu zahtjeva.

3.3. Front-end

3.3.1. *DatePickerInput*

Vue.js, prethodno spomenuti programski okvir, korišten je upravo za ovu funkcionalnost. Prilikom odabira datuma od kojega korisnik želi da njegov plaćeni, odnosno neplaćeni dopust traje, te datuma kada će njegov dopust završiti, korišten je ovaj programski okvir, odnosno njegova logika. Ova datoteka smještena je unutar *resources/components* direktorija, a osim nje, ondje se nalaze i *bootstrap.js* te *app.js* koje Laravel sam kreira. *DatePickerInput* komponenta je registrirana upravo unutar *app.js* datoteke, a napisana je u u svom zasebnom direktoriju. Ondje je definirano sve što je potrebno za odabir datuma, postavljeno je da će *datepicker* imati format datuma, da je polje obavezno, onemogućen je odabir prethodnih datuma te su datumi koji se nalaze u prošlosti također onemogućeni za odabir. Sama logika ove komponente prikazana je na slici 3.10.

```
<template>
    <div>
        <datepicker v-model="date"
                    :name="name"
                    :required="true"
                    :disabledDates="{
                        to: new Date(new Date().setDate(new Date().getDate()-1))
                    }"
                    :format="'yyyy-MM-dd'"
                    required
        ></datepicker>
<!--        <input type="hidden" name="{{name}}" :value="date">-->
    </div>
</template>
```

Sl. 3.10. Logika kod *DatePickerInput.vue*

3.3.2. Pogledi

Treća komponenta *MVC* arhitekture odnosi se na poglede. Pogledi obuhvaćaju *HTML* kod koji aplikacija sadrži te odvaja logiku kontrolora od prezentacijske logike. Pohranjuju se unutar *resources/views* direktorija. Uz poglede se često koristi i *Blade*, jednostavan, ali vrlo moćan *template* alat koji se, za razliku od kontrolora, zasniva na nasljeđivanju predložaka i sekcijama. Svi *Blade* predlošci obično koriste *.blade.php* ekstenziju u imenu. Osnovni i glavni pogled ove web aplikacije datoteka je imena *app.blade.php*, smještena unutar *resources/views/layouts* mape. Ovaj *layout* služi kao temelj aplikacije, korišten je radi bolje preglednosti koda, a glavna funkcionalnost je navigacijska traka koja je implementirana pomoću *Bootstrapa*. To je traka koja se nalazi na vrhu stranice te služi boljoj preglednosti, ali i navigaciji ako postoji više zaslona. U ovoj web aplikaciji ona sadrži ime aplikacije u jednom uglu, dok se u drugom nalazi prikaz imena i prezimena trenutno prijavljenog korisnika, a klikom na strjelicu pored, otvara se padajući izbornik gdje se korisnik može odjaviti ili otići na prikaz svog profila, te ga po potrebi ažurirati. Dio koda koji se izvršava za uspješan prikaz padajućeg izbornika prikazan je na slici 3.11.

```
<li class="nav-item dropdown">
    <a id="navbarDropdown" class="nav-link dropdown-toggle" href="#" role="button" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false" v-pre>
        {{ Auth::user()->name }} <span class="caret"></span>
    </a>

    <div class="dropdown-menu dropdown-menu-right" aria-labelledby="navbarDropdown">
        @if(auth()->user()->isUser())
            <a class="dropdown-item" href="{{ route('profile') }}">
                {{ __('Profil') }}
            </a>
        @elseif(auth()->user()->isAdmin())
            <a class="dropdown-item" href="{{ route('admin.company') }}">
                {{ __('Tvrtka') }}
            </a>
        @endif
        <a class="dropdown-item" href="{{ route('logout') }}"
           onclick="event.preventDefault();
                         document.getElementById('logout-form').submit();">
            {{ __('Odjava') }}
        </a>

        <form id="logout-form" action="{{ route('logout') }}" method="POST" style="display: none;">
            @csrf
        </form>
    </div>
</li>
```

Sl. 3.11. Implementacija padajućeg izbornika kod navigacijske trake

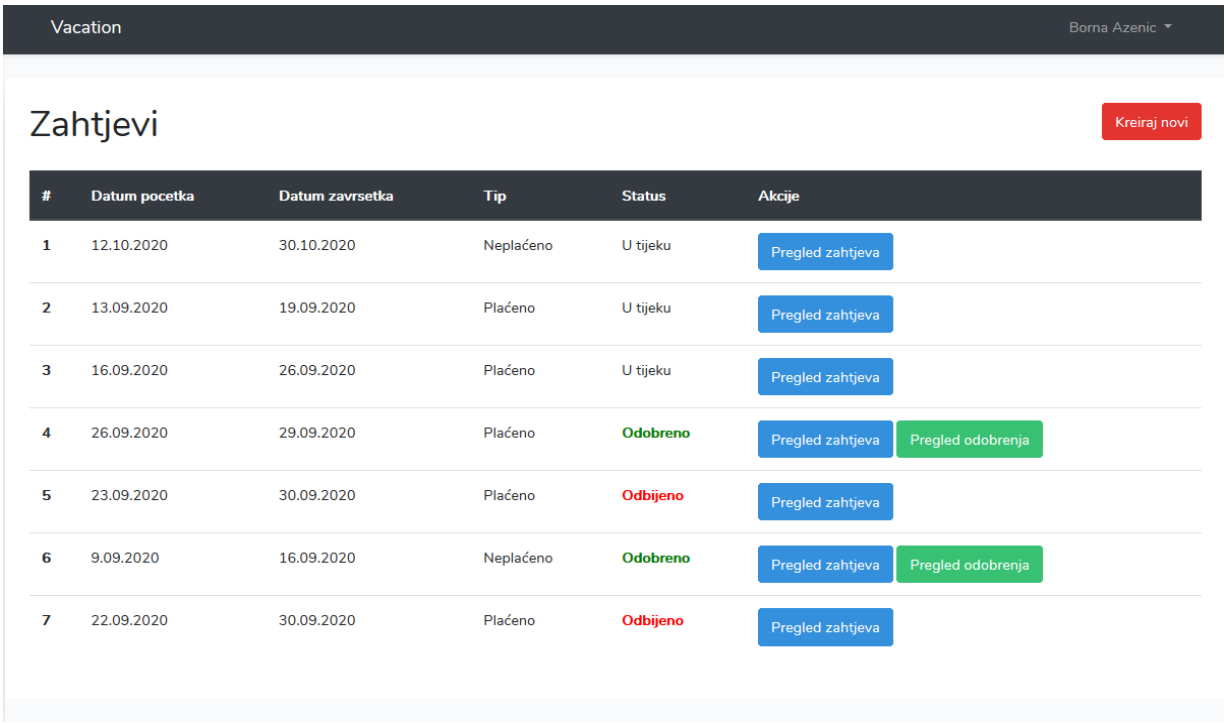
Sljedeći pogled koji će biti spomenut je *profile.blade.php*, koji sadrži *HTML* kod za prikaz zaslona korisničkog profila. Pogled sadrži isključivo tekstualna polja koja su pomoću *div* separatora podijeljena u nekoliko dijelova, a primjer jednog takvog dijela, sa svim pripadajućim oznakama možemo vidjeti na slici 3.12. Važno je napomenuti kako svaki od zaslona web aplikacije sadrži svoj pripadajući pogled, a oni su napisani istom logikom.

```
<h1 class="pb-3 pt-2">Lokacija</h1>
<div class="form-group row">
    <label for="address" class="col-sm-2 col-form-label">Adresa</label>
    <div class="col-sm-10">
        <input type="text" class="form-control" id="address" name="location[address]" required value="{{ auth()->user()->location ? auth()->user()->location->address : null }}">
    </div>
</div>
<div class="form-group row">
    <label for="zip" class="col-sm-2 col-form-label">ZIP</label>
    <div class="col-sm-10">
        <input type="text" class="form-control" id="zip" name="location[zip]" required value="{{ auth()->user()->location ? auth()->user()->location->zip : null }}">
    </div>
</div>
<div class="form-group row">
    <label for="city" class="col-sm-2 col-form-label">Mjesto</label>
    <div class="col-sm-10">
        <input type="text" class="form-control" id="city" name="location[city]" required value="{{ auth()->user()->location ? auth()->user()->location->city : null }}">
    </div>
</div>
<div class="form-group row">
    <label for="country" class="col-sm-2 col-form-label">Zemlja</label>
    <div class="col-sm-10">
        <input type="text" class="form-control" id="country" name="location[country]" required value="{{ auth()->user()->location ? auth()->user()->location->country : null }}">
    </div>
</div>

<div class="float-right pb-4 pt-4">
    <button class="btn btn-danger" style="margin-left: auto;">Spremi</button>
</div>
```

