UNIVERZITET U BEOGRADU
EKONOMSKI FAKULTET

MASTER TEZA

PRIMENA LARAVEL FREJMVORKA U IZRADI VEB APLIKACIJE ZA POSLOVNE SISTEME

Kandidat:
Danilo Čobeljić 2672/21

Mentor:
dr Aleksandra Zečević

Beograd, jun 2024. godine

Izjava o akademskoj čestitosti

Student/kinja: Danilo Čobeljić

Broj indeksa: 2672/21

Autor/ka master rada pod nazivom:

Primena Laravel frejmvorka u izradi veb aplikacije za poslovne sisteme

Potpisivanjem izjavljujem:

- da je rad isključivo rezultat mog sopstvenog istraživačkog rada;
- da sam rad i mišljenja drugih autora koje sam koristio/la u ovom radu naznačio/la ili citirao/la u skladu sa Uputstvom;
- da su svi radovi i mišljenja drugih autora navedeni u spisku literature/referenci koji su sastavni deo ovog rada i pisani u skladu sa Uputstvom; o da sam dobio/la sve dozvole za korišćenje autorskog dela koji se u potpunosti/celosti unose u predati rad i da sam to jasno naveo/la;
- da sam svestan/na da je plagijat korišćenje tuđih radova u bilo kom obliku (kao citata, parafraza, slika, tabela, dijagrama, dizajna, planova, fotografija, filma, muzike, formula, veb sajtova, kompjuterskih programa i sl.) bez navođenja autora ili predstavljanje tuđih autorskih dela kao mojih, kažnjivo po zakonu (Zakon o autorskom i srodnim pravima, Službeni glasnik Republike Srbije, br. 104/2009, 99/2011, 119/2012), kao i drugih zakona i odgovarajućih akata Univerziteta u Beogradu;
- da sam da sam svestan/na da plagijat uključuje i predstavljanje, upotrebu I distribuiranje rada predavača ili drugih studenata kao sopstvenih;
- da sam svestan/na posledica koje kod dokazanog plagijata mogu prouzrokovati na predati master rad i moj status;
- da je elektronska verzija master rada identična štampanom primerku i pristajem na njegovo objavljivanje pod uslovima propisanim aktima Univerziteta.

Beograd, 21.06.2024 Potpis

Izjava o korišćenju

Dozvoljavam da se objave moji lični podaci vezani za dobijanje akademskog naziva master ekonomiste, kao što su ime i prezime, godina i mesto rođenja i datum odbrane rada.

Ovi lični podaci mogu se objaviti na mrežnim stranicama digitalne biblioteke, u elektronskom katalogu i u publikacijama Univerziteta u Beogradu – Ekonomskog fakulteta.

Ovlašćujem biblioteku Univerziteta u Beogradu – Ekonomskog fakulteta da u svoj digitalni repozitorijum unese moj završni (master) rad pod naslovom:

Primena Laravel frejmvorka u izradi veb aplikacije za poslovne sisteme

koji je moje autorsko delo.

Završni (master) rad sa svim prilozima predao/la sam u elektronskom formatu pogodnom za trajno arhiviranje.

Moj završni (master) rad, pohranjen u Digitalnom repozitorijumu Univerziteta u Beogradu – Ekonomskog fakulteta i dostupan u otvorenom pristupu, mogu da koriste svi koji poštuju odredbe sadržane u CC BY licenci Kreativne zajednice (*Creative Commons*), a kojom je dozvoljeno umnožavanje, distribucija i javno saopštavanje dela, i prerade, uz adekvatno navođenje imena autora, čak i u komercijalne svrhe.

Potpis autora

U Beogradu, 21.06.2024.

Sažetak

U ovom radu istražuje se značaj Laravel frejmvorka u razvoju veb aplikacija za poslovne sisteme. Počevši od osnovnih karakteristika i prednosti Laravela, rad detaljno analizira primenu ovog frejmvorka kroz izradu tiketing sistema. Kroz praktične primere, opisani su koraci za kreiranje Laravel projekta, rutiranje, korišćenje Blade šablona i direktiva, rad sa bazom podataka koristeći Docker, kao i ostale ključne funkcionalnosti poput validacije podataka, rad sa sesijama i paginacije. Takođe, razmatra se integracija Tailwind CSS okvira za dodavanje stila aplikaciji. Kroz ovo istraživanje, ističe se važnost Laravela kao snažnog alata za razvoj veb aplikacija u poslovnom okruženju.

Ključne reči**:** Laravel, veb aplikacije, poslovni sistemi, tiketing sistem

Abstract

This paper explores the significance of the Laravel framework in the development of web applications for business systems. Starting from the basic features and advantages of Laravel, the paper analyzes in detail the application of this framework through the development of a ticketing system. Through practical examples, the steps for creating a Laravel project, routing, using Blade templates and directives, working with databases using Docker, as well as other key functionalities such as data validation, session management, and pagination are described. Additionally, the integration of the Tailwind CSS framework for styling the application is discussed. This research highlights the importance of Laravel as a powerful tool for developing web applications in a business environment

Keywords: Laravel, web applications, business systems, ticketing system

Sadržaj

UVOD

U današnje vreme veb aplikacije se mogu smatrati ključnim alatima u poslovanju, pružajući rešenja za širok spektar zadataka koji su od suštinskog značaja za efikasno vođenje kompanija. One mogu biti jednostavne, kao što su kalkulatori ili to-do liste, ili kompleksne, kao što su sistemi za upravljanje odnosima s klijentima (CRM) ili planiranje resursa preduzeća (ERP), a u zavisnosti od potreba i ciljeva organizacije.

Veb aplikacije suštinski mogu biti fokusirane na jedan specifičan zadatak i mogu biti relativno jednostavne za razvoj. S druge strane, online tržišta ili sistemi za upravljanje projektima predstavljaju složenije aplikacije koje zahtevaju rad sa različitim izvorima podataka i mogu rešavati mnogo širi spektar zadataka.

Troškovi i trajanje razvoja veb aplikacija variraju u zavisnosti od njihove složenosti i opsega zadataka koje treba rešiti. Razumevanje ovih faktora ključno je za planiranje i implementaciju uspešnih veb projekata.

Razvoj i integracija veb aplikacija obezbeđuje kompanijama nekoliko prednosti koje mogu postati značajni argumenti koji idu u prilog ulaganju u njihov razvoj. Prvenstveno to može biti 24/7 dostupnost, a koja omogućava korisnicima interakciju sa veb aplikacijom putem personalnog računara, laptopa ili pametnog telefona koji imaju pristup internetu. Na primer, posetioci online prodavnice mogu vršiti porudžbine putem veb sajta u bilo koje doba, nezavisno od radnog vremena prodavnice. Ovakav pristup eliminiše geografska ograničenja i omogućava poslovanje na globalnom nivou, uz pristup potencijalima optimizacije za pretraživače (SEO) i višestrukim digitalnim marketinškim mogućnostima.

Zatim, razvoj veb aplikacija je prilično ekonomičan, za razliku od mobilnog razvoja, nije potrebno naručivati više verzija aplikacija za različite platforme. Korisnici mogu da interaguju sa proizvodom putem jednog interfejsa za sve uređaje kao što je veb pretraživač, što značajno ubrzava razvoj i smanjuje troškove.

Takođe i lakoća pristupa poslovnim podacima omogućava da podaci koje korisnici razmenjuju sa veb aplikacijom budu dostupni u realnom vremenu na udaljenom serveru (cloud), a kojima se može pristupiti u bilo koje doba i sa bilo kog uređaja sa internet pristupom. Ovo smanjuje

rizik od gubitka podataka zbog hardverskih kvarova, jer su podaci sigurno smešteni u cloud skladištu i imaju svoje rezervne kopije.

Osim svega ovog, veb aplikacije unapređuju produktivnost i saradnju, tako što omogućavaju razmenu podataka između svih sektora kompanije, olakšavajući pronalaženje i proveru podataka bez potrebe za ličnim sastancima.

Veb aplikacije omogućavaju i automatizaciju brojnih ručnih procesa, povećavajući produktivnost zaposlenih i smanjujući troškove. Korisnici mogu pristupati podacima u realnom vremenu kako bi ubrzali donošenje odluka i unapredili kvalitet odluka.

Nadogradnja i održavanje veb aplikacija su prilično jednostavni, jer su promene na serverskoj strani gotovo odmah dostupne korisnicima. Nove verzije aplikacije ne prolaze kroz proces pregleda i odobravanja sadržaja trećih strana, kao što je slučaj sa mobilnim aplikacijama za iOS ili Android operativni sistem. Održavanje, ažuriranje i proširenje veb aplikacija je mnogo lakše nego kod bilo kog drugog tipa softvera.

Veb aplikacije su između ostalog i visoko skalabilne, čime pružaju fleksibilnost u pogledu rasta. Kada je potrebno povezati više zaposlenih, sve što je potrebno su dodatni serverski resursi, što se lako postiže. Veb aplikacije omogućavaju brze i jednostavne promene, integraciju sa API-jima trećih strana i korišćenje drugih korisnih funkcija, što ih čini neprocenjivim alatom za kompanije koje rastu i prilagođavaju se promenama u poslovnom okruženju.

Veb aplikacije su izuzetno pogodne za poslovne sisteme iz više razloga. Prvenstveno, one omogućavaju univerzalnu dostupnost putem veb pretraživača na različitim uređajima, uključujući računare, tablete i pametne telefone. Ovo korisnicima omogućava pristup poslovnom sistemu sa bilo kog mesta i u bilo koje vreme, povećavajući fleksibilnost i produktivnost.

Takođe, veb aplikacije zahtevaju lakšu implementaciju i održavanje u poređenju sa desktop aplikacijama. Instalacija i ažuriranje su jednostavniji jer ne zahtevaju distribuciju i instalaciju softvera na svakom korisničkom uređaju. Održavanje je takođe olakšano jer se promene mogu primeniti na serveru centralno, bez potrebe za nadogradnjom svakog korisničkog uređaja. Osim toga, veb aplikacije smanjuju troškove poslovanja jer ne zahtevaju kupovinu i održavanje

skupih desktop računara ili licenci za softver. Korisnici mogu koristiti svoje postojeće uređaje za pristup aplikaciji, što dodatno smanjuje troškove.

Veb aplikacije su i skalabilne, što znači da se lako mogu prilagoditi porastu broja korisnika ili zahteva. Dodavanje novih servera ili resursa može se relativno brzo implementirati kako bi se osiguralo neprekidno funkcionisanje poslovnog sistema čak i u uslovima povećane opterećenosti.

Još jedan ključni faktor u korist veb aplikacija i njihovoj primeni za poslovne sisteme jeste bezbednost. Veb aplikacije omogućavaju implementaciju različitih sigurnosnih mehanizama kao što je autentifikacija korisnika, enkripcija podataka i kontrola pristupa, čime se osigurava sistem i samo ovlašćeni korisnici imaju pristup važnim resursima.

Između ostalog veb aplikacije olakšavaju integraciju sa postojećim poslovnim sistemima, poput ERP ili CRM sistema, putem standardnih protokola i API-ja. Ovo pojednostavljuje razmenu podataka između različitih sistema i poboljšava efikasnost poslovanja. U suštini, veb aplikacije pružaju brojne prednosti poslovnim sistemima, poboljšavajući efikasnost poslovanja i korisničko iskustvo.

Kada je reč o izradi veb aplikacija i uopšte o veb razvoju, tada se može reći da je veoma velika primena jezika PHP (Hypertext Preprocessor), kao široko korišćenog programskog jezika posebno dizajniranog za veb razvoj. Zbog svoje jednostavnosti, PHP je idealan i za početnike u programiranju, nudeći jasnu i logičnu sintaksu koja olakšava brzo usvajanje osnovnih programerskih principa. Razvijen sa fokusom na veb aplikacije, PHP omogućava efikasno upravljanje sesijama, obradu formulara i laku interakciju sa različitim bazama podataka, što ga čini posebno pogodnim za aplikacije koje zahtevaju česte operacije s bazama podataka. PHP se najčešće koristi sa MySQL bazama podataka, ali podržava i druge sisteme za upravljanje bazama podataka kao što su PostgreSQL, Oracle i Microsoft SQL Server.

PHP ima ogromnu zajednicu programera, što osigurava veliki broj dostupnih resursa, biblioteka i alata koji mogu pomoći u ubrzavanju razvoja i rešavanju potencijalnih problema. Jednostavna integracija sa većinom veb servera, uključujući Apache, dodatno povećava njegovu adaptabilnost i popularnost u različitim hosting okruženjima.

Kao tehnologija otvorenog koda, PHP je besplatan za upotrebu i modifikaciju, što ga čini privlačnim izborom zbog odsustva licencnih troškova. Njegova fleksibilnost omogućava programerima da kreiraju kako jednostavne tako i kompleksne veb aplikacije, te da ga koriste u kombinaciji sa različitim frontend tehnologijama kao što su HTML, CSS i JavaScript. Uprkos pojavi novijih tehnologija, PHP je i dalje relevantan u svetu veb razvoja zahvaljujući svojoj sposobnosti da se prilagodi i integriše sa novim alatima i praksama.

Da bi se pravile aplikacije visokog kvaliteta pomoću PHP jezika, najčešće se koristi Laravel okvir (framework) kao jedan od najpopularnijih i najmoćnijih okvira koji se koristi za razvoj veb aplikacija.

Laravel je popularan PHP okvir za razvoj veb aplikacija koji se sve više koristi u izradi poslovnih sistema. Ovaj okvir nudi programerima mnoge alate i funkcionalnosti koje olakšavaju proces razvoja i održavanja kompleksnih veb aplikacija.

Primenom Laravel okvira u izradi veb aplikacija za poslovne sisteme, programeri mogu brže i efikasnije kreirati robustna i skalabilna softverska rešenja. Laravel pruža podršku za autentifikaciju korisnika, upravljanje korisničkim pravima, interakciju sa bazom podataka, rutiranje, testiranje i mnoge druge funkcionalnosti koje su od suštinskog značaja za poslovne aplikacije.