Sl. 3.12. Isječak koda iz pogleda *profile.blade.php*

List.blade.php pogled je koji se odnosi na zaslon gdje se prijavljenom korisniku prikazuju njegovi prethodno podneseni zahtjevi, a za koje on može vidjeti datum početka, odnosno završetka, tip dopusta koji želi, status istog tog dopusta te je pristuna kolona akcije, gdje on može vidjeti svoj zahtjev, a ako je zahtjev odobren, može vidjeti i odobrenje. Prikaz tog zaslona nalazi se na slici 3.13., a dio implementacije koda je na slici 3.14.



Sl. 3.13. Zaslon s prikazanim zahtjevima

```
    <table class="table">
        <thead class="thead-dark">
        <tr>
            <th scope="col">#</th>
            <th scope="col">Datum pocetka</th>
            <th scope="col">Datum zavrsetka</th>
            <th scope="col">Tip</th>
            <th scope="col">Status</th>
            <th scope="col">Akcije</th>
        </tr>
        </thead>
        <tbody>
        @if(!count($vacations))
            <tr><td colspan="5" class="text-center font-italic">Nema zapisa</td></tr>
        @endif

        @foreach($vacations as $k => $vacation)
        <tr>
            <th scope="row">{{$k + 1}}</th>
            <td>{{ $vacation->start_date->format('j.m.Y') }}</td>
            <td>{{ $vacation->end_date->format('j.m.Y') }}</td>
            <td>{{ $vacation->getTypeFormatted() }}</td>
            <td>
                @if($vacation->isApproved())
                    <span style="color: green; font-weight: bold;">{{ $vacation->getStatusFormatted() }}</span>
                @elseif($vacation->isRejected())
                    <span style="color: red; font-weight: bold;">{{ $vacation->getStatusFormatted() }}</span>
                @else
                    {{ $vacation->getStatusFormatted() }}
                @endif
            </td>
            <td>
                @if($vacation->isPending() || $vacation->isRejected())
                    <a target="_blank" href="{{ route('vacations.request.pdf', $vacation->id) }}" class="btn btn-primary">Pregled zahtjeva</a>
                @elseif($vacation->isApproved())
                    <a target="_blank" href="{{ route('vacations.request.pdf', $vacation->id) }}" class="btn btn-primary">Pregled zahtjeva</a>
                    <a target="_blank" href="{{ route('vacations.approval.pdf', $vacation->id) }}" class="btn btn-success">Pregled odobrenja</a>
                @endif
            </td>
        </tr>
        @endforeach
        </tbody>
    </table>
</div>
```

Sl. 3.14. Implementacija koda za zaslon sa zahtjevima

3.3.3. Dokumenti – zahtjev i odobrenje

Glavni cilj ovog diplomskog rada bio je zapravo generirati dokumente za plaćeni ili neplaćeni dopust, a to su nekakav zahtjev koji zaposlenik podnosi te nekakvo odobrenje koje nadređeni kasnije potpisuje, odnosno izdaje. Sama logika generiranja ovih zahtjeva objašnjena je na stranicama ispred, a ovdje je bitno objasniti kako ovi dokumenti izgledaju te koje točno elemente sadržavaju. Predložak za ove dokumente nalazi se unutar *views/vacations* direktorija, a ondje su kreirani *view.blade.php*, odnosno *approval.blade.php* predlošci. Oba ova predloška sastoje se od glavnih *HTML* elemenata te najosnovnije *CSS* logike. Primjer uspješno generiranog zahtjeva za plaćenim, odnosno neplaćenim dopustom prikazan je na slici 3.15., dok je generirano odobrenje prikazano na slici 3.16.


FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Zahtjev #: 17
Datum zahtjeva: 12.09.2020.

Borna Azenic
Ulica 1, Valpovo, 31550, Hrvatska
borna@mail.com
OIB: 7778889912
09811112222

FERIT
Kneza Trpimira 2B
31000 Osijek
Hrvatska

Zahtjev za korištenje plaćenog dopusta

Poštovani,

molim da mi se odobri korištenje plaćenog dopusta u trajanju od 26.09.2020. do 29.09.2020.

U Osijeku, 12.09.2020

Potpis voditelja

Sl. 3.15. Zahtjev u obliku PDF dokumenta

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Na temelju članka 57. Pravilnika o radu i sukladno članku 25. Statuta Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek i u skladu s Odlukom o korištenju godišnjeg odmora, a prema zahtjevu Borna Azenic, dekan Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek, prof. dr. sc. Drago Žagar, donosi sljedeću

ODLUKU
o pravu korištenja dijela godišnjeg odmora

I.

Borna Azenic, zaposlen na radnom mjestu I.vrste - Student na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek, odobrava se korištenje godišnjeg odmora u trajanju 3 dana.

II.

Radnik će dio godišnjeg odmora koristiti u vremenskom periodu od 26.09.2020 do 29.09.2020.

III.

Za vrijeme korištenja godišnjeg odmora radniku se isplaćuje plaća u visini kao da je radio u redovnom radnom vremenu

Dostavljeno:
1. Borna Azenic
2. Ured za računovodstveno-financijske poslove
3. Personalni dosje
4. Arhiva

_____
Potpis dekana

Sl. 3.16. Odobrenje u obliku PDF dokumenta

4. TESTIRANJE APLIKACIJE

U ovom poglavlju bit će detaljnije opisano testiranje aplikacije. Bit će prikazano izvršavanje nekoliko testnih slučajeva pomoću kojih će biti prikazani svi bitniji dijelovi ove aplikacije te zapravo način na koji aplikacija kao cjelina funkcionira. Osim toga, drugi dio testiranja izveden je programski, odnosno unutar aplikacije dodani su testovi koji će u sklopu *PHPUnit* programskog okruženja biti provedeni te će i na taj način biti ispitana uspješnost aplikacije.

4.1. Postavke testiranja

4.1.1. Definicija testnih slučajeva

U ovom potpoglavlju bit će definirana četiri testna slučaja čije će, nadamo se, uspješno provođenje pokriti i testirati ispravnost web aplikacije. Testni slučajevi su:

- Registracija, prijava i odjava korisnika – provjera autentifikacije korisnika.
- Kreiranje novog zahtjeva – provjera ispravnosti funkcionalnosti dodavanja novog zahtjeva.
- Ažuriranje tvrtke te osobnih podataka – provjera osnovnih razlika između admina i običnog korisnika.
- Prihvaćanje ili odbijanje pristiglog zahtjeva – provjera generiranja odobrenja.

4.1.2. Programska definicija testova

U ovom potpoglavlju biti će opisani testovi koji su definirani unutar same aplikacije. Oni su programski napisani, a za potrebe testiranja korišteni su *Factories* alati, koje osigurava *Laravel*. To su metode koje nam omogućavaju lakše kreiranje testnih korisnika u bazi podataka, nalaze se unutar *database* direktorija, a unutar svake te datoteke najvažnija je *faker* biblioteka koja zapravo radi posao generiranja nasumičnih podataka potrebnih za testiranje. Za potrebe ove aplikacije napisane su 3 takve datoteke, *UserFactory*, *CompanyFactory* te *LocationFactory*, a dio koda za *UserFactory* prikazan je na slici 4.1. Unutar *phpunit.xml* navedeno je koja baza podataka se koristi te koja konekcija na bazu, a inače se koristi *SQLite* radi manjeg zagađenja nad podacima. Prilikom implementacije određene funkcionalnosti, analogno se radi test koji će potom provjeriti radi li ta funkcionalnost ispravno. Ovo je jako korisno jer ako se naknadno rade izmjene i ažuriranja, prisutna je sigurnost da dijelovi koji su prethodno napisani rade. Ili, u slučaju da ne rade, odmah je jasno što ne funkcionira i zašto.

```php
$factory->define(\App\Models\User::class, function (Faker $faker) {
    return [
        'first_name' => $faker->firstName,
        'last_name' => $faker->lastName,
        'role' => 'user',
        'phone' => $faker->phoneNumber,
        'oib' => $faker->randomNumber(8),
        'email' => $faker->unique()->safeEmail,
        'password' => '$2y$10$92IXUNpkjO0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi', // password
        'remember_token' => Str::random(10),
        'location_id' => factory(\App\Models\Location::class)->create(),
    ];
});
```

Sl. 4.1. Logika koda za *UserFactory*

Aplikacija ima tri testa, od kojih je svaki deskriptivno napisan. *LoginTest* prvi je od testova, a samo ime govori da se odnosi na formu za prijavu. Test sadrži nekoliko metoda od kojih svaka provjerava neku od funkcionalnost forme za prijavu, a primjer takve metode je *testCanViewLoginForm()*, koja provjerava otvara li se forma za prijavu. Pomoću *assertStatus()* metode provjerava se status stranice, ako vrati 200 znači da je sa sesijom sve u redu, a na samom kraju testa vrši se provjera da li je vraćen pogled na *auth.login*. Implementacija koda za ovu metodu nalazi se na slici 4.2.

```php
public function testCanViewLoginForm()
{
    $response = $this->get('/login');

    $response->assertStatus(200);
    $response->assertViewIs('auth.login');
}
```

Sl. 4.2. *testCanViewLoginForm()* metoda

Drugi test naziva je *ProfileTest*, a koristi se ista metodologija za provjeru nekih od funkcionalnosti aplikacije. Sadrži metode koje provjeravaju može li se vidjeti pogled, vidimo li na formi tekst koji bi se trebao prikazivati te mogu li se ažurirati podaci. Potom, pomoću već spomenute *faker* biblioteke, kreiran je testni korisnik koji je spremljen u bazu podataka. On se prijavio na stranicu te se izvršava provjera može li ažurirati svoje podatke. Treći, a ujedno i posljednji test,

VacationTest, zadužen je za provjeru ispravnosti unesenih datuma kod kreiranja zahtjeva za dopustom, a ta je metoda prikazana na slici 4.3.

```
public function testCannotCreateVacationWhenDateIsInvalid()
{
    $company = factory(Company::class)->create();
    $location = factory(Location::class)->create();

    $user = factory(User::class)->create([
        'password' => bcrypt($password = 'mypassword'),
        'location_id' => $location->id,
    ]);

    $response = $this->actingAs($user)->get('/my-vacations/create');

    $response->assertStatus(200);
    $response->assertSee('Zahtjev za godišnjim');

    $response = $this->post('/my-vacations', [
        'start_date' => '2021-02-10',
        'end_date' => '2021-02-05',
        'type' => 'paid',
    ]);

    $response->assertSessionHas('invalid_date', true);

}
```

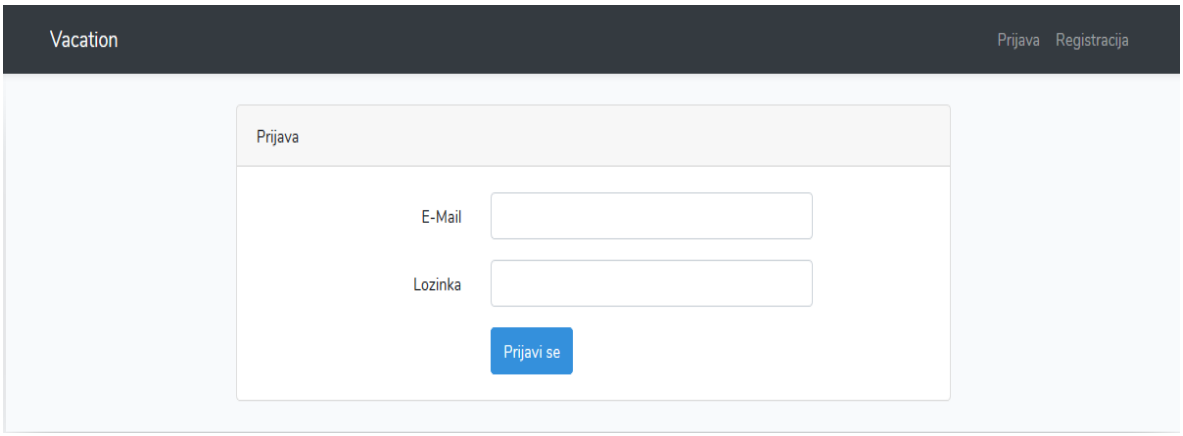
Sl. 4.3. Provjera ispravnosti unesenih datuma