Jedna od ključnih prednosti korišćenja Laravela za poslovne sisteme je njegova modularnost i fleksibilnost. Programeri mogu lako integrisati različite komponente i biblioteke kako bi prilagodili aplikaciju specifičnim potrebama poslovnog okruženja. Takođe, Laravel pruža čistu i intuitivnu sintaksu, što olakšava održavanje koda i saradnju između članova tima.

Pored ovog, Laravel zajednica je aktivna i podržava razmenu iskustava, razvoj novih funkcionalnosti i rešavanje problema koji se javljaju tokom razvoja veb aplikacija. To omogućava programerima da brže rešavaju izazove i da se usredsrede na implementaciju poslovnih zahteva umesto na rešavanje tehničkih problema.

Suštinski, primena Laravel okvira u izradi veb aplikacija za poslovne sisteme donosi mnoge prednosti kao što su brži razvojni ciklusi, veća skalabilnost, bolja održivost i lakše upravljanje kompleksnim projektima. Zbog ove činjenic sve više kompanija i razvojnih timova bira Laravel kao okvir za izgradnju sofisticiranih poslovnih rešenja.

1. Značaj veb aplikacija u poslovnim sistemima

Poslovni sistemi su ključni za uspešno upravljanje organizacijom, omogućavajući koordinaciju poslovnih procesa, efikasno upravljanje resursima i informacijama kako bi se ostvarili ciljevi organizacije. Oni integrišu različite funkcije unutar preduzeća, uključujući proizvodnju, distribuciju, prodaju, marketing i finansije, što doprinosi boljoj efikasnosti i koordinaciji. Automatizacija rutinskih zadataka i procesa u okviru ovih sistema smanjuje potrebu za manuelnim radom, povećavajući time efikasnost i preciznost operacija.

Pored interne organizacije, poslovni sistemi olakšavaju upravljanje ljudskim, finansijskim, materijalnim i informacionim resursima, što je ključno za optimalno funkcionisanje preduzeća. Podrška odlučivanju je još jedna važna funkcija poslovnih sistema, omogućavajući liderima da donose starteške odluke koristeći analizu podataka i poslovnu inteligenciju. Ovo pomaže u praćenju performansi organizacije, identifikaciji trendova i prilagođavanju strategija za postizanje poslovnih ciljeva.

Integracija sa eksternim partnerima kao što su dobavljači, kupci i drugi poslovni partneri takođe je značajan aspekt poslovnih sistema, jer poboljšava razmenu podataka i informacija, olakšavajući time saradnju i efikasnost poslovnih procesa. Sve ove komponente zajedno čine poslovne sisteme vitalnim za efikasnost, produktivnost i konkurentnost svake organizacije.

Veb aplikacije su ključne za poslovanje iz nekoliko važnih razloga. Prvo, omogućavaju globalnu dostupnost i pristupačnost, dopuštajući zaposlenima da pristupe poslovnim sistemima sa bilo kog mesta i uređaja koji ima pristup internetu. Ovo unapređuje fleksibilnost u radnom okruženju i povećava produktivnost. Takođe, korišćenje veb aplikacija može smanjiti troškove poslovanja jer ne zahteva velike investicije u hardver i infrastrukturu.

Jedna od ključnih prednosti veb aplikacija je mogućnost brzog ažuriranja i implementacije promena. To omogućava agilno poslovanje i brzu reakciju na promene na tržištu ili unutar organizacije, bez potrebe za ručnom instalacijom na svakom uređaju. Takođe, veb aplikacije omogućavaju jednostavnu integraciju sa drugim softverskim sistemima i servisima, olakšavajući razmenu podataka i koordinaciju aktivnosti između različitih delova organizacije.

Dodatno, veb aplikacije omogućavaju personalizovano iskustvo za korisnike, prilagođavajući se njihovim potrebama i preferencijama, što može povećati angažovanost i zadovoljstvo korisnika.

U suštini veb aplikacije su neophodan alat za modernizaciju i unapređenje poslovanja, pružajući pristupačnost, efikasnost, fleksibilnost, integraciju, personalizaciju i analitiku potrebnu za uspeh u današnjem digitalnom svetu.

U ranim danima razvoja veba, stvaranje veb aplikacija zahtevalo je od programera da pažljivo napišu kod za svaki aspekt aplikacije, od jedinstvene poslovne logike do uobičajenih funkcionalnosti poput autentifikacije korisnika, provere unosa i pristupa bazi podataka. Svaka komponenta je morala da bude izgrađena ispočetka, što je rezultiralo dugim ciklusima razvoja i velikim naporom.

Međutim, sistem razvoja veba se od tada drastično promenio. Danas, programeri imaju pristup velikom broju okvira za razvoj aplikacija i biblioteka koje olakšavaju proces razvoja. Ovi okviri nude unapred izgrađena rešenja za mnoge uobičajene zadatke, omogućavajući programerima da se više fokusiraju na jedinstvene karakteristike svojih aplikacija.

I pored obilja dostupnih okvira, programeri često nailaze na dilemu u izboru pravog. Brz tempo tehnološkog napretka znači da kada se programer osposobi u jednom okviru, u međuvremenu se pojave novi, moguće altrenativno bolji i privlačniji u smislu promene.

U situaciji velikog broja opcija, postavlja se pitanje: zašto uopšte koristiti okvire? Šta opravdava njihovo usvajanje u odnosu na ručno kodiranje ili druge alternative? Ovo pitanje postaje još važnije kada se razmatra određeni okvir kao što je Laravel. Šta izdvaja Laravel od svojih konkurenata i zašto bi programeri trebalo da se odluče za njega umesto za druge? Odgovor leži u razumevanju prednosti koje okviri nude. Pružajući standardizovanu strukturu i set alata, okviri poput Laravela omogućavaju programerima da izgrade robustne, skalabilne i održive aplikacije sa većom efikasnošću. Apstrahuju složenosti uobičajenih zadataka, omogućavajući programerima da se fokusiraju na implementaciju jedinstvene poslovne logike i pruže vrednost krajnjim korisnicima.

U suštini, iako obilje okvira može delovati naporno zbog izbora, svaki okvir ima svoju svrhu i svoj set prednosti i nedostataka. Pažljivom procenom zahteva projekta i mogućnosti različitih okvira, programeri mogu donositi važne odluke koje vode uspešnom razvoju veb aplikacija.

2. Laravel frejmvork i njegove karakteristike

Veb programiranje je proces kreiranja i održavanja veb sajtova, veb aplikacija i drugih digitalnih proizvoda kojima se pristupa preko interneta. Veb programiranje uključuje korišćenje programskih jezika i alata za dizajniranje i razvijanje veb baziranih proizvoda. Veb programiranje takođe uključuje korišćenje raznih okvira i alata za veb razvoj koji pojednostavljuju i automatizuju proces razvoja. Primeri popularnih okvira za veb razvoj uključuju Angular, React i Vue za front-end razvoj, kao i Laravel, Django i Ruby on Rails za back-end razvoj.

Velika prednost PHP programskog jezika je što je teško naći veb server koji ne podržava PHP. Međutim, savremeni alati za PHP imaju strože zahteve nego ranije. Iz tog razloga je Laravel okvir jedan od najboljih alata za razvoj, gde se može osigurati usklađeno lokalno i udaljeno serversko okruženje za PHP kod.

2.1 Uvod u Laravel frejmvork

Laravel je kreirao Taylor Otwell 2011. godine. U pitanju je besplatan PHP veb okvir otvorenog koda, dizajniran da pojednostavi i ubrza razvoj veb aplikacija. Laravel se pridržava arhitektonskog obrasca MVC (Model-View-Controller)[1] i uključuje standarde kao što su PSR-2 za kodiranje i PSR-4 za automatsko učitavanje. Ovaj okvir je posebno poznat po svojoj pogodnosti za razvoj softvera vođen testiranjem (TDD - Test Driven Development), koji je jednostavan za implementaciju u Laravel-u.

Laravel je dostupan na GitHub platformi, što omogućava korisnicima da pristupe kodu, preuzmu ga i doprinesu izradi projekta kroz različite interakcije poput predlaganja izmena ili prijavljivanja problema. Ovo hostovanje na GitHub-u čini Laravel lako dostupnim i otvorenim za kolaboraciju sa globalnom zajednicom razvojnih programera.

Laravel je dizajniran tako da podržava arhitekturu zasnovanu na mikroservisima, što znači da se aplikacija može graditi kao skup manjih, nezavisnih servisa. Ovi servisi su specijalizovani za određene funkcije i komuniciraju međusobno putem definisanih interfejsa, najčešće kroz API-je. Ovakav pristup omogućava fleksibilnost i efikasnost u razvoju i održavanju aplikacija,

[1] obrazac dizajna softvera koji se obično koristi za razvoj korisničkih interfejsa koji deli srodnu programsku logiku na tri međusobno povezana elementa.

posebno u složenim sistemima gde različiti aspekti aplikacije zahtevaju različite tehnologije i pristupe.

Između ostalog Laravel nudi veliku proširivost kroz upotrebu prilagođenih ili postojećih paketa trećih strana. Ovi paketi mogu lako da se integrišu u aplikaciju, omogućavajući razvojnim programerima da prošire funkcionalnosti bez potrebe za pisanjem sopstvenog obimnog koda. To Laravel-u daje mogućnost da se brzo prilagodi različitim potrebama projekata, čineći ga izuzetno prilagodljivim i moćnim alatom za razvoj veb aplikacija.

2.2 Osnovne karakteristike i prednosti Laravel-a

Osnovni principi Laravel-a uključuju MVC arhitekturu (Model-View-Controller), koja omogućava jasno razdvajanje logike aplikacije od korisničkog interfejsa. Ovo olakšava održavanje i skaliranje aplikacija, jer promene u jednom delu aplikacije ne utiču direktno na druge delove.

Jedan od ključnih alata Laravel-a je *Eloquent ORM* (Object-Relational Mapping), koji omogućava rad sa bazom podataka koristeći objektno-orijentisane modele. Ovo olakšava manipulaciju podacima i izgradnju složenih upita, pružajući programerima intuitivan način za interakciju sa bazom podataka.

Pored toga, Laravel dolazi sa bogatim setom ugrađenih alata i funkcionalnosti, uključujući autentifikaciju, rutiranje, sesije, validaciju i još mnogo toga. Ovi alati omogućavaju programerima da brzo razvijaju funkcionalnosti svojih aplikacija bez potrebe za pisanjem velike količine koda.

Još jedan važan alat Laravel-a je *Composer*, moćan alat za upravljanje zavisnostima i paketima, gde su zavisnosti biblioteke ili paketi koda koje veb aplikacija koristi kako bi dodala funkcionalnosti bez potrebe za ponovnim pisanjem koda. *Composer* omogućava lako dodavanje eksternih biblioteka i komponenti u Laravel projekte, što omogućava programerima da koriste najnovije tehnologije i rešenja u svojim aplikacijama.

Ukratko, Laravel je moćan i sveobuhvatan okvir koji olakšava razvoj modernih veb aplikacija. Njegova jednostavna sintaksa, bogat set alata i aktivna zajednica čine ga idealnim izborom za programere koji žele brzo i efikasno da razvijaju visoko kvalitetne aplikacije.

U Laravel razvoju, pored uobičajenih *backend* tehnika, često se koriste i *frontend* tehnologije kako bi se poboljšao korisnički doživljaj i estetika aplikacija.

Backend se odnosi na deo aplikacije koji radi na serveru i upravlja poslovnom logikom, bazama podataka, autentifikacijom korisnika, i drugim funkcionalnostima koje su nevidljive korisniku. U Laravel-u, *backend* se razvija koristeći PHP i Laravel okvir, gde se definišu rute, kontroleri, modeli i drugi aspekti *server-side* logike.

Frontend se odnosi na deo aplikacije koji je vidljiv korisniku i sa kojim korisnik direktno komunicira. Ovo uključuje dizajn stranica, korisnički interfejs (UI) i sve dinamičke elemente prikazane u pretraživaču. U Laravel aplikacijama, *frontend* se često razvija korišćenjem HTML-a, CSS-a, JavaScript-a i *frontend* okvira kao što su *Vue.js* i *Tailwind CSS.*

Vue.js je progresivni JavaScript okvir za izgradnju korisničkih interfejsa. On se često koristi u Laravel projektima zbog svoje jednostavnosti, performansi i mogućnosti reaktivnog programiranja. *Vue.js* omogućava lakšu organizaciju koda i pruža mogućnost primene komponentne arhitekture, što ga čini idealnim za izgradnju dinamičnih i interaktivnih korisničkih interfejsa. Integracija *Vue.js*-a sa Laravel-om je jednostavna i omogućava programerima da brzo razvijaju kompleksne *frontend* funkcionalnosti.

Tailwind CSS je moderni CSS okvir koji omogućava brzo stilizovanje veb aplikacija pomoću klase-baziranog pristupa. Ovaj pristup znači da umesto korišćenja predefinisanih komponenti ili stilova, *Tailwind CSS* omogućava programerima da definišu stilove direktno u HTML-u pomoću predefinisanih klasa. Ovo olakšava prilagođavanje stila aplikacije i omogućava veću fleksibilnost u dizajnu. *Tailwind CSS* se često koristi u Laravel projektima zbog svoje jednostavnosti, skalabilnosti i mogućnosti prilagođavanja, što ga čini popularnim izborom među programerima. Integracija *Tailwind CSS*-a sa Laravel-om je jednostavna i omogućava programerima da brzo stilizuju svoje veb aplikacije u skladu sa dizajnerskim smernicama.

Još jedan od alata jeste *Blade* koji se nalazi u Laravel ekosistemu i koji omogućava programerima da efikasno generišu HTML kod koristeći PHP sintaksu. Ključne osobine i načini na koje se može koristiti *Blade* u Laravel projektima su sledeći:

- **Jednostavnost i preglednost**. *Blade* pruža čistu i intuitivnu sintaksu za pisanje HTML-a, što olakšava razvoj korisničkog interfejsa. Programeri mogu koristiti uobičajene PHP

kontrolne strukture poput *if-else*, *foreach* petlji i *include* direktiva za dinamičko generisanje HTML-a.

- **Nasleđivanje layout-a**. *Blade* omogućava programerima da definišu osnovne *layout* datoteke koje se nasleđuju na drugim stranicama. Ovo omogućava konzistentan izgled i strukturu aplikacije, olakšavajući održavanje i skaliranje.

- **Kompaktan i fleksibilan kod**. *Blade* omogućava uključivanje promenljivih, komponenti i delova stranice na jednostavan način. Ovo omogućava programerima da rešavaju kompleksne zadatke sa minimalnim naporom, čineći kod čitljivijim i efikasnijim.

- **Direktive i kontrolne strukture**. Blade pruža različite direktive i kontrolne strukture poput @if, @foreach, @include, @extends, @yield i mnoge druge. Ove direktive olakšavaju manipulaciju podacima i organizaciju koda, omogućavajući programerima da brzo i efikasno razvijaju dinamičke veb stranice.

- **Integracija sa Laravel-om**. *Blade* je integrisan direktno u Laravel, što olakšava njegovo korišćenje u projektima. Laravel pruža podršku za *Blade* komponente, što omogućava programerima da organizuju i ponovo koriste delove koda na efikasan način.

U suštini, *Blade* je moćan alat koji olakšava generisanje dinamičkog HTML koda u Laravel projektima. Njegova jednostavna sintaksa, fleksibilnost i integracija sa Laravel-om čine ga popularnim izborom među programerima za izgradnju korisničkog interfejsa u veb aplikacijama.

2.3 Primena realnih aplikacija razvijenih u Laravel-u

U primeni realnih aplikacija razvijenih u Laravel-u, ovaj moćan PHP okvir pokazuje svoju snagu i sposobnost da se nosi sa širokim spektrom zahteva u stvarnom svetu. Nadalje će se navesti nekoliko primera kako se Laravel koristi u različitim scenarijima:

E-trgovina. Laravel se često koristi za izgradnju platformi za elektronsku trgovinu zbog svoje sposobnosti da efikasno upravlja kompleksnim transakcijama, inventarom proizvoda i korisničkim nalozima. Sa ugrađenom podrškom za autentifikaciju, autorizaciju i plaćanja,

Laravel omogućava programerima da brzo razvijaju sigurne i pouzdane aplikacije za elektronsku trgovinu.

Administrativni paneli. Laravel je popularan izbor za izgradnju administrativnih panela i upravljačkih tabli zbog svoje jasne i pregledne arhitekture. Sa mogućnošću generisanja brzih i efikasnih CRUD (**C**reate, **R**ead, **U**pdate, **D**elete) operacija, Laravel omogućava programerima da brzo razvijaju aplikacije za upravljanje sadržajem, korisnicima i drugim resursima.

Socijalne mreže. Laravel se često koristi za izgradnju socijalnih mreža zbog svoje sposobnosti da efikasno upravlja kompleksnim mrežama veza, aktivnostima korisnika i generisanjem vesti. Sa podrškom za autentifikaciju, autorizaciju i slanje obaveštenja, Laravel omogućava programerima da brzo razvijaju interaktivne socijalne platforme.

Rezervacije i planiranje. Laravel se često koristi za izgradnju aplikacija za rezervaciju i planiranje zbog svoje sposobnosti da efikasno upravlja vremenom, datumima i rasporedima. Sa podrškom za kalendare, notifikacije i automatizaciju zadataka, Laravel omogućava programerima da brzo razvijaju aplikacije za rezervaciju karata, događaja i drugih resursa.

Obrazovanje. Laravel se koristi i u oblasti obrazovanja za izgradnju platformi za učenje na daljinu, sisteme za upravljanje kursevima i alate za proveru znanja studenata. Sa podrškom za korisničke panele, ocene, testove i analitiku, Laravel omogućava programerima da brzo razvijaju interaktivne i obrazovne aplikacije.

U svakom od ovih scenarija, Laravel pruža moćne alate i funkcionalnosti koji olakšavaju razvoj i održavanje visoko kvalitetnih veb aplikacija. Sa svojom jednostavnom sintaksom, bogatim ekosistemom i aktivnom zajednicom, Laravel je veoma popularan izbor među programerima širom sveta za razvoj realnih aplikacija.

3. Primena Laravel-a u razvoju tiketing sistema

Primenu Laravel-a je najefikasnije objasniti na primeru razvoja nekog softvera. Primer koji će poslužiti u svrhe objašnjavanja primene Laravel-a i konačno za potrebe ovog rada jeste razvoj tiketing sistema. Tiketing sistemi su neophodni alati za organizaciju i praćenje zadataka, zahteva i podrške unutar različitih organizacija. Kroz niz koraka, objasniće se ključni aspekti razvoja takvog sistema, uključujući pravljenje projekta, rutiranje, kreiranje i korišćenje *Blade* šablona, rad sa bazama podataka, validacija podataka, upravljanje sesijama, i dodavanje stilova. Svaki korak će pružiti jasne smernice za implementaciju funkcionalnog i efikasnog tiketing sistema koristeći Laravel.

3.1 Pravljenje Laravel projekta

Proces izdrade jednog Laravel projekta biće predstavljen kroz detaljno navođenje svih koraka u postupku. Projekat se nalazi na sledećoj veb adresi: https://srv542162.hstgr.cloud/.

Korak 1: Instalacija Composer-a na Ubuntu operativnom sistemu

Ubuntu operativni sistem se uobičajeno koristi jer se 80% do 90% veb aplikacija nalazi na Linux serverima, zahvaljujući njihovoj stabilnosti, sigurnosti, isplativosti i činjenici da su otvorenog koda. Instalacija *Composer*-a na Ubuntu operativnom sistemu je prvi i ključan korak za razvoj PHP aplikacija:

1. Ažuriranje paket lista

Pre početka instalacije, potrebno je ažurirati listu dostupnih paketa na Ubuntu sistemu kako bi se osiguralo da su sve informacije o paketima ažurne. Ovo se postiže izvršavanjem komande: **`sudo apt update`**.

2. ***Instalacija PHP-a i dodatnih ekstenzija***

Composer je alat napisan u PHP-u, stoga je neophodno imati instaliran PHP na sistemu, zajedno sa određenim proširenjima. U okviru ovoga se instalira PHP i sve potrebne ekstenzije.

3. Preuzimanje *Composer* instalacione skripte

Preuzimanje instalacione skripte se vrši sa zvanične *Composer* stranice. Sledeća komanda će preuzeti instalacioni skript sa veb stranice:

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
```

4. ***Provera integriteta instalacione skripte***

Pre nastavka, neophodno je da se proveri integritet preuzete instalacione skripte.

5. Instalacija *Composer*-a

Sledeći korak je instalacija *Composer*-a, koja se vrši izvršavanjem sledeće komande: **`sudo php composer-setup.php --install-dir=/usr/local/bin --filename=composer`**

6. ***Provera Instalacije***

Konačno, da bi se potvrdilo da je *Composer* uspešno instaliran, izvršava se sledeća komanda: **`composer --version`**

Ova komanda će prikazati trenutnu verziju *Composer*-a ukoliko je instalacija bila uspešna.

Korak 2: Instalacija Laravel okvira

Laravel dolazi sa sopstvenim instalacionim alatom koji olakšava stvaranje novih projekata. Da bi se instalirao Laravel, koristiće se *Composer* koji je prethodno već instaliran. U terminalu će se izvršiti sledeća komanda: **`composer create-project laravel/laravel task-list`.** Ova komanda će kreirati novi Laravel projekat u folderu *task-list*. Važno je napomenuti da ime foldera nije unapred određeno i može biti prilagođeno potrebama ili svrsi projekta. Na primer, umesto "task-list", može se odabrati bilo koje drugo ime koje odgovara projektu koji se kreira.

Korak 3: Instalacija Node-a i NPM-a

Budući da će ovaj projekat koristiti *Tailwind CSS*, potrebno je da se instalira *Node.js* i NPM (Node Package Manager), zbog toga što *Tailwind CSS* koristi NPM kao mehanizam za upravljanje zavisnostima i paketima. Slično kao sto *Composer* upravlja zavisnostima i paketima u PHP-u. *Node.js* je potreban kako bi se pokrenuo NPM, a pomoću NPM-a se instalira *Tailwind CSS* kao jedan od paketa.

Proveriće se da li je uspešno instaliran Node.js pomoću komande **`node -v`**. Preostalo je da se instalira NMP što će se postići izvršavanjem sledeće komande: **`sudo apt install npm`**.

Korak 4: Pokretanje Laravel razvojnog servera

Pošto je kreiran projekat, može se pokrenuti Laravel razvojni server kako bi lokalno testirala aplikacija. U direktorijumu gde se nalazi Laravel projekat izvršiće se sledeća komanda:

```
Php artisan serve
```

Komanda *php artisan serve* kreira lokalni server na kome će biti pokrenut Laravel projekat. Posle izvršenja ove komande može se pristupiti projektu otvaranjem veb pretraživača putem linka **127.0.0.1:8000** ili **localhost:8000.** Korišćenjem localhost-a, pretraživač se zapravo povezuje sa sopstvenim računarom. IP adresa 127.0.0.1 je specifična IP adresa koja se koristi za identifikaciju lokalnog računara na mreži. Ova adresa se naziva "loopback" adresa, što znači da se koristi za usmeravanje saobraćaja nazad na isti uređaj na kojem je zahtev generisan.

Otvoriće se još jedan terminal gde je definisan projekat i izvršiće se komanda **`npm run dev`**. Ova komanda se koristi u razvojnom (dev) okruženju za izvršavanje različitih zadatka koji se nalaze u *package.json* datoteci u delu "scripts".

U Laravel projektima, ova komanda se obično koristi za pokretanje skripti za kompilaciju, optimizaciju i organizaciju resursa kao što su CSS, JavaScript i druge stavke, koje su potrebne za ispravno prikazivanje aplikacije.

3.2 Sve o rutiranju

Rutiranje predstavlja ključni aspekt u Laravel okviru, omogućavajući definisanje URL ruta za obradu HTTP zahteva. Ovaj koncept omogućava usmeravanje korisničkih zahteva ka odgovarajućim delovima aplikacije, pokrećući odgovarajuće akcije.

Definisanje ruta

U Laravel-u, definisanje ruta obično se obavlja u *routes/web.php* datoteci za veb rute i *routes/api.php* datoteci za API rute. Koristeći globalni *Route* objekat, mogu se koristiti metode poput get, post, put, delete za definisanje ruta. Route objekat u Laravel-u predstavlja središnju komponentu koja omogućava definisanje ruta za veb aplikaciju.

```
Route::get('/putanja, 'Kontroler@metoda');
```

Kako je navedeno, /putanja predstavlja URL putanju, dok 'Kontroler@metoda' označava kontroler i metodu koji će biti pozvani kada se bude zahtevala ta ruta.

U primeru veb aplikacije koja je napravljena za potrebe ovog rada, kada korisnik pristupi stranici *localhost/profile*, automatski će se pokrenuti metoda *edit*, koja se nalazi u klasi *ProfileController*, a koja je definisana u ruti:

```
Route::get('/profile',[ProfileController::class,'edit'])
    ->name ('profile.edit').
```

Parametri rute i callback funkcije

Rute u Laravel-u mogu imati dinamičke parametre koji se prosleđuju kroz URL putanju. Ovi parametri se obično označavaju u obliku {parametar} u definiciji rute.

```
Route::get('/korisnik/{id}', 'KorisnikController@show');
```

Ovde, {id} predstavlja dinamički parametar koji će biti prosleđen *show* metodi KorisnikController-a. Međutim, naziv metode je mogao biti drugačiji, na primer *display* ili *view*. U tom slučaju drugi argument *Route::get* metode bi bio ‘KorisnikController@display’ ili ‘KorisnikContoller@view’. Važno je samo da ime metode odgovara funkcionalnosti koju obavlja.

U task list veb aplikaciji definisana je ruta :

```
Route::get('tasks/{task}', function (Task $task) {
    return view('task', [
    'task' => $task
    ]);
})->middleware(['auth', 'verified'])->name('tasks.show');
```

Kao što se može videti, nije uvek obavezno da se kreira kontroler i metoda kako bi se izvršila određena akciju kada korisnik pristupi određenoj ruti. Umesto toga, može da se koristi *callback* funkcija direktno u definiciji rute. *Callback* funkcija je funkcija koja se prosleđuje kao argument drugoj funkciji i poziva se unutar te druge funkcije.

Ovo je posebno korisno kada je zadatak koji treba obaviti prilikom pristupa ruti relativno jednostavan ili mali. Umesto da se kreira poseban kontroler i metoda, može da se direktno u definiciji rute navede anonimna funkcija koja će izvršiti neophodnu akciju.

Na primer, u slučaju ove aplikacije, definisana je ruta: **`Route::get('tasks/{task}', function (Task $task){...})`**. Ova ruta koristi anonimnu funkciju kao *callback*, koja prima instancu *Task* modela kao argument. Ova funkcija može izvršiti sve potrebne radnje u vezi sa prikazivanjem informacija o zadatku direktno unutar svoje definicije, bez potrebe za kreiranjem posebnog kontrolera i metoda.

Korišćenje *callback* funkcija na ovaj način omogućava da se brzo i efikasno reaguje na zahteve korisnika za određenim rutama, čime se olakšava implementacija manjih zadataka u aplikaciji.

Imenovanje ruta

Rute se mogu imenovati kako bi se omogućilo lakše referenciranje iz drugih delova aplikacije. Ovo je posebno korisno kada se generišu URL putanje u aplikaciji.