4.2. Rezultati testiranja

4.2.1. Izvršavanje testnih slučajeva

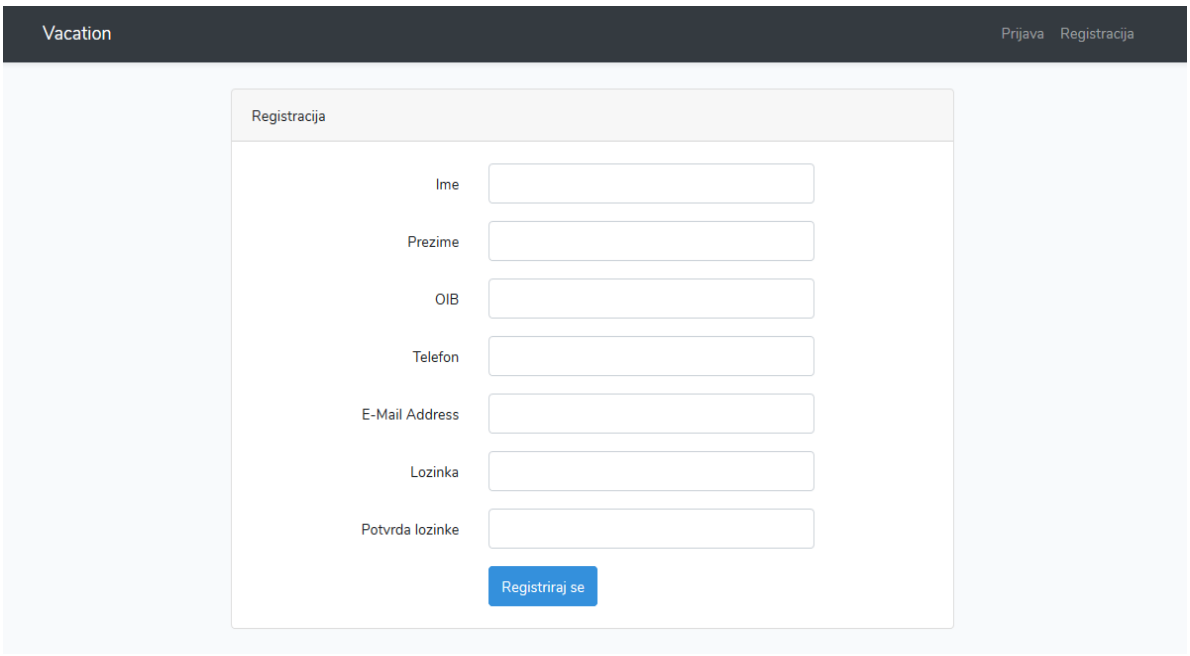
1. Registracija, prijava i odjava korisnika.

Nakon pokretanja aplikacije, korisniku se otvara forma na kojoj treba unijeti mail te lozinku. Izgled te forme moguće je vidjeti na slici 4.4.



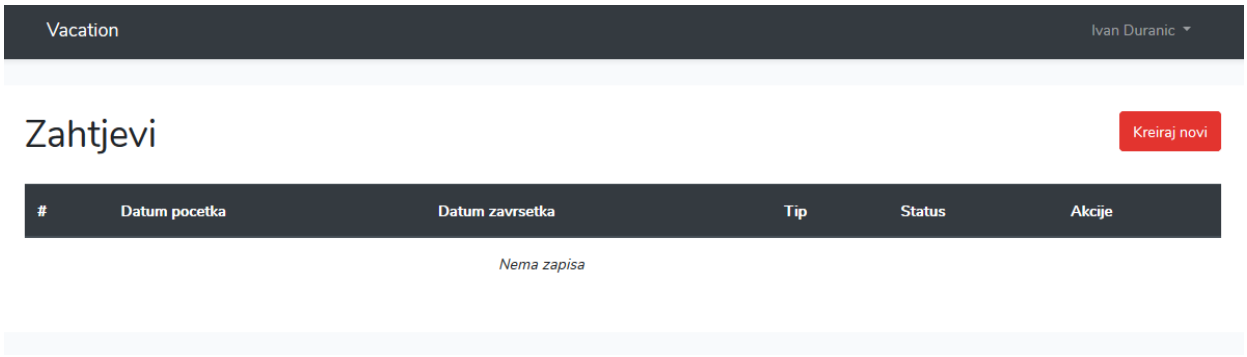
Sl. 4.4. Forma za prijavu

Nakon klika na 'Registracija' u gornjem desnom uglu, korisniku se otvara sljedeća forma prikazana na slici 4.5.



Sl. 4.5. Forma za registraciju

Nakon što ispuni sve potrebne podatke, klikom na 'Registriraj se' korisnički račun biti će uspješno kreiran te se korisniku otvara početni ekran, prikazan na slici 4.6.

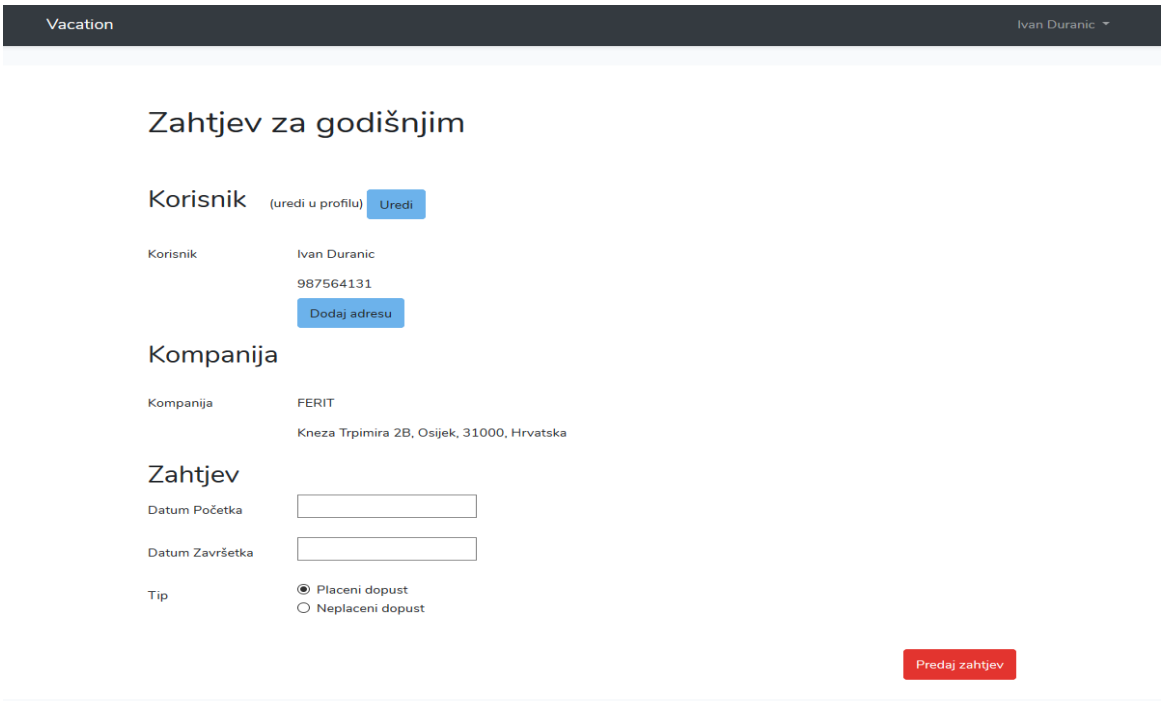


Sl. 4.6. Početni zaslon aplikacije

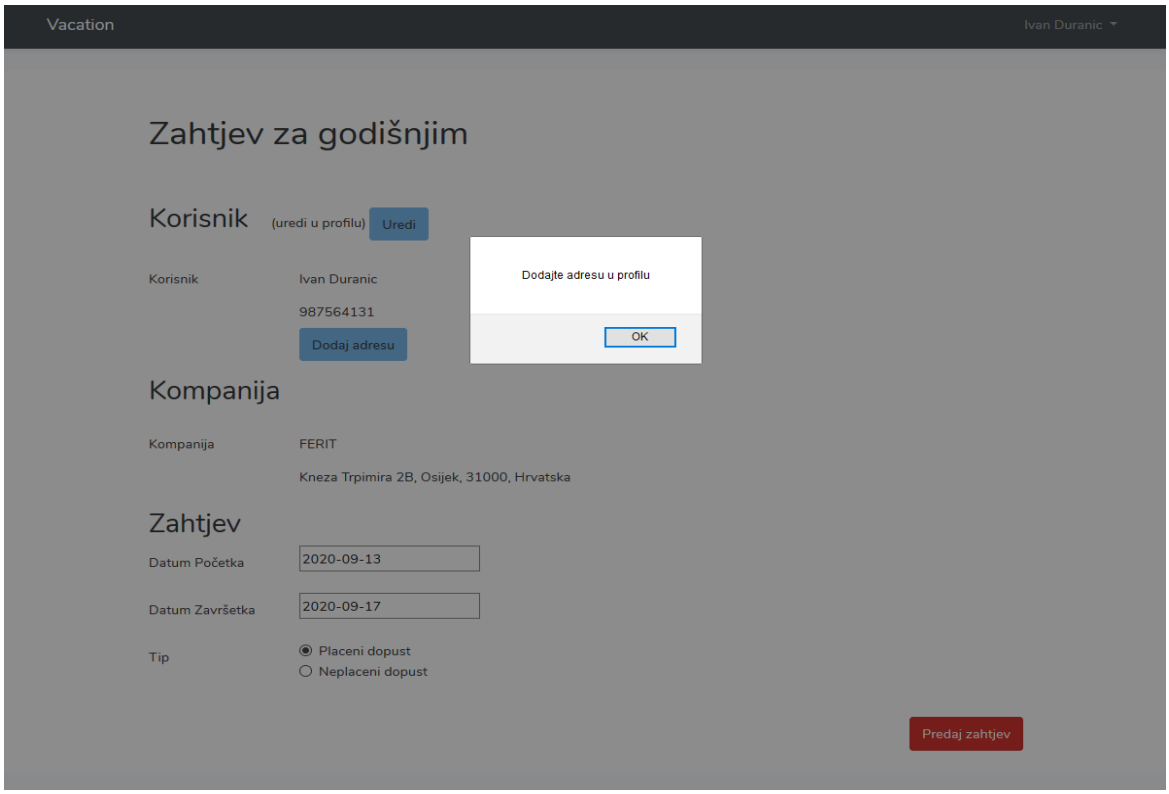
Klikom na 'Odjava' do koje se dolazi preko padajućeg izbornika kod imena i prezimena trenutno prijavljenog korisnika, on se uspješno odjavljuje iz aplikacije.

2. Kreiranje novog zahtjeva.

Klikom na 'Kreiraj novi' korisniku se otvara nova forma gdje on unosi sve potrebno generiranje zahtjeva. Izgled forme prikazan je na slici 4.7. Pritom, obavezno je da prije predaje zahtjeva ažurira podatke u profilu jer je potrebna i adresa, a izgled skočnog prozora koji se prikaže u slučaju da nije dodao adresu nalazi se na slici 4.8. Dodana su i ograničenja za datume, nije moguće odabrati datum iz prošlosti te nije moguće odabrati da datum završetka bude prije datuma početka.

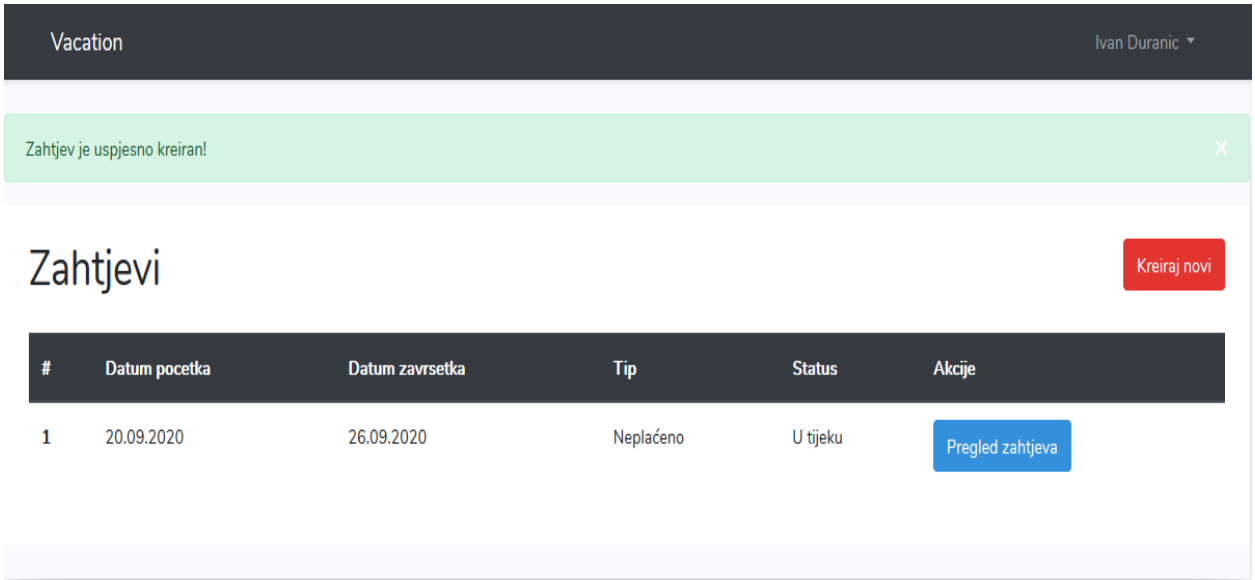


Sl. 4.7. Forma za generiranje zahtjeva



Sl. 4.8. Upozorenje kod nepostojeće adrese

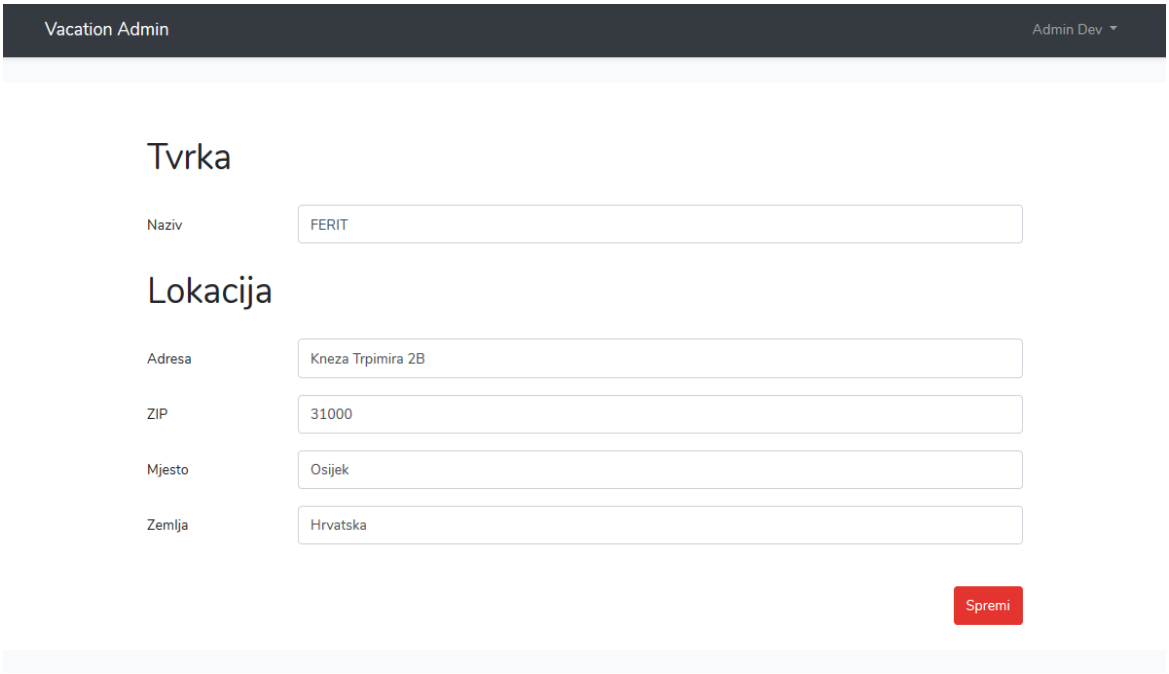
Nakon odabira datuma početka i završetka, te odabira tipa dopusta koji će mu odgovarati, korisnik klikom na tipku 'Predaj zahtjev' u donjem desnom uglu kreira zahtjev. Korisnik zatim na ekranu može vidjeti obavijest da je zahtjev uspješno kreiran te može pregledati zahtjev, a sve to vidi se iz slike 4.9.