```
Route::get('/profil', 'ProfilController@show')>name('profil');
```

Kako je navedno, ruta je imenovana kao 'profil', što omogućava generisanje URL-a korišćenjem imena rute (*route('profil')*) umesto unosa apsolutne putanje URL-a.

Grupisanje ruta

Laravel omogućava grupisanje sličnih ruta kako bi se primenile zajedničke osobine, kao što su *middleware*, prefiksi ruta, itd.

```
Route::prefix('admin')->middleware('auth')->group(function () {
   Route::get('/dashboard', 'AdminController@dashboard');
   Route::get('/korisnici', 'AdminController@korisnici');
});
```

Prefix('admin') metoda postavlja prefiks za sve rute unutar grupe. U ovom slučaju, sve rute unutar grupe će počinjati sa */admin*. To znači da će *ruta /dashboard* postati */admin/dashboard*, a ruta */korisnici* postati */admin/korisnici*.

Middleware('auth') metoda dodaje *middleware* na sve rute unutar grupe. *Middleware* se koristi za filtriranje ili preusmeravanje zahteva pre nego što dospe do rute. U ovom slučaju, koristi se middleware 'auth', što znači da će sve rute unutar grupe biti dostupne samo autorizovanim korisnicima.

Može se videti da će sve rute unutar grupe *admin* imati prefiks */admin* u URL-u i zahtevati autentifikaciju kroz *auth middleware*.

Rutiranje je osnovni koncept u Laravel-u koji omogućava organizovanje i upravljanje navigacijom u aplikaciji. Razumevanje rutiranja je ključno za izgradnju robustnih i skalabilnih Laravel aplikacija.

3.3 Blade šablon

Blade je esencijalni deo Laravel ekosistema koji pruža programerima mogućnosti korišćenja moćnih alata za generisanje dinamičkog HTML-a. Ovaj sistem ne samo što olakšava kreiranje i manipulisanje HTML-om, već pruža i elegantan način za uključivanje PHP logike.

Sintaksa Blade-a

Blade se ističe jednostavnom i lako čitljivom sintaksom koja omogućava uključivanje dinamičkih podataka, petlji, uslovnih izjava i drugih konstrukcija direktno u HTML kod. Ovo omogućava veću fleksibilnost i kontrolu nad prezentacijom podataka.

Blade fajlovi se inače kreiraju u folderu: *root-projekta/resources/views*. Važno je napomenuti da svaki *Blade* fajl mora imati ekstenziju *.blade.php*.

Nasleđivanje layout-a

Jedna od najkorisnijih karakteristika *Blade*-a je sposobnost nasleđivanja *layout*-a ili šablona. Ovo omogućava definisanje osnovnog *layout*-a koji sadrži zajedničke elemente stranice, poput zaglavlja i podnožja, koji se zatim mogu naslediti i proširiti u specifičnim prikazima.

Layout šabloni se kreiraju u folderu *root-projekta/resources/layouts*. U primeru ove aplikacije definisan je *layout* pod nazivom *app.blade.php*. Da bi se uključio ovaj layout u neki šablon, koristi se sledeća sintaksa: **`<x-app-layout></x-app-layout>`.**

Uključivanje delova HTML-a

Blade takođe omogućava uključivanje delova HTML-a iz drugih fajlova koristeći *@include* direktivu. Ovo je korisno za ponovnu upotrebu koda. Na primer, ako postoji zaglavlje sekcija koja treba da se koristi na svakoj stranici veb sajta, sekcija će se definisati u zasebnom *Blade* fajlu, *header.blade.php* i zatim uključiti na svakoj stranici koristeći @include('header').

Komponente i slotovi

U novijim verzijama Laravel-a, uvedene su komponente i slotovi kao naprednije metode organizacije *Blade* koda. Komponente omogućavaju pakovanje HTML-a i PHP logike u ponovno upotrebljive delove, dok slotovi omogućavaju dinamičko umetanje sadržaja unutar komponenti.

Prednosti *Blade* šablona

Osnovne prednosti *Blade* šablona koje se mogu navesti su sledeće:

- **Čitljivost i lakoća**. Sintaksa *Blade*-a je čista i laka za razumevanje, što olakšava održavanje i razvoj aplikacija.
- **Fleksibilnost**. *Blade* pruža širok spektar mogućnosti za manipulaciju HTML-om i PHP logikom, omogućavajući programerima da brzo i efikasno reaguju na zahteve aplikacije.

- **Ubrzan razvoj**. Nasleđivanje *layout*-a, pripajanje delova *frontend* koda i korišćenje komponenti značajno ubrzava proces razvoja aplikacije, štedeći vreme i resurse.

Blade šabloni su neizostavan deo Laravel ekosistema, pružajući elegantno rešenje za generisanje dinamičkog HTML-a u Laravel aplikacijama. Njihova jednostavna sintaksa, fleksibilnost i bogatstvo funkcionalnosti čine ih neprocenjivim alatom u razvoju modernih veb aplikacija.

3.4 Blade direktive

Blade direktive su specijalne oznake u Laravel *Blade* šablon sistemu koje omogućavaju dodavanje dinamičkih funkcionalnosti i PHP logike direktno u HTML kod. Ove direktive čine pisanje i čitanje *Blade* fajlova intuitivnijim i efikasnijim, olakšavajući programerima rad na *frontend* delu aplikacije. Sledi nekoliko uobičajenih *Blade* direktiva i njihovih funkcionalnosti:

1. @if, @else, @elseif

Direktive @if, @else, i @elseif omogućavaju implementaciju uslovnih izjava direktno unutar *Blade* fajlova, omogućavajući prikazivanje ili skrivanje delova HTML-a na osnovu zadatih uslova. Ovo je osnovni mehanizam za kontrolu toka u *Blade* šablonima.

2. @foreach

Direktiva @foreach se koristi za iteraciju kroz nizove ili kolekcije podataka, generišući HTML za svaki element niza. Ovo je izuzetno korisno za prikaz liste stavki, kao što su korisnici, proizvodi ili blog postovi.

3. @include

@include omogućava uključivanje i ponovno korišćenje drugih *Blade* fajlova unutar trenutnog fajla. Ovo pomaže u organizaciji koda i smanjenju ponavljanja koda, omogućavajući da se delovi šablona, kao što su zaglavlje ili navigacija, izdvoje u zasebne fajlove.

4. @yield, @section, @extends

Ove direktive se koriste za nasleđivanje i proširivanje *layout*-a. @extends se koristi u šablonu dete (child) kako bi nasledio *layout* iz roditeljskog (master) šablona. @section se koristi za definisanje delova sadržaja koji se unose u odgovarajuće @yield mesto u master šablonu.

5. @csrf

Direktiva @csrf generiše CSRF (Cross-Site Request Forgery) token unutar forme, štiteći aplikaciju od napada putem lažnih zahteva.

6. @lang

Direktiva @lang omogućava lako prevođenje tekstova unutar *Blade* šablona koristeći Laravel-ove lokalizacijske fajlove i podržavajući internacionalizaciju aplikacije.

7. @isset, @empty

Direktive @isset i @empty proveravaju da li je promenljiva definisana (nije null) ili prazna, omogućavajući kondicionalno prikazivanje sadržaja na osnovu ovih uslova.

8. @forelse

Direktiva @forelse je slična direktivi @foreach, ali sa dodatnom opcijom za prikazivanje alternativnog sadržaja ukoliko je niz prazan.

9. @can, @cannot

Ove direktive proveravaju da li trenutni korisnik ima određene dozvole ili uloge, omogućavajući prikaz ili sakrivanje delova UI-a na osnovu korisničkih dozvola.

10. @verbatim

Direktiva @verbatim omogućava da se unutar njenog bloka sadržaj ne interpretira kao *Blade* direktiva, već se prikazuje direktno, što je korisno za prikazivanje *Blade* sintakse ili kada se koriste *frontend* okviri koji koriste sličnu sintaksu.

11. @inject

Direktiva @inject se koristi za ubacivanje instanci Laravel servisa direktno u *Blade* šablone, omogućavajući lak pristup funkcionalnostima servisa.

12. @once

Direktiva @once ograničava izvršavanje određenog dela koda unutar šablona na samo jedno izvršavanje, čak i ako se šablon uključuje više puta.

13. @json

Direktiva @json konvertuje PHP niz ili objekat u JSON format i prikazuje ga u HTML kodu.

14. @error, @enderror

Direktive @error i @enderror su specijalizovane za prikazivanje grešaka validacije formi u Laravel aplikacijama. Kada se koristi Laravelova validacija zahteva, sve greške validacije se automatski šalju nazad u prikaz, gde se mogu prikazati koristeći ove direktive.

3.5 Rasporedi koji koriste nasleđivanje šablona

Rasporedi koji koriste nasleđivanje šablona su ključni koncept u razvoju veb aplikacija, posebno kada je reč o modernim MVC (Model-View-Controller) okvirima kao što je Laravel. Ova tehnika omogućava razvojnim programerima da kreiraju ponovno upotrebljive šablone (*master layout*) koji definišu osnovnu strukturu veb stranice, kao što su zaglavlje, podnožje, navigacija i slično. Zatim, specifični delovi sadržaja mogu biti unešeni u ovaj *layout* korišćenjem mehanizma nasleđivanja šablona. Ovaj pristup ne samo da smanjuje redundantnost i olakšava održavanje koda, već i poboljšava konzistentnost i efikasnost razvojnog procesa.

Prednosti nasleđivanja šablona

- **Konzistentnost UI-a**: *Master layout* osigurava da se zajednički elementi korisničkog interfejsa kao što su zaglavlje, podnožje i navigacija održavaju konzistentnim kroz sve stranice aplikacije.
- **Smanjenje ponavljanja koda**: Eliminiše potrebu za ponavljanjem istog HTML koda na više stranica, što olakšava održavanje i ažuriranje aplikacije.
- **Fleksibilnos**t: Omogućava lako dodavanje, uklanjanje ili izmenu elemenata korisničkog interfejsa na centralizovan način, bez potrebe za modifikacijom svake pojedinačne stranice.
- **Efikasnost razvoja**: Smanjuje vreme potrebno za razvoj i testiranje, jer se promene mogu brzo implementirati na sve stranice koje nasleđuju *master layout*.

Implementacija u Laravel-u

Laravel koristi tzv. *Blade templating engine* za implementaciju nasleđivanja šablona. Način na koji se postiže nasleđivanje šablona je sledeći:

1. **Definisanje master layouta**: Potrebno je da se kreira *Blade* fajl koji služi kao *master layout*. Unutar ovog fajla, *{{ $slot }}* se koristi za definisanje sekcija koje će biti popunjene sadržajem specifičnim za pojedinačne stranice:

```
<!-- resources/views/layouts/app.blade.php -->
<html>
<head>
  <title>App Name - @yield('title')</title>
</head>
<body>
  @include('header')
  {{ $slot }}
  @include('footer')
</body>
</html>
```

2. **Nasleđivanje master layout-a**: Kada se koristi stranica, koristi se *<x-app-layout> </x-app-layout>* kako bi se nasledio *master layout*, a @section direktive se koriste kako bi popunili prethodno definisane sekcije sa sadržajem:

```
<!-- resources/views/welcome.blade.php -->
@extends('layouts.app')

@section('title', 'Welcome')

@section('content')
<p>This is my body content.</p>
@endsection
```

3.6 Pokretanje baza podataka koristeći *Docker*

Docker je moćna platforma otvorenog koda koja omogućava razvoj, otpremanje i izvršavanje aplikacija u izolovanim okruženjima koji se zovu kontejneri. Ovi kontejneri omogućavaju softveru da radi pouzdano i konzistentno na različitim razvojnim i produkcijskim okruženjima. *Docker* koristi resurse operativnog sistema host-a, što ga čini jednostavnijim i bržim u poređenju sa tradicionalnim virtuelnim mašinama koje zahtevaju sopstveni operativni sistem.

Gde *Docker* može pomoći u projektu?

Korišćenje *Docker*-a u projektu donosi brojne prednosti, naročito kada je reč o konfiguraciji i upravljanju zavisnostima kao što su baze podataka i alati za administraciju. Nadalje, u navedenom primeru projekta *docker-compose.yml* fajl će predstaviti način kako *Docker* može

olakšati rad sa MySQL bazom podataka i *Adminer*-om, alatom za upravljanje bazama podataka putem veb interfejsa. *Docker* može da obezbedi:

- **Jednostavnost konfiguracije**: *Docker* omogućava definisanje i distribuciju okruženja baze podataka kroz jednostavan *docker-compose.yml* fajl, što znači da svako ko radi na projektu može lako pokrenuti isto okruženje bez obzira na operativni sistem ili druge lokalne konfiguracije.

- **Izolaciju**: Svaka komponenta sistema, kao što su aplikacija, baza podataka i alati za administraciju, mogu biti upakovani u zasebne kontejnere. To znači da promene u jednom servisu neće uticati na druge, smanjujući tako rizik od konflikta zavisnosti.

- **Ponovljivost**: Korišćenjem *Docker*-a, može se osigurati da aplikacija radi isto u svakom okruženju, od razvoja do produkcije, smanjujući probleme tipa "radi na mom računaru".

- **Lakoću ažuriranja i skalabilnost**: Ažuriranje verzije baze podataka ili promena konfiguracije može se lako postići promenom u *docker-compose.yml* fajlu, bez potrebe za manuelnim instalacijama ili konfiguracijama. Takođe, *Docker* omogućava lako skaliranje servisa.

Docker-compose.yml fajl ima sledeći sadržaj:

```yaml
version: "3.9"
services:
 mysql:
   image: mariadb:10.8.3
   command: --default-authentication-plugin=mysql_native_password
   restart: always
   environment:
     MYSQL_ROOT_PASSWORD: root
   ports:
     - 3306:3306
 adminer:
   image: adminer
   restart: always
   ports:
     - 8080:8080
```

Ovaj kod predstavlja konfiguracioni fajl za *Docker Compose*, koji definiše skup servisa i kako će oni biti pokrenuti u *Docker* kontejnerima. *Docker Compose* omogućava upravljanje multi-kontejnerskim aplikacijama definisanjem servisa i mreža u *.yml* formatu.

MySQL servis koristi *Docker* sliku *mariadb:10.8.3* za kreiranje kontejnera, koja je standardna za MariaDB baze i verziju koja je kompatibilna sa MySQL-om. Navedene su konfiguracije kao što su *restart* politika, lozinka za *root* korisnika i portovi.

Deo konfiguracije *ports* unutar *docker-compose.yml* fajla definiše mapiranje portova između kontejnera i host mašine. Specifično, sintaksa - 3306:3306 označava mapiranje portova, gde je format *host_port:container_port*. Prvi broj (pre dvotačke) predstavlja port (*host_port*) na host mašini, odnosno mašini na kojoj se *Docker* izvršava. Ovo je port preko kojeg se može pristupiti servisu izvan *Docker* kontejnera, na primer preko veb pretraživača ili aplikacije. Drugi broj (posle dvotačke) označava port unutar *Docker* kontejnera (*container_port*) koji se mapira na port hosta. Ovo je port koji servis unutar kontejnera koristi za komunikaciju.

U navedenom primeru, 3306:3306 znači da se port 3306 unutar *Docker* kontejnera (koji koristi MariaDB) mapira direktno na port 3306 na host mašini. Ovo omogućava da se sa host mašine može pristupiti bazi podataka koja se izvršava unutar *Docker* kontejnera koristeći standardni port 3306, kao da se baza direktno izvršava na host mašini.

Ovo mapiranje portova je ključno za omogućavanje komunikacije između aplikacije i servisa koji se izvršavaju unutar *Docker* kontejnera, kao i za pristup tim servisima sa eksternih uređaja ili aplikacija koje nisu deo *Docker* okruženja.

Adminer servis koristi standardnu *adminer* sliku, omogućavajući pristup i upravljanje MySQL bazom podataka kroz veb interfejs. Takođe, konfigurisan je i port i politika restartovanja.

Korišćenjem ovakvog *docker-compose* fajla, može se sa jednom komandom postaviti kompletno okruženje za razvoj koje uključuje bazu podataka (MariaDB) i interfejs za njeno upravljanje (*adminer*). *Docker* u ovom kontekstu, igra ključnu ulogu u pojednostavljenju razvojnog ciklusa i olakšava upravljanje infrastrukturom projekta.

U okviru navedene aplikacije, fajl *docker-compose.yml* smešten je u korenom (*root*) direktorijumu projekta. Da bi se pokrenula konfiguracija definisana u ovom fajlu, otvoriće se terminal direktno u *root*-u projekta. Sledeći korak je izvršavanje komande: **`docker compose`**

`up -d`, koja će inicirati i pokrenuti definisane kontejnere prema konfiguraciji navedenoj u *docker-compose.yml* fajlu.

Komanda **`docker compose up`** služi za kreiranje i pokretanje kontejnera. Dodavanje parametra koji se koristi u komandnoj liniji da bi se odredilo kako treba izvršiti određenu komandu (*flag*) *-d* (skraćenica za *–detach*), instruiše *Docker* da pokrene kontejnere u pozadinskom režimu. Ovo znači da će se kontejneri izvršavati nezavisno od terminala, omogućavajući da se nastavi sa radom u terminalu bez posebnih prekidanja logovima i informacijama o stanju kontejnera. Pokretanje kontejnera u pozadinskom režimu je često preferirana opcija prilikom razvoja, jer omogućava da servisi, poput baze podataka i administrativnih alata poput *Adminer*-a, budu stalno dostupni bez zauzimanja terminala. Takođe, olakšava se rad sa aplikacijom i omogućava se lako praćenje logova kroz posebne komande kada je to potrebno, umesto da se kontinuirano prati izlaz u terminalu.

3.7 Konekcija sa bazom, kreiranje modela, migracija, fabrika modela i sejalice

Konekcija sa bazom

Za uspostavljanje konekcije sa bazom podataka unutar Laravel aplikacije, ključna uloga pripada *.env* fajlu, koji služi kao centralno mesto za definisanje varijabli okruženja. Pošto se radi sa *Docker*-om u lokalom okruženju i koristi se konfiguracija koja obuhvata DBMS MariaDB, kako je određeno u *docker-compose.yml* fajlu, ključno je pažljivo konfigurisati *.env* fajl.

U slučaju navedenog projekta, *docker-compose.yml* fajl konfiguriše MariaDB bazu sa osnovnim konfiguracijama, uključujući lozinku *root* korisnika i mapiranje portova. Ovo direktno utiče na to kako će se konfigurisati *.env* fajl u Laravel aplikaciji. Pretpostavljajući da je ime baze podataka koju želimo da koristimo *laravel-10-task-list*, sledi način kako bi trebalo ažurirati *.env* fajl:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel-10-task-list
DB_USERNAME=root
DB_PASSWORD=root
```

Ove promene omogućavaju Laravel aplikaciji da uspostavi konekciju sa bazom podataka koja se pokreće unutar *Docker* kontejnera, koristeći MariaDB. Sledi detaljno objašnjenje kako svaka od ovih varijabli utiče na konekciju:

- **DB_CONNECTION**: Definiše da se koristi MySQL drajver za konekciju, budući da je MariaDB kompatibilna i može se koristiti sa MySQL drajverom.
- **DB_HOST**: Postavljen je na 127.0.0.1, ukazujući na činjenicu da se baza podataka nalazi na istoj mašini gde se i aplikacija izvršava, što je slučaj i u našem primeru jer se koristi *Docker* na lokalnom razvojnom okruženju.
- **DB_PORT**: Port 3306 je standardni MySQL/MariaDB port koji je takođe mapiran unutar *docker-compose.yml*, omogućavajući komunikaciju između Laravel-a i baze podataka.
- **DB_DATABASE**: Ime baze podataka *laravel-10-task-list* je ime koje je odabrano.
- **DB_USERNAME i DB_PASSWORD**: Za autentifikaciju se koristi *root* korisnik sa lozinkom *root,* kako je definisano u *docker-compose.yml* fajlu.

Konfigurisanjem *.env* fajla na ovaj način, osigurava se da Laravel aplikacija može pouzdano da se poveže i komunicira sa MariaDB bazom podataka koja se izvršava unutar *Docker* kontejnera, čime se olakšava razvoj i testiranje aplikacije u izolovanom okruženju.

Važno je naglasiti da *.env* fajl sadrži osetljive informacije koje se tiču konfiguracije okruženja, uključujući pristupne podatke za bazu podataka, API ključeve i druge osetljive podatke. Iz tog razloga, veoma je bitno da *.env* fajl nikada ne bude deo repozitorijuma koji se deli ili objavljuje, posebno u sistemima za kontrolu verzija poput Git-a. Ovo je standardna praksa u razvoju softvera kako bi se zaštitile osetljive informacije od neovlašćenog pristupa.

Da bi se osiguralo da se *.env* fajl nikada ne pošalje (*push*) na Git, trebalo bi ga eksplicitno isključiti dodavanjem u *.gitignore* fajl projekta. Fajl *.gitignore* Git-u govori koje fajlove ili direktorijume treba ignorisati prilikom čuvanja stanja projekta ili ažuriranja projekta na repozitorijumu. Fajl *.gitignore* se nalazi u osnovnom direktorijumu projekta.

Ovim se osigurava da će *.env* fajl biti ignorisan od strane Git-a, štiteći sve konfiguracijske podatke. Takođe, dobra praksa je da se pored *.env* fajla održava i *.env.example* fajl, koji sadrži

iste ključeve kao i *.env* fajl, ali bez osetljivih informacija. Ovo omogućava drugim programerima da lako postave svoje lokalno okruženje bez izlaganja tajnih podataka. Fajl *.env.example* treba da bude uključen u Git repozitorijum.

Uključivanjem *.env.example* fajla i isključivanjem *.env* fajla iz Git-a, osigurava se da aplikacija ostane sigurna i da je moguće lako podesiti za novi razvojni ambijent, dok se istovremeno štite osetljive informacije od neovlašćenog pristupa.

Kreiranje modela i migracije

Modeli su ključni deo Laravel-ovog *Eloquent ORM*-a (Object-Relational Mapping), koji omogućava jednostavnu interakciju sa bazom podataka kroz objektno orijentisan pristup. Svaki model u Laravel-u obično predstavlja tabelu u bazi podataka i svaka instanca modela odgovara redu u toj tabeli. Modeli omogućavaju definisanje veza između različitih tabela, kao što su *hasOne*, *hasMany*, *belongsTo* i *belongsToMany*. Kroz ove veze, može se lako pristupati povezanim podacima i mogu se izvršavati kompleksni upiti bez potrebe za pisanjem SQL koda. Modeli takođe omogućavaju definisanje atributa, pravila validacije, prilagođenih upita i mnogo više, čineći rad sa podacima efikasnim i organizovanim.

Aplikacija koja je data kao primer ima dva modela:

1. Task
2. User

Migracije služe kao sistem za kontrolu verzija baze podataka, pružajući programerima jednostavan način za menjanje i deljenje strukture baze podataka. Laravel migracije su definisane kao PHP klase i svaka migracija sadrži metode *up()* i *down()*. Metoda *up()* se koristi za dodavanje novih tabela, kolona ili indeksa u bazu podataka, dok se metoda *down()* koristi za brisanje istih. Ovo omogućava jednostavno povraćaj (*rollback*) promena ukoliko je to potrebno. Migracije se kreiraju pomoću *Artisan CLI* (Command-Line Interface) alata i mogu se pokretati pojedinačno ili grupno. Korišćenjem migracija, može se osigurati da je struktura baze podataka dosledna kroz različita okruženja i među članovima tima, što je posebno korisno u timskom radu i pri razvoju aplikacija koje zahtevaju česte promene u strukturi baze.

Zajedno, modeli i migracije predstavljaju snažan set alata u Laravel-u, koji omogućava efikasno upravljanje podacima i strukturom baze podataka. Korišćenjem ovih funkcionalnosti,

mogu se brzo razvijati aplikacije sa složenom logikom i podacima, uz održavanje čistog i organizovanog koda.

U primeru aplikacije, modeli i migracije su generisani koristeći *php artisan* komandu. Specifično, za kreiranje modela *Task* zajedno sa njegovom migracijom, upotrebljena je komanda **`php artisan make:model Task -m`**. Ovaj postupak automatski generiše *Task* model i stvara migracijski fajl za kreiranje *task* tabele u bazi podataka. Ime tabele automatski se određuje u množini, bazirano na imenu modela. Struktura migracije za tabelu *tasks* predstavljena je na slikama 1 i 2.

```php
<?php

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

return new class extends Migration
```

Slika 1. Kod koji predstavlja primer migracije, koja se koristi za definisanje i upravljanje strukturom baze podataka
Izvor: Autorska obrada

```php
    * Run the migrations.
    */
    public function up(): void
    {
        Schema::create('tasks', function (Blueprint $table) {
            $table->id();
            $table->string('name')->nullable(false);
            $table->text('description');
            $table->text('long_description')->nullable(true);
            $table->boolean('is_completed')->default(false);
            $table->timestamps();
        });
    }

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

Slika 2. Struktura migracije za tabelu tasks
Izvor: Autorska obrada

Kao što se može videti, tabela *tasks* sadrži kolone: id, name, description, long_description, is_completed, uz automatsko kreiranje kolona created_at i updated_at kroz upotrebu timestamps() metode.

Preostaje još da se generiše model i migracija za *user*-a. To će se postići primenom komande **`php artisan make:model User -m`**.

Fabrika modela i sejalice

U Laravel-u, fabrike (factory) i sejalice (seeder) su alati koji olakšavaju testiranje i inicijalno popunjavanje baze podataka testnim podacima. Oba alata su ključna za efikasan razvoj i testiranje aplikacija, omogućavajući brzo generisanje velike količine realnih podataka.

Fabrike u Laravel-u se koriste za definisanje načina kako generisati modele i njihove atribute. Ovo je posebno korisno kada su potrebni modeli sa validnim podacima za testiranje ili razvoj. Fajlovi fabrika su definisani pomoću *Faker* biblioteke koja generiše realne podatke poput imena, adresa i telefonskih brojeva. Na primer, ukoliko koristimo model *User*, može se kreirati fabrika koja će generisati korisnike sa slučajno odabranim imenima, email adresama i slično.

U primeru aplikacije kreirana je fabrika *TaskFactory.php*, gde se detaljnije kod može videti na slici 3.