Sl. 4.9. Uspješno kreiran zahtjev

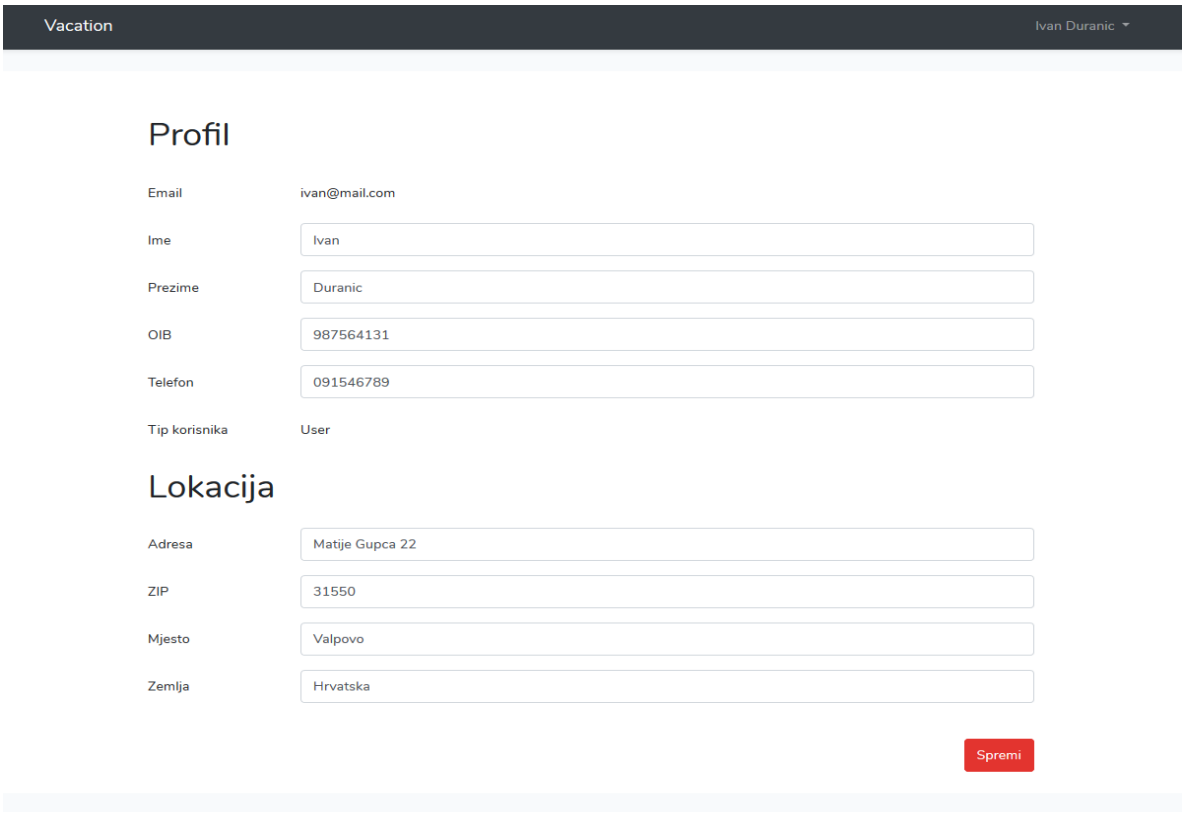
3. Ažuriranje tvrtke te osobnih podataka u profilu.

Korisnik koji ima *admin* ovlasti u svom profilu može, klikom na padajući izbornik u gornjem desnom uglu, te potom na tipku 'Tvrtka' ažurirati podatke o tvrtki te na taj način prilagoditi aplikaciju nekom drugom poslodavcu, a izgled te forme prikazan je na slici 4.10.



Sl. 4.10. Ažuriranje podataka o tvrtki

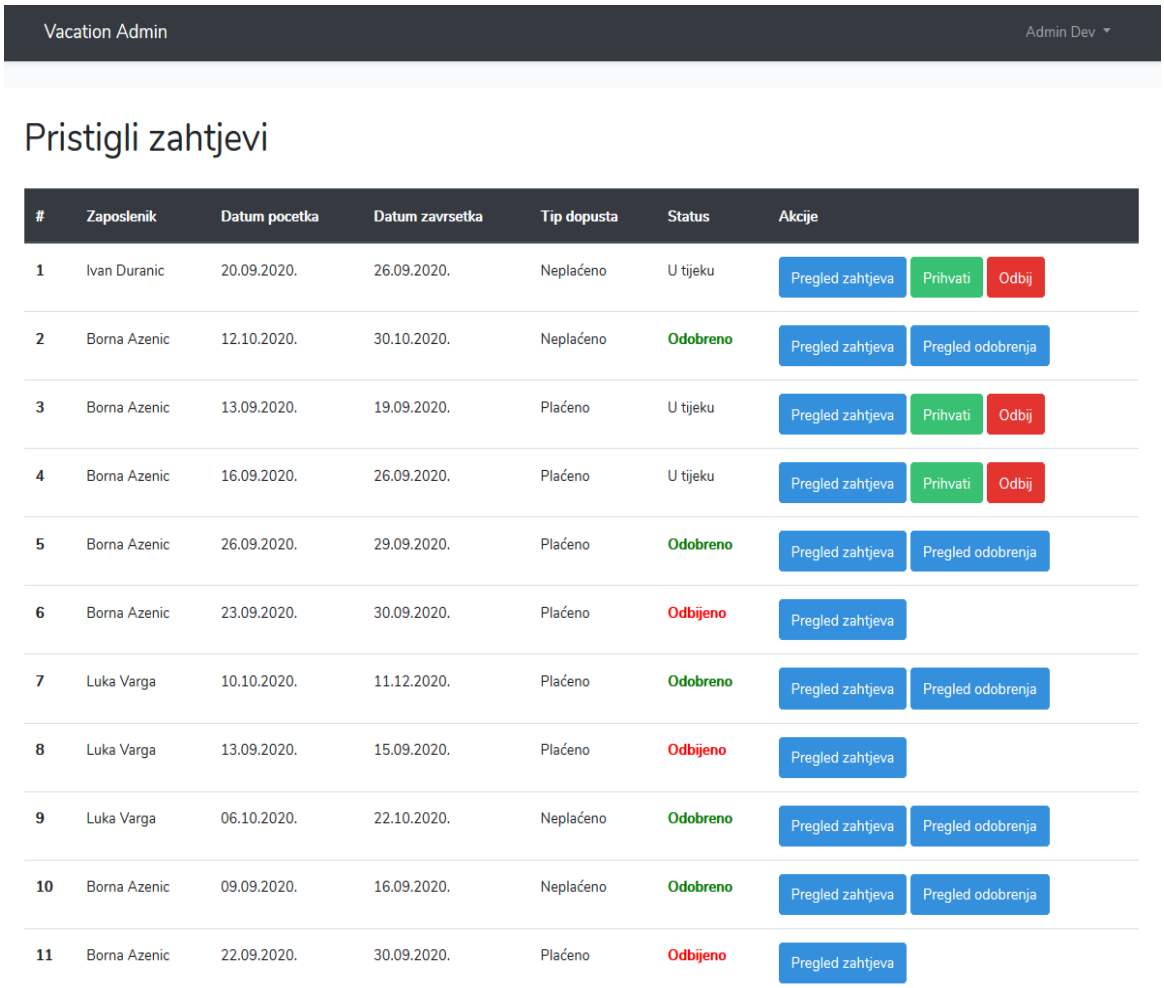
Osim ažuriranja podataka o tvrtki, regularan korisnik može u postavkama svojega profila ažurirati i svoje osobne podatke. Postupak je sličan onom gore opisanom, preko padajućeg izbornika u gornjem desnom uglu pristupa svojem profilu te ondje može lako izmijeniti podatke, a ta forma nalazi se na slici 4.11.