```
<?php
```

```php
namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;

/**
* @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Task>
*/

class TaskFactory extends Factory
{
  /**
   * Define the model's default state.
   *
   * @return array<string, mixed>
   */
  public function definition(): array
  {
     return [
        'name' => fake()->sentence(),
        'description' => fake()->paragraph(),
        'long_description' => fake()->paragraph(7, true),
        'is_completed' => fake()->boolean(),
     ];
  }}
```

Slika 3. Kreiranje klase fabrike koja se koristi za generisanje podataka za model Task
Izvor: Autorska obrada

Važni delovi koda, koji su značajni za generisanje podataka su sledeći:

- **'name' => fake()->sentence()** - Ova linija generiše nasumičnu rečenicu koja će biti korišćena kao vrednost za *name* atribut. Laravel-ova *fake()* funkcija omogućava pristup različitim metodama *Faker* biblioteke za generisanje testnih podataka. U ovom slučaju, *sentence()* metoda generiše standardnu rečenicu.
- **'description' => fake()->paragraph()** – Pomoću ovog se generiše nasumičan pasus za *description* atribut. Metoda *paragraph()* podrazumevano vraća jedan pasus, čineći ovu vrednost idealnom za opisivanje ili kratke napomene.
- **'long_description' => fake()->paragraph(7, true)** - Za *long_description* atribut generiše se nešto detaljniji tekst, sa 7 rečenica. Opcioni drugi argument *true* određuje da se sve rečenice vraćaju kao jedan string, umesto da se vraćaju u formi niza.
- **'is_completed' => fake()->boolean()** - Konačno, za *is_completed* atribut, generiše se nasumična Boolean vrednost (true ili false). Ovo je korisno za statusne oznake, poput označavanja da li je zadatak završen ili ne.

Sejalice se koriste za popunjavanje baze podataka definisanim setom podataka. Nakon što se definišu modeli i migracije, može se koristiti sejalica kako bi se uneli inicijalni podaci u bazu.

Ovo je korisno ne samo za testiranje, već i za popunjavanje baze podataka potrebnim podacima prilikom inicijalnog pokretanja aplikacije. Laravel sejalice su inače PHP skripte koje koriste *Eloquent* model, koji omogućava rad sa bazama podataka koristeći objekte umesto SQL upita, čime se olakšava manipulacija podacima i interakcija sa bazom podataka. Takođe sejalice koriste i *DB facade* komponentu za unos podataka, a koja pruža statičke metode koje omogućavaju izvršavanje raznih SQL upita bez potrebe za pisanjem složenih SQL kodova.

U primeru aplikacije, unutar klase *DatabaseSeeder.php*, dodata je linija `Task::factory(20)->create();` u metodi *run*. Ovim postupkom, kada se pokrene komanda `php artisan db:seed`, automatski će se generisati 20 zapisa u tabeli *tasks*. Generisanje podataka za ove zapise biće izvršeno prema pravilima definisanim u *TaskFactory* klasi.

3.8 Čitanje podataka iz baze

U kontekstu Laravel aplikacija, čitanje podataka iz baze je proces koji se obavlja kroz *Eloquent ORM* (Object-Relational Mapping) ili *Query Builder*, omogućavajući razvojnim programerima da na intuitivan i efikasan način manipulišu podacima. Laravel, kao vodeći PHP okvir, pruža robustan set alata za rad sa bazom podataka, čime se znatno pojednostavljuje interakcija sa podacima i unapređuje produktivnost razvoja.

Eloquent ORM

Eloquent ORM predstavlja Laravel-ovu implementaciju aktivnog zapisa (*Active Record*) obrasca, omogućavajući svakom modelu da odgovara tabeli u bazi podataka, a instanci modela – redu u tabeli. Ovo značajno olakšava čitanje podataka, jer modeli intuitivno rade sa svojim podacima, kao i sa podacima povezanih modela.

Primer čitanja podataka koristeći *Eloquent* u primeru aplikacije je sledeći:

Za model *Task*, koji odgovara tabeli *tasks* u bazi, čitanje svih zadataka bi se moglo izvesti kroz sledeći jednostavan poziv: `$tasks = App\Models\Task::all();`.

Ovaj kod vraća kolekciju svih zadataka iz baze, kroz koju se može iterirati i prikazati u aplikaciji. Za filtriranje podataka, *Eloquent* nudi različite metode poput *find*, *where*, *orderBy* i druge, koje omogućavaju precizno i fleksibilno pretraživanje podataka.

Na primer, da bi se prikazao zadatak koji ima identifikacioni broj 10, može se upotrebiti sledeći kod: `$taskId10 = App\Models\Task::findOrFail(10);`.

Query Builder

Za razvojne programere koji preferiraju direktan rad sa SQL upitima ili su im potrebne složenije manipulacije podacima, Laravel nudi opcije korišćenja *Query Builder*-a. Ovaj alat omogućava pisanje čistih i bezbednih SQL upita koristeći PHP sintaksu, bez potrebe za manuelnim pisanjem SQL koda.

Primer kako se čitaju podaci koristeći *Query Builder* je sledeći:

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

Ovaj poziv vraća sve zadatke iz *tasks* tabele, vraćajući kolekciju koja se može koristiti unutar aplikacije. *Query Builder* podržava širok spektar SQL operacija, uključujući selekciju (select), filtriranje (where), sortiranje (orderBy), grupisanje (groupBy) i mnoge druge, omogućavajući kompleksne upite i manipulaciju podacima. Ukoliko je na primer, potrebno prikazati zadatak sa identifikacionim brojem 10, to se može postići koristeći sledeći izraz: `$taskId10 = DB::table('tasks')->where('id', 10)->get();`.

3.9 Forme i CSRF zaštita

U svetu veb razvoja, sigurnost forme predstavlja ključni aspekt zaštite veb aplikacija od različitih vrsta napada. Laravel okvir bavi se sigurnošću na ozbiljan način i uključuje ugrađene mehanizme za zaštitu od uobičajenih sigurnosnih pretnji. Jedan od najvažnijih mehanizama zaštite je CSRF **(***Cross-Site Request Forgery***)** zaštita.

CSRF zaštita

CSRF napadi omogućavaju napadaču da izvrši neželjene akcije u ime korisnika bez njegovog znanja ili saglasnosti, koristeći autentifikaciju preko koje je korisnik već povezan sa sajtom. Da bi se ovo sprečilo, Laravel koristi CSRF token, koji je jedinstveni, nasumično generisan string vezan za korisničku sesiju. CSRF token osigurava da je zahtev poslat iz legitimnog izvora - odnosno iz same aplikacije, a ne sa nekog spoljnog ili zlonamernog sajta.

Primena CSRF zaštite u Laravel-u

Laravel automatski generiše i proverava CSRF token za svaki POST, PUT, PATCH, i DELETE zahtev kroz *middleware*, osiguravajući da su svi takvi zahtevi legitimni. Da bi se dodao CSRF token u formu, Laravel obezbeđuje *@csrf Blade* direktivu koja se dodaje unutar HTML forme:

```
<form action="{{ route('tasks.destroy', $task) }}" method="POST" class="grow">
  @csrf
  @method('DELETE')
  <button type="submit" class="w-full bg-red-950/80 text-white font-bold shadow-sm
sm:rounded-lg p-2">
    Delete
  </button>
</form>
```

Slika 4. Prikaz koda koji predstavlja HTML formu koja se koristi za brisanje zadatka u Laravel aplikaciji
Izvor: Autorska obrada

Kada se forma pošalje, Laravel-ov *middleware VerifyCsrfToken* proverava da li je poslati CSRF token validan i odgovara onom koji je sačuvan u sesiji korisnika. Ukoliko token nije validan ili nedostaje, Laravel odbacuje zahtev, sprečavajući potencijalni CSRF napad.

Na stranici primera sajta koja se nalazi na adresi: *localhost:8000/tasks/1,* postoji skriveno polje unutar forme koje izgleda na sledeći način: **`<input type="hidden" name="_token" value="XjV6FejU2TG3fTe6RFUfeGht6TuxU4vBiuRVGmyE" autocomplete="off">`**. Ovo polje sadrži bezbednosni token u svom *value* atributu, dok *name* atribut jasno ukazuje da je reč o CSRF tokenu.

Direktiva **`@method('DELETE')`** u Laravel *Blade* šablonima služi kao način da se precizira HTTP metoda koju treba koristiti za obradu forme, što je u ovom slučaju metoda DELETE. HTML forme prirodno podržavaju samo GET i POST metode. Međutim, moderni veb razvoj i RESTful konvencije[2] zahtevaju upotrebu dodatnih metoda kao što su PUT, PATCH i DELETE za izvršavanje specifičnih akcija nad resursima.

Kada se kreira forma koja treba da izvrši DELETE zahtev, kao što je brisanje resursa u Laravel aplikaciji, direktno korišćenje DELETE metoda nije moguće kroz standardni metod atribut

[2] Skup pravila i smernica koje se koriste za dizajniranje i implementaciju REST (Representational State Transfer) API-ja. Ove konvencije omogućavaju standardizovani način interakcije sa resursima preko HTTP protokola.

forme. Umesto toga, Laravel omogućava korišćenje *@method Blade* direktive da se odredi prava HTTP metoda. Većina pretraživača podržava samo dva osnovna metoda: GET i POST.

Kada se forma sa ovom direktivom obrađuje, Laravel generiše skriveno polje (_method) unutar forme, koje sadrži vrednost 'DELETE'. U primeru aplikacije, na stranici dostupnoj na adresi *localhost:8000/tasks/1*, nalazi se forma koja uključuje skriveno polje definisano kao **`<input type="hidden" name="_method" value="DELETE">`**. Ovo polje automatski generiše Laravel kada se koristi direktiva **`@method('DELETE')`**, omogućavajući da forma simulira slanje DELETE zahteva. Laravel-ov *middleware* zatim proverava ovo skriveno polje prilikom obrade zahteva i tretira zahtev kao DELETE zahtev, čak iako je originalno poslat kao POST. Ovo omogućava razvojnim programerima da prate RESTful konvencije bez ograničenja HTML forme. Ova metoda ne samo da olakšava implementaciju RESTful akcija u Laravel aplikacijama, već i pomaže u očuvanju jasne strukture koda i podržava dobre prakse u dizajnu veb aplikacija.

Prednosti CSRF zaštite

Implementacija CSRF zaštite u Laravelu pruža sledeće prednosti:

- **Sigurnost**: Smanjuje rizik od CSRF napada koji mogu kompromitovati aplikaciju i korisničke podatke.
- **Jednostavnost**: Korišćenjem @csrf direktive, lako je dodati CSRF zaštitu bez potrebe za manuelnim generisanjem ili proverom tokena.
- **Automatizacija**: Integracijom CSRF zaštite u *middleware*, Laravel automatski upravlja tokenima, čineći aplikaciju sigurnijom bez dodatnog truda razvojnog tima.

CSRF protekcija je samo jedan deo Laravelovog pristupa sigurnosti veb aplikacija. Integracijom ovakvih mehanizama, Laravel omogućava razvoj sigurnih i zaštićenih aplikacija, štiteći i aplikaciju i korisnike od različitih vrsta napada.

3.10 Validacija i sortiranje podataka

Validacija i sortiranje podataka predstavljaju dve ključne komponente u razvoju veb aplikacija korišćenjem Laravel okvira, omogućavajući razvojnim programerima da efikasno upravljaju korisničkim unosima i prikazuju podatke na organizovan način.

Validacija podataka

Validacija podataka u Laravel-u je proces koji osigurava da korisnički unosi zadovoljavaju određene kriterijume pre nego što se obrade ili sačuvaju u bazi podataka. Laravel pruža bogat set funkcionalnosti za validaciju, koji se može lako koristiti kako u kontrolerima, tako i u *form request* klasama, čime se postiže bolja organizovanost koda i smanjuje njegova redundantnost.

U Laravel-u, validacija može biti implementirana i direktno unutar rutnih definicija u *web.php* fajlu, koristeći *Closure* (anonimne funkcije). Ova metoda omogućava brzu i jednostavnu validaciju podataka za manje aplikacije gde kreiranje zasebnih kontrolera ili *form request* klasa može biti zahtevno. U primeru aplikacije, specifično je razvijena klasa *TaskRequest.php* koja se koristi za validaciju podataka u vezi sa zadacima. Ova klasa nasleđuje *FormRequest*, koji je deo Laravel okvira i pruža strukturiran način za upravljanje validacijom i autorizacijom formi.

```php
<?php
namespace App\Http\Requests;
use Illuminate\Foundation\Http\FormRequest;
class TaskRequest extends FormRequest
{
  public function authorize(): bool
  {
    return true;
  }
  public function rules(): array
  {
    return [
      'name' => 'required|max:255',
      'description' => 'required',
      'long_description' => 'required'
    ];
  }}
```

Slika 5. Prikaz koda koji predstavlja Laravel FormRequest klasu koja se koristi za validaciju podataka unetih kroz HTTP zahteve

Izvor: Autorska obrada

Struktura TaskRequest klase

Klasa *TaskRequest* je dizajnirana sa ciljem da olakša i centralizuje proces validacije formi, omogućavajući jasno definisanje pravila validacije za podatke koje korisnik šalje. Ovo se postiže kroz dve glavne metode:

- Metoda *authorize* određuje da li korisnik ima dozvolu da izvrši ovaj zahtev. U navedenom slučaju, ova metoda jednostavno vraća *true*, implicirajući da svaki korisnik može poslati ovaj zahtev. U složenijim aplikacijama, ovde bi mogla da se implementira logika za proveru korisnikovih prava i dozvola.
- Metoda *rules* vraća asocijativni niz pravila validacije, gde ključevi predstavljaju imena polja forme, a vrednosti su pravila koja se primenjuju na te podatke. U navedenom primeru, definisana su pravila za tri polja: name, description i long_description. Svako od ovih polja zahteva da bude popunjeno (required), dok *name* polje takođe ne sme premašiti 255 karaktera (max:255).

Sortiranje podataka

Sortiranje podataka je još jedan važan aspekt razvoja aplikacija, omogućavajući korisnicima da pregledaju podatke u logičkom redosledu. Laravel nudi jednostavne načine za sortiranje podataka koristeći *Eloquent ORM* ili *Query Builder*, čime se omogućava lako upravljanje redosledom prikaza podataka.

Sortiranje pomoću Eloquent-a:

Eloquent omogućava sortiranje rezultata upita korišćenjem metoda kao što je *orderBy*:

```
$tasks = App\Models\Task::orderBy('id'', 'desc')->get();
```

Sortiranje pomoću Query Builder-a:

Slično kao i *Eloquent*, Laravel-ov *Query Builder* takođe nudi metode za sortiranje, omogućavajući fino podešavanje upita direktno na nivou baze podataka:

```
$tasks = DB::table('tasks')->orderBy('id', 'desc')->get();
```

Validacija i sortiranje podataka su ključni za razvoj intuitivnih i sigurnih veb aplikacija. Laravel pruža set alata koji olakšavaju implementaciju ovih funkcionalnosti, čineći razvojni proces bržim i efikasnijim, dok istovremeno povećava sigurnost i korisničko iskustvo prilikom korišćenja aplikacije.

3.11 Sesije, greške i pop-up poruke

Sesije, greške i pop-up poruke su ključni elementi interakcije između korisnika i veb aplikacije, a koje imaju vitalnu ulogu u poboljšanju korisničkog iskustva i sigurnosti. Laravel, kao moderni PHP okvir, pruža bogat set funkcionalnosti za lako upravljanje ovim aspektima.

Sesije

Sesije u Laravel-u omogućavaju privremeno čuvanje informacija o korisniku tokom njegove interakcije sa aplikacijom. Ovo je ključno za funkcionalnosti poput autentifikacije korisnika, gde je potrebno sačuvati status prijave korisnika dok se kreće kroz različite delove aplikacije. Laravel olakšava rad sa sesijama kroz intuitivni API, omogućavajući čuvanje, pristup i brisanje podataka sesije sa lakoćom.

Laravel nudi različite drajvere za upravljanje sesijama i svaki ima svoje specifičnosti, prednosti i najbolje scenarije upotrebe. Izbor drajvera za sesije može imati značajan uticaj na performanse i skalabilnost veb aplikacije. U primeru aplikacije koristi se *fajl drajver* za sesije, što je sasvim odgovarajuće za manje aplikacije. Međutim, za aplikacije većeg obima, razmatranje alternativa kao što su *Redis* ili *Memcached* može doneti značajne benefite.

Korišćenje *fajl drajvera* znači da se sesijski podaci čuvaju u fajlovima unutar servera. Ovo je jednostavan i direktan način za čuvanje sesija, koji ne zahteva dodatnu infrastrukturu ili konfiguraciju. Iako je ovo praktično za razvoj i manje aplikacije, korišćenje fajl sistema može postati usko grlo u terminima performansi kada aplikacija raste.

Redis i *Memcached* su napredni sistemi za upravljanje keširanjem u memoriji, koji nude brže performanse u odnosu na tradicionalno čuvanje sesija u fajlovima. Oba sistema rade tako što privremeno čuvaju podatke u RAM-u, što omogućava aplikaciji brži pristup i obradu sesijskih podataka.

Situacije kada je odgovarajuće koristiti *Redis* ili *Memcached* su sledeće:

- **Veće aplikacije**: Kada se aplikacija razvija i počinje da prima veći broj zahteva, brzina pristupa sesijskim podacima postaje kritična. *Redis* i *Memcached* mogu značajno poboljšati performanse sesija tako što smanjuju vreme potrebno za čitanje i pisanje sesijskih podataka.
- **Distribuirane aplikacije**: Za aplikacije koje se izvršavaju na više servera (u klasteru), potrebno je deljenje podataka sesije između instanci aplikacije. *Redis* i *Memcached* omogućavaju lako deljenje sesija među serverima, što je neophodno za održavanje kontinuiteta sesije kroz distribuiranu infrastrukturu.
- **Potreba za brzinom i skalabilnošću**: Ako aplikacija zahteva brzu obradu velikog broja sesija ili ako je u planu da se skalira aplikacija, prelazak na *Redis* ili *Memcached* može pružiti potrebnu infrastrukturu za podršku ovim zahtevima.

Greške

Upravljanje greškama je još jedan ključan segment razvoja aplikacija. Laravel pruža mehanizme za elegantno rukovanje greškama i izuzecima, automatski pretvarajući izuzetke u korisničke greške i prikazujući ih na odgovarajući način. Razvojni programeri mogu definisati prilagođene stranice za različite tipove grešaka (npr., 404 stranica nije pronađena), čime se poboljšava korisničko iskustvo i pružaju korisne informacije kada dođe do grešaka.

Pop-up poruke

Pop-up poruke, često nazivane i *flash* poruke, koriste se za prikazivanje privremenih poruka korisnicima. One su posebno korisne za prikazivanje povratnih informacija nakon što korisnik izvrši neku akciju, kao što je uspešno slanje forme ili ažuriranje profila. Laravel olakšava implementaciju *flash* poruka kroz sesije, omogućavajući da se jednostavno postave poruke koje će biti prikazane korisniku samo jedanput, pri sledećem zahtevu.

U primeru aplikacije prikazaće se primena navedenog. Naime, unutar glavnog *layout* fajla *layouts/app.blade.php*, dodat je kod koji proverava da li postoji poruka sesije pod ključem *'success'*. Ako postoji, prikazuje se info blok sa porukom uspeha. Ovo je odličan način da se korisnicima pruže povratne informacije o uspešno izvršenim akcijama, poput brisanja zadatka.

```
@if(session()->has('success'))
  <div class="bg-green-100 border border-green-400 text-green-700 px-4 py-3 rounded
relative" role="alert">
    <span class="block sm:inline">{{ session()->get('success') }}</span>
```

```
  </div>
@endif
```

Slika 6. Primer prikaza flash poruke u Laravel-u koristeći Blade template engine
Izvor: Autorska obrada

Ovaj segment koda koristi Laravel-ov *session() helper* kako bi proverio da li postoji poruka pod ključem *'success'*. Ako takva poruka postoji, ona se prikazuje unutar definisanog elementa. Stilovi su primenjeni da bi se poruka istakla kao pozitivan *feedback*, koristeći *Tailwind CSS* klasu za bojenje.

Takođe, prikazaće se rukovanje porukama u *web.php*. Zapravo, kada korisnik izvrši akciju koja zahteva povratnu informaciju, poput brisanja zadatka, u *web.php* fajlu, ruta koja upravlja brisanjem dodaje poruku uspeha u sesiju koristeći **`->with('success', 'Task deleted successfully');`**. Ovo efikasno postavlja poruku sesije koja će biti dostupna na sledećem HTTP zahtevu:

```
Route::delete('/tasks/{task}', function (Task $task) {
   $user = Auth::user();
   $userName = $user->name;
   $taskUpdate = Task::find($task->id);
   $taskUpdate->user_admin = $userName;
   $taskUpdate->save();
   $task->delete();

   return redirect()->route('tasks.index')
     ->with('success', 'Task deleted successfully');
})->middleware(['auth', 'verified'])->name('tasks.destroy');
```

Slika 7. Primer koda koji implementira DELETE rutu za brisanje zadatka u Laravel aplikaciji
Izvor: Autorska obrada

U navedenom kodu nakon što se zadatak uspešno obriše, korisnik se redirektuje nazad na listu zadataka sa sesijskom porukom *'success'*, koja sadrži tekst *'Task deleted successfully'*. Zahvaljujući implementaciji u *layouts/app.blade.php*, ova poruka će biti prikazana korisniku na sledećoj stranici koju poseti.

Prikazivanje poruke greške

Laravel koristi *@error* direktivu za prikazivanje poruke greške ukoliko validacija za *description* polje ne prođe. Ova direktiva hvata i prikazuje specifičnu poruku greške koju Laravel generiše za to polje. Poruka je obuhvaćena elementom sa klasama za stilizovanje, što se može videti na slici 8.

```
<input type="text" name="description" id="description" placeholder="Your description"
value="{{ $task->description }}" class="text-white bg-blue-950/80 border-2 w-full p-4
rounded-lg @error('description') border-red-500 @enderror">
@error('description')
  <div class="text-red-500 mt-2 text-sm">
    {{ $message }}
  </div>
@enderror
```

Slika 8. Primer koda sa integrisanom validacijom grešaka u Laravel Blade template engine
Izvor: Autorska obrada

3.12 Ažuriranje, brisanje, čuvanje starih vrednosti u formi

Laravel pruža elegantne načine za upravljanje bazama podataka koristeći *Eloquent ORM*, kao i različite mehanizme sigurnosti, uključujući i zaštitu od CSRF (Cross-Site Request Forgery) napada. Veoma je važno predstaviti način na koji Laravel vrši ažuriranje, brisanje i čuvanje starih vrednosti u formama, sa posebnim osvrtom na CSRF zaštitu.

Ažuriranje

Da bi se ažurirali postojeći zapis u Laravel-u, prvo je potrebno preuzeti taj zapis iz baze, izmeniti njegove atribute i potom sačuvati te promene. Na primer, korišćenjem *Eloquent ORM*-a:

```
$user = User::find(1); // Preuzima korisnika sa ID-jem 1
$user->name = 'Novo ime'; // Menja ime korisnika
$user->save(); // Čuva promene u bazi
```

Slika 9. Primer ažuriranja podatka
Izvor: Autorska obrada

Proces ažuriranja zadatka u Laravel aplikaciji, uključujući kako se koristi *Blade* šablon za formu i kako su implementirane rute, može se opisati na sledeći način:

U *edit.blade.php* fajlu kreirana je forma koja omogućava korisnicima da ažuriraju zadatke. Ova forma je dostupna na URL-u *http://localhost:8000/tasks/{task}/edit*, gde *{task}* predstavlja ID zadatka koji treba ažurirati. Forma koristi HTTP metodu PUT, pomoću *Blade* direktive *@method('PUT')*. Kada korisnik popuni formu i pošalje je, forma se obrađuje kroz akciju PUT na URL-u *http://localhost:8000/tasks/{task}*, koja je definisana u Laravel ruteru. Ova akcija koristi metod *update* da ažurira zadatak.

Kada se forma pošalje, *update* metoda procesuira podatke. Ovaj metod proverava validnost unosa, ažurira zadatak u bazi podataka sa novim vrednostima i onda izvršava redirekciju na *task.show* rutu, šaljući *flash* poruku o uspehu "Task updated successfully".

Dodatno, aplikacija ima i funkciju za promenu stanja zadatka, dostupnu na *http://localhost:8000/tasks/{task}/toggle-complete*. Pozivom ove rute, stanje zadatka se menja, nakon čega sledi redirekcija na *task.show* rutu sa istom *flash* porukom "Task updated successfully".

Brisanje

Brisanje zapisa je slično ažuriranju, samo što umesto čuvanja izmena, zapis se briše iz baze na sledeć način:

```
$user = User::find(1); // Preuzima korisnika sa ID-jem 1
$user->delete(); // Briše zapis iz baze
```

Proces brisanja zadatka u aplikaciji implementiran je kroz formu koja koristi HTTP metodu DELETE. Ova metoda je omogućena pomoću *@method Blade* direktive, što Laravel-u omogućava da interpretira pravilno POST zahtev kao DELETE zahtev. Ova forma se nalazi na ruti *http://localhost/tasks/{task}*, gde *{task}* predstavlja identifikator zadatka koji treba obrisati.

Forma za brisanje zadatka sadrži *@csrf* direktivu kako bi se osigurala zaštita od CSRF napada, čime se povećava bezbednost aplikacije.

Kada korisnik klikne na dugme za brisanje zadatka izvršava se gore navedena ruta, zadatak se briše, vrši se redirekcija na *home* stranu sa *flesh* porukom kako je uspešno izvršeno brisanje podatka.

Čuvanje starih vrednosti

Laravel omogućava lako čuvanje i pristupanje starim vrednostima unutar forme kroz globalnu pomoćnu funkciju *old()*. Ovo je posebno korisno kada forma nije uspešno prosleđena i potrebno je da se prikaže ponovo sa originalnim vrednostima koje je korisnik uneo:

```
<input type="text" name="username" value="{{ old('username') }}">
```

Ova funkcija automatski koristi podatke iz sesije da bi popunila polja sa vrednostima koje su prethodno unete.

3.13 Dodavanje paginacije

Paginacija je ključna funkcionalnost za efikasno upravljanje prikazom velikog broja zapisa u veb aplikacijama. Laravel nudi ugrađenu podršku za paginaciju kroz svoj ORM sistem, *Eloquent*, što olakšava integraciju paginacije sa minimalnim kodiranjem. Ovaj deo zapravo objašnjava proces dodavanja paginacije na listu zadataka u Laravel aplikaciji.

Kreiranje paginacije

Da bi se implementirala paginacija, prvo je potrebno prilagoditi upit koji preuzima zadatke iz baze podataka tako što će se koristiti *paginate()* metoda modela *Eloquent*. Na primer, ako je potrebno prikazati 10 zadataka po stranici, upit bi izgledao na sledeći način:

```
$tasks = Task::orderBy('created_at', 'desc')->paginate(10);
```

Ovaj kod izvlači zadatke iz baze podataka, sortira ih prema datumu kreiranja (od najnovijih ka najstarijima) i ograničava broj prikazanih rezultata na 10, po stranici.

Prikaz paginacije u Blade šablonu

Nakon što je paginacija implementirana u kontroleru, potrebno je dodati kontrole za navigaciju paginacije u *Blade* šablonu, tako što će se koristiti ugrađena *Blade* direktiva *{{$tasks->links() }}*. Ova direktiva automatski generiše HTML kod potreban za navigacione linkove paginacije. Prikaz kontrole paginacije može se dodati na dnu liste zadataka u *Blade* šablonu:

```
<div class="container">
    @foreach ($tasks as $task)
        <div>{{ $task->name }}</div>
    @endforeach
    {{ $tasks->links() }}
</div>
```

Slika 10. Primer preuzimanja zadataka i prikazivanja paginacija
Izvor: Autorska obrada

Prilagođavanje izgleda paginacije

Laravel koristi *Tailwind CSS* za inicijalno stilizovanje paginacije, ali je moguće prilagoditi izgled paginacije prema potrebama aplikacije. Laravel omogućava lako prilagođavanje navigacionih linkova tako što će se objaviti navigacione *view* komponente u Laravel projektu, a koje se mogu modifikovati po želji:

```
php artisan vendor:publish --tag=laravel-pagination
```

Navedena komanda kopira inicijalne navigacione *view* datoteke u *resources/views/vendor/pagination* direktorijum, gde se mogu prilagoditi prema potrebama dizajna aplikacije.

U aplikaciji, paginacija je implementirana koristeći *Eloquent* model *Task* unutar Laravel-a. Metoda *paginate(15)* koristi se za automatsko deljenje zapisa iz baze podataka na stranice, sa limitom od 15 zadataka po stranici. Ova metoda olakšava upravljanje velikim setovima podataka tako što korisnicima omogućava da pregledaju zadatke u manjim, upravljivim blokovima, umesto da se svi zadaci učitaju odjednom.