Sl. 4.11. Ažuriranje osobnih podataka u profilu

4. Prihvaćanje ili odbijanje pristiglog zahtjeva.

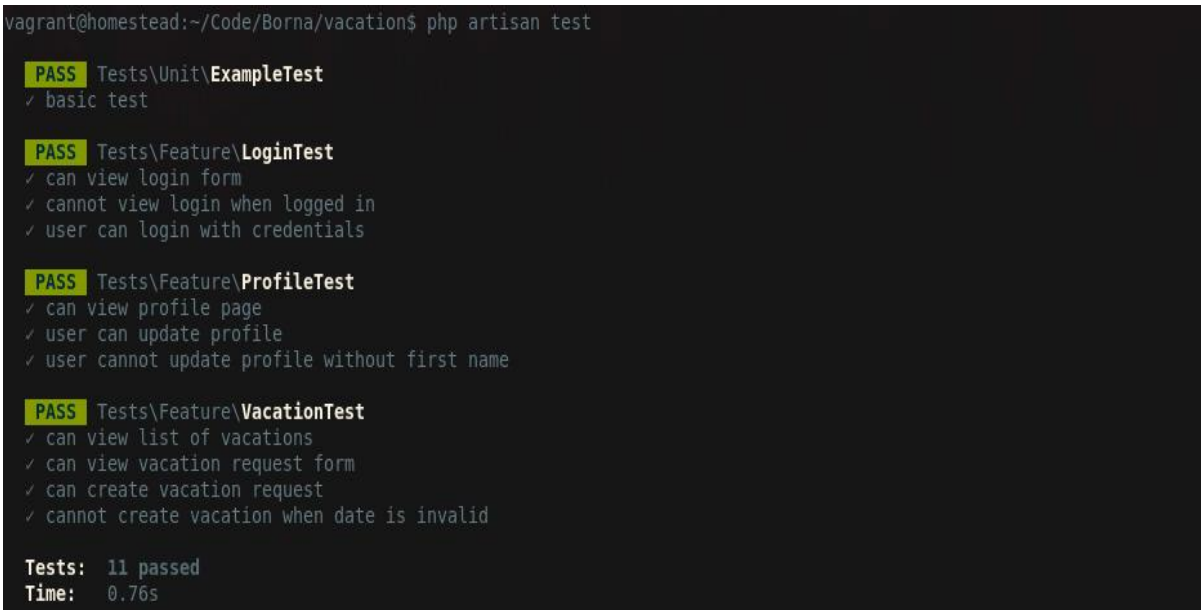
Korisnik s *admin* ovlastima u svojem profilu može, ovisno o situaciji u tvrtki, svaki pristigli zahtjev odobriti ili prihvatiti. Forma na kojoj se može vidjeti izgled svih pristiglih zahtjeva, kao i njihovih statusa (u tijeku, odobreno ili odbijeno) prikazana je na slici 4.12. Klikom na tipku 'Prihvati', zahtjev se prihvaća, a klikom na 'Odbij', zahtjev se odbija. Neovisno o ishodu, zahtjev se i dalje može pregledati.



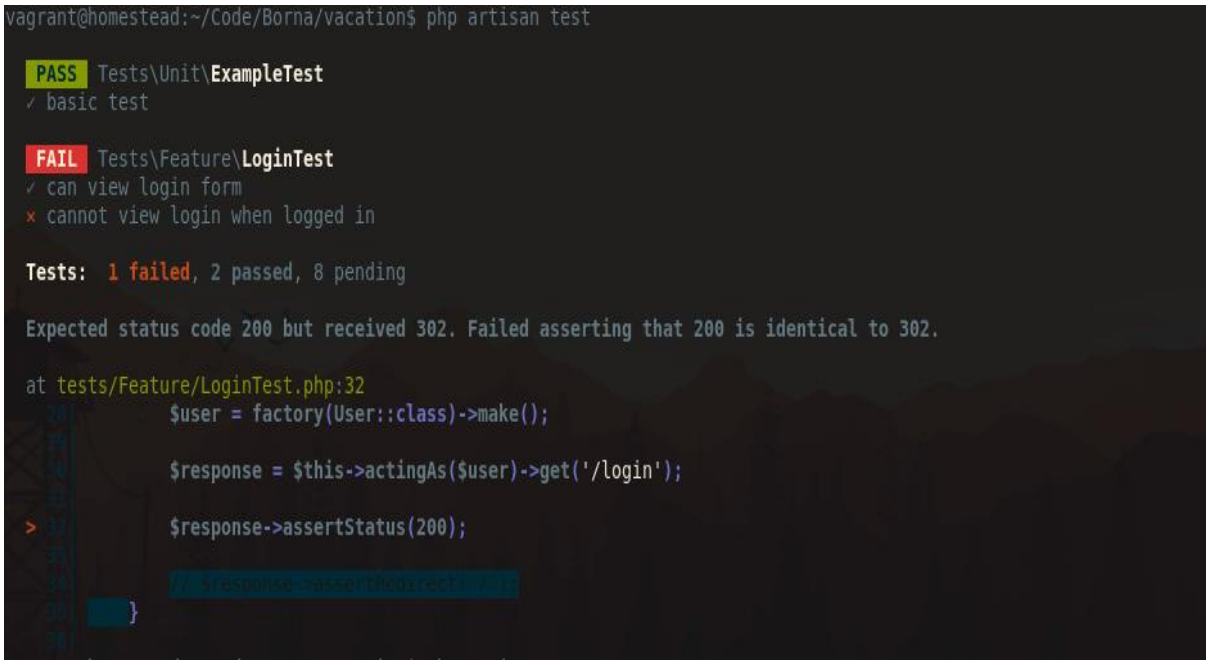
Sl. 4.12. Prikaz pristiglih zahtjeva

4.2.2. Uspješnost programskih testova

Rezultati programskih testova prikazani su na slici 4.13., a može se donijeti zaključak kako su svi testovi uspješno izvršeni. Primjera radi, prikazano je i kako izgleda kada ne prođu svi testovi, a taj slučaj nalazi se na slici 4.14.