```
Route::get('/tasks', function () {
    // return all tasks with pagination of 15
    return view('tasks', [
      'tasks' => Task::paginate(15)
    ]);
  })->middleware(['auth', 'verified'])->name('tasks.index');
```

3.14 Prebacivanje stanja zadatka

U Laravel aplikaciji, funkcionalnost promene stanja zadatka (na primer, iz "otvoren" u "rešen" i obrnuto) omogućava korisnicima efikasno upravljanje svojim zadacima. Ova funkcionalnost se implementira korišćenjem dugmeta "Toggle" koje pokreće akciju u kontroleru putem HTTP POST zahteva. Segmeti upravljanja zadacima su:

- **Ruta:**

Ruta *tasks.toggle-complete* definiše se u Laravel ruteru (web.php), gde se za obradu zahteva koristi odgovarajući kontroler. Metoda HTTP POST se koristi za inicijalni zahtev, dok Laravel-ova *@method* direktiva određuje da se zapravo izvršava HTTP PUT zahtev. Ovo je primer RESTful prakse gde PUT zahtevi označavaju ažuriranje resursa.

- **Kontroler:**

U kontroleru koji upravlja zadacima, metoda *toggleComplete()* bi se bavila logikom za promenu stanja zadatka. Ova metoda proverava trenutno stanje zadatka, ažurira ga u suprotno stanje (otvoren/rešen), i čuva promene u bazi podataka.

- **Prikaz i povratne informacije:**

Nakon što je akcija izvršena, korisnik se preusmerava nazad na detalje zadatka sa *flash* porukom koja obaveštava o uspešnoj promeni stanja. *Flash* poruke u Laravelu se često koriste za jednokratna obaveštenja koja se prikazuju nakon redirekcije.

- **Frontend implementacija:**

Na *frontend*-u, dugme "Toggle" može biti implementirano unutar forme koja sadrži skrivene *input* elemente za identifikaciju zadatka. JavaScript ili Livewire (za real-time aplikacije) mogu se koristiti za dodavanje dinamičkog ponašanja, kao što je ažuriranje statusa zadatka bez osvežavanja stranice.

- **Sigurnost:**

Važno je osigurati da akcije promene stanja mogu izvršavati samo autorizovani korisnici. *Middleware* za autentifikaciju i autorizaciju može se koristiti za zaštitu rute i obezbeđivanje da samo vlasnici ili ovlašćeni korisnici mogu menjati stanje zadatka.

- **Testiranje**:

Testiranje ove funkcionalnosti može uključivati jedinične i funkcionalne testove. Jedinični testovi mogu se fokusirati na metodu kontrolera, dok funkcionalni testovi mogu simulirati klik na dugme "Toggle" i verifikovati odgovarajuće redirekcije i promene u stanju zadatka.

3.15 Dodavanje stila sa Tailwind CSS okvirom

U savremenom razvoju veb aplikacija, jedan od ključnih izazova je postizanje optimalnog balansa između funkcionalnosti, estetike i performansi. Laravel, kao jedan od vodećih PHP okvira, pruža robusnu osnovu za izgradnju skalabilnih i sigurnih veb aplikacija. Međutim, za razvoj korisničkog interfejsa, Laravel se oslanja na dodatne alate i biblioteke. U ovom kontekstu, izbor *Tailwind CSS*-a kao alata za stilizaciju predstavlja strateški važnu odluku.

Tailwind CSS, moderan i funkcionalan CSS okvir, omogućava developerima da brzo i efikasno dizajniraju korisničke interfejse bez potrebe za pisanjem velike količine prilagođenog CSS-a. Za razliku od tradicionalnih CSS okvira, kao što su *Bootstrap* ili *Foundation*, koji dolaze sa predefinisanim stilovima komponenata, *Tailwind CSS* pruža više kontrole i fleksibilnosti kroz *utility-first* pristup. Ovaj pristup omogućava razvojnim programerima da konstruišu dizajn koristeći funkcionalne klase direktno unutar HTML-a, što vodi do bržeg razvojnog procesa i manje složenog koda.

Integracija *Tailwind CSS*-a u Laravel projekte dodatno je pojednostavljena kroz podršku alata kao što su *Laravel Mix* i *Vite*, koji olakšavaju kompilaciju i smanjivanje koda. Smanjivanje i optimizacija koda je proces optimizacije web sadržaja, kao što su HTML, CSS i JavaScript fajlovi, sa ciljem smanjenja njihove veličine tako što se uklanjaju nepotrebni razmaci, beline, komentari i drugi suvišni znakovi. Ovo, u kombinaciji sa Laravel-ovim efikasnim metodama za rukovanje rutama, autentifikacijom i sesijama, čini celokupnu platformu izuzetno pogodnom za izgradnju modernih, odzivnih i vizuelno privlačnih veb aplikacija.

Korišćenje *Tailwind CSS*-a u Laravel-u nije samo pitanje estetike, već i pitanje efikasnosti u razvoju i performansi aplikacije. S obzirom na to da *Tailwind* omogućava stilizaciju "na mestu", razvojni programeri mogu brzo iterirati i testirati promene u dizajnu bez potrebe za dubokim pristupom u eksterne stilove ili CSS fajlove. Ovaj pristup ne samo da ubrzava razvoj,

već i pomaže u održavanju čistoće i modularnosti koda, što je ključno za skalabilne i održive aplikacije.

Potrebno je detaljno razmotriti tehničke aspekte implementacije *Tailwind CSS*-a unutar Laravel okvira, kao prikazati i praktične primere koji ilustruju prednosti ovog pristupa u stvarnim aplikacijama.

Stilizovanje cele veb aplikacije

Stilizovanje kompleksnih veb aplikacija zahteva efikasan i modularan pristup, posebno kada je u pitanju integracija sa moćnim *backend* sistemima kao što je Laravel. Laravel je poznat po svojoj agilnosti i fleksibilnosti, omogućavajući razvijanje robustnih aplikacija sa bogatim funkcionalnostima. Međutim, za postizanje privlačnih korisničkih interfejsa, neophodno je pažljivo planiranje i implementacija frontend aspekata.

1. Konzistentnost i modularnost

Korišćenjem komponenti *Blade*, kao što su *<x-app-layout>* i *<x-guest-layout>*, Laravel omogućava razvoj konzistentnih i ponovno upotrebljivih UI elemenata. Ove komponente mogu biti stilizovane koristeći CSS okvir kao što je *Tailwind CSS,* koji nudi veliku fleksibilnost.

2. Pristup "Utility-first"

Tailwind CSS, kao "utility-first" okvir, pruža razvijanje seta *utility* klasa koje se mogu primeniti direktno na HTML elemente unutar *Blade* šablona. Ovo omogućava brzo stilizovanje i vizuelne izmene bez potrebe za pisanjem dugih i kompleksnih CSS fajlova. Na primer, u kreiranju forme za zadatke, korišćene su klase poput *bg-gray-100*, *border-2*, i *rounded-lg* za brzo i efikasno stilizovanje.

3. Održivost i skalabilnost

Stilizovanje pomoću *Tailwind CSS*-a, Laravel aplikaciji pomaže u održivosti i skalabilnosti projekta. S obzirom na to da stilovi nisu u vezi sa specifičnim komponentama već su globalno dostupni i konzistentni, ovo smanjuje ponavljanje i povećava efikasnost razvojnog procesa. *Utility* klase mogu se koristiti i prilagođavati na nivou cele aplikacije, što olakšava održavanje i ažuriranje stilova.

4. Brza iteracija i testiranje

Prilagodljivost i modularnost *Tailwind CSS*-a omogućava brze iteracije dizajna i stilova, što je idealno za agilne razvojne procese. Promene u dizajnu se mogu brzo implementirati i testirati bez potrebe za zahvatima u CSS-u, što je često vremenski zahtevno i sklonije greškama.

5. Integracija i automatizacija

Laravel Mix, alat za upravljanje resursima u Laravel-u, omogućava jednostavnu integraciju i automatizaciju procesa kao što su kompilacija i smanjivanje koda u CSS fajlovima. Ovo značajno poboljšava performanse aplikacije, smanjujući vreme učitavanja i optimizujući isporuku resursa krajnjim korisnicima.

Ukratko, stilizovanje veb aplikacija u Laravel-u koristeći *Tailwind CSS* nije samo pitanje estetske preferencije, već i strateški izbor koji doprinosi efikasnosti, održivosti i skalabilnosti projekta. Ovaj pristup omogućava da se fokus prebaci sa tehničkih detalja stilizovanja na stvaranje korisničkih iskustava koja su i funkcionalna i vizuelno privlačna.

ZAKLJUČAK

Laravel se ističe kao jedan od značajnijih okvira za razvoj aplikacija iz više razloga. Prvo, njegova osnovna svrha je brz razvoj aplikacija. Fokusiran je na jednostavno učenje i minimizaciju koraka od početka izrade nove aplikacije do njenog puštanja u rad. Svi uobičajeni zadaci u izgradnji veb aplikacija, od interakcije sa bazom podataka do autentifikacije, olakšani su zahvaljujući komponentama koje Laravel pruža. Osim toga, Laravel nudi čitav sistem alata koji olakšavaju razvoj i puštanje aplikacija u rad, uključujući alate za lokalni razvoj, upravljanje serverima i napredne alate za implementaciju.

Jedna od ključnih karakteristika Laravela je "konvencija pre konfiguracije". To znači da ako se koriste Laravelove podrazumevane postavke, znatno će se smanjiti obim posla u odnosu na druge okvire koji zahtevaju detaljno konfigurisanje čak i ako se koriste preporučene postavke. Projekti izgrađeni na Laravelu često zahtevaju manje vremena nego oni izgrađeni na drugim PHP okvirima.

Takođe, Laravel se ističe fokusom na jednostavnost. Dok je moguće koristiti razne složene arhitektonske obrasce sa Laravelom, dokumentacija i zajednica obično preporučuju početak sa najjednostavnijom mogućom implementacijom. Ovo omogućava programerima da kreiraju najjednostavniju moguću aplikaciju koja rešava njihove potrebe.

Laravel-ova inspiracija dolazi iz drugih jezika i okvira poput Rubyja, Railsa i funkcionalnih programskih jezika, umesto iz Java sveta kao kod većine drugih PHP okvira. Dok mnogi moderni PHP okviri teže ka sveobuhvatnosti i kompleksnosti, Laravel se okreće ekspresivnim, dinamičnim i jednostavnim praksama kodiranja i karakteristikama jezika. Ova kombinacija karakteristika čini Laravel jednim od najpopularnijih i najcenjenijih PHP okvira za razvoj modernih veb aplikacija.

LITERATURA

1. Al-Hawari F., Barham H. (2019). „A machine learning based help desk system for IT service management", Jurnal of King Saud University – Computer and Information Sciences(2021):702 - 718.
2. Al-Sharji S., Al-Mahruqi A., Kumar R. (2014) „Help Desk Management System for PC Troubleshooting", IJAIS Volume 7, No.7: 8 – 14.
3. Begović B., Atanasijević S., Dulić S., Milićević V. (2014) „Primena HelpDesk-a u funkciji povećanja efikasnosti poslovanja u državnoj upravi", Poreska Uprava Srbije, Visoka Tehnička škola strukovnih studija Kragujevac
4. Frantz G. M. (2007) . „The Evolution of the IT Help Desk to the Service Desk", CCSI Technology Solutions, White Paper
5. Gohil F., Kumar V. (2019). „Ticketing System", IJTSRD Volume 3, Issue 4: 155 – 156.
6. Harcenko M., Dorogovs P., Romanovs A. (2010). „IT Service Desk Implementation Solutions", Scientific Journal of Riga Technical University:
7. Knight M. (2005). „Analysis, Design and Implementation of a Helpdesk Management System", University of Leeds, School of Computer Studies
8. Kumar N. G. (2013) „Electronic Support Ticket Management System", IJERT Volume 2, Issue 9: 1544 – 1547.
9. Masongsong P. R., Damian E. M. A. (2016). „Help Desk Management System", World Congress on Engineering and Computer Science 2016
10. Milićević L. V. (2015). „Unapređenje savremenih HelpDesk poslovnih sistema primenom naprednih inteligentnih softverskih alata", Ekonomski fakultet u Nišu
11. Popović J. (2021). „Razvoj modernih veb-aplikacija'', CET, Beograd
12. Poyenyard P. (2002). „Help Desk System for IT Service", Computer and Engineering Management of Assumption University
13. Rachmawati E., Suhendra, Kom S., Kom M. (2018). „Web-Based Ticketing System Helpdesk Application Using Codeigniter Framework (Case Study: PT Commonwealth Life)", IJCSMC Volume 7 Issue. 12: 29 – 41.
14. Racz C. (2021). „Learn Primary Skills Of LARAVEL Fast And Easily: Laravel PHP Web", Kompjuter biblioteka, Beograd

15. Roth – Berghofer R. T. (2004). „Learning from HOMER, a Case – Based Help Desk Support System", Advances in Learning Software Organizations: 88 – 97.
16. Serbest S., Goksen Y., Dogan O., Tokdemir A. (2015). „Design and Implementation of Help Desk System on the Effective Focus of Information System", Procedia Economics and Finance
17. Stauffer M. (2019). Laravel - radni okvir za izradu modernih PHP aplikacija, Kompjuter biblioteka, Beograd
18. Trachtenberg A., Sklar D. (2008). „PHP kuvar'', Mikro knjiga, Beograd
19. Welling L., Thomson L. (2017). „PHP i MYSQL Razvoj aplikacija za veb'', Mikro knjiga, Beograd
20. Zandstra M. (2021). "PHP 8 objekti, obrasci i praksa", Kompjuter biblioteka, Beograd

Veb izvori:

1. https://stackoverflow.com
2. https://laravel.com
3. https://www.w3schools.com
4. https://github.com
5. https://www.docker.com
6. https://tailwindcss.com
7. https://www.php.net
8. https://www.phpmyadmin.net
9. https://osticket.com
10. https://itsourcecode.com/free-projects/laravel/task-management-system-project-in-laravel-with-source-code/