Sl. 4.13. Uspješno provedeni programski testovi



Sl. 4.14. Neuspješno provedeni programski testovi

5. ZAKLJUČAK

U današnje suvremeno doba tehnologija zaista napreduje brzinom svjetlosti te se svakodnevno pojavljuju aplikacije koje olakšavaju posao, ali i život svim stanovnicima ove planete, a posebno onima koji imaju pristup internetskoj konekciji. Sukladno toj činjenici, ne treba iznenaditi potreba za izradom ove jednostavne, ali ipak zanimljive web aplikacije.

U sklopu ovog diplomskog rada osmišljena je i programski ostvarena web aplikacija koja asistira poslodavcima i zaposlenicima u odlasku na godišnji odmor. Sam proces sada je ubrzan i automatiziran, a ciljevi koji su na početku postavljeni uspješno su ostvareni u prikladnom programskom okruženju. Korisniku je omogućena registracija, a potom i prijava u sustav gdje on zatim ima mogućnost odabira perioda u kojem treba plaćeni ili neplaćeni dopust, a kreirani zahtjev tada može ponovno pregledati. S druge pak strane, postoji i korisnik koji ima *admin* ovlasti, a on može promijeniti ime tvrtke te na taj način prilagoditi aplikaciju različitim poslodavcima. Osim toga, može pregledati sve pristigle zahtjeve te ih naravno, odobriti ili odbiti.

Aplikacija je izrađena u *PHP* programskom jeziku, uz pomoć *Vue.js, Bootstrap* te *Laravel* programskih okvira. Za bazu podataka korišten je *MySQL*, ali su i u sklopu *Laravela* korištene migracije radi lakšeg manevriranja s kreiranim tablicama. Aplikacija je uspješno testirana, kako programski, tako i kroz testne slučajeve, a oba testiranja pokazala su da program funkcionira kako je i predviđeno te da su uspješno implementirane sve funkcionalnosti koje su bile zadane na početku. Znanje stečeno tijekom izrade ove web aplikacije vrlo je značajno te se ono lako može primijeniti na izradu nekih sličnih web rješenja te na neke zadatke koji imaju veze s automatizacijom poslovanja ili identičnih procesa koji su zamorni i dugi.

LITERATURA

[1] Lynn Beighley, Michael Morrison, „Head First PHP & MySQL“

[2] PHP fundamentals, https://www.pluralsight.com/courses/php-fundamentals, pristup ostvaren 10. svibnja 2020.

[3] Architecture Overview, https://www.php.net/manual/en/mongodb.overview.php, pristup ostvaren 20. svibnja 2020.

[4] Server-side scripting, https://www.computerhope.com/jargon/s/server-side-scripting.htm, pristup ostvaren 15. svibnja 2020.

[5] TechDifferences, https://techdifferences.com/difference-between-server-side-scripting-and-client-side-scripting.html, pristup ostvaren 15. svibnja 2020.

[6] Fundamentals of MySQL, https://www.udemy.com/course/fundamentals-of-mysql/#instructor-1, pristup ostvaren 21. svibnja 2020.

[7] What is a Relational Database Management System, Codeacademy articles, https://www.codecademy.com/articles/what-is-rdbms-sql, pristup ostvaren 25. svibnja 2020.

[8] Getting Started with Laravel (PHP Framework) – The Bacics, https://www.pluralsight.com/courses/laravel-php-framework-getting-started-the-basics, pristup ostvaren 01. lipnja 2020.

[9] https://cupsofcocoa.files.wordpress.com/2011/08/mvc-diagram1.png, pristup ostvaren 01. lipnja 2020.

[10] Vue.js Fundamentals, https://www.pluralsight.com/courses/vuejs-fundamentals, pristup ostvaren 10. lipnja 2020.

[11] Bootstrap: Getting Started, https://www.pluralsight.com/courses/bootstrap-getting-started, pristup ostvaren 25. kolovoza 2020.

[12] Learn Bootstrap, https://www.codecademy.com/learn/learn-bootstrap, pristup ostvaren 27. kolovoza 2020.

[13] 10 Key Features to Look Up in a Bootstrap Template, Creative Tim, https://www.creative-tim.com/blog/web-design/key-features-look-bootstrap-template/, pristup ostvaren 27. kolovoza 2020.

SAŽETAK

Naslov: Web aplikacija za vođenje evidencije godišnjih odmora zaposlenika

Cilj ovog diplomskog rada izraditi je web aplikaciju koja će olakšati vođenje evidencije odlazaka zaposlenika na godišnji odmor. U okviru završnog rada izrađena je slična aplikacija, međutim ova aplikacija može se pokrenuti preko weba te je u nekim pogledima unaprijeđena. Aplikacija uvelike olakšava tvrtkama odlazak njihovih radnika na plaćene dopuste, godišnje odmore i sl. Na početku ovog diplomskog rada kratko su opisana prava radnika propisana zakonima u Republici Hrvatskoj, ali i prava koja svaki poslodavac prema istim tim zakonima posjeduje. Potom su opisane tehnologije koje su potrebne za izradu ove web aplikacije. Korišten je *PHP* programski jezik, u kombinaciji s *Laravel* te *Vue.js* programskim okvirima. Baza podataka detaljno je osmišljena pomoću *MySQL-a*. Ovim putem sam proces je automatiziran te je u potpunosti smanjena zamorna dokumentacija, a ako znamo da je automatizacija budućnost IT-a, tada ova aplikacija još više dobiva na svojoj vrijednosti.

Ključne riječi: godišnji odmor, evidencija, *PHP*, *Laravel*, *Vue.js*, baza podataka

ABSTRACT

Title: Web application for tracking the evidence of requests for vacation

The goal of this project was to develop a web application which will make easier to track the record of employees going on vacation. As a part of bachelor thesis, a similar application was developed, however, this application can be started through the web server and some of the features have been improved. The application greatly facilitates the procedure every company has with its own employees when going on a vacation. At the beginning of this work, a short introduction on working rights has been made, but also there was a comment about rights every employer has. After that, all technologies used for developing this project have been shortly described. PHP programming language has been used, in combination with Laravel framework and Vue.js framework. Database is designed using MySQL programming tool. This way, the process of going on vacation was automated and accelerated, and on the other hand, tiresome documentation has been reduced. If we know that automatization is the future of the IT, then this application's value is rapidly increasing.

Key words: vacation, records, PHP, Laravel, Vue.js, database

ŽIVOTOPIS

Borna Azenić rođen je 25. listopada 1995. godine u Osijeku. Nakon osnovne škole, u istom gradu upisuje i s odličnim uspjehom završava općU gimnaziju. Iako je tijekom svojeg srednjoškolskog obrazovanja težio kemiji i biologiji, odlučuje upisati studij računarstva na Fakultetu Elektrotehnike, Računarstva i Informacijskih Tehnologija u Osijeku. 2017. godine postaje sveučilišni prvostupnik inženjer računarstva. Iste godine, nastavlja svoje obrazovanje upisom na diplomski studij, blok programsko inženjerstvo. Svoja znanja nadograđuje samostalnim usavršavanjem, kao i obavljanjem prakse u Marrow Labs tvrtki na posljednjoj godini studija.

PRILOZI

- Word dokument rada
- PDF dokument rada
- Programski kod aplikacije