Usporedba Angular i Vue.js radnih okvira za razvoj web aplikacija

Monjac, Antonio

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

Usporedba Angular i Vue.js radnih okvira za razvoj web aplikacija

Rijeka, rujan 2023.

Antonio Monjac

0069088998

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

Usporedba Angular i Vue.js radnih okvira za razvoj web aplikacija

Mentor: doc.dr.sc. Marko Gulić

Rijeka, rujan 2023.

Antonio Monjac

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 8. ožujka 2023.

Zavod: **Zavod za računarstvo**
Predmet: **Razvoj web aplikacija**
Grana: **2.09.06 programsko inženjerstvo**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Antonio Monjac (0069088998)**
Studij: Sveučilišni prijediplomski studij računarstva

Zadatak: **Usporedba Angular i Vue.js radnih okvira za razvoj web aplikacija / The comparison of Angular and Vue.js frameworks for web application development**

Opis zadatka:

Treba napraviti detaljnu usporedbu Angular i Vue.js radnih okvira klijentske strane koji se koriste za razvoj web aplikacija. Izvršit će se usporedba ta dva radna okvira pomoću dvije identične RESTful jednostranične aplikacije (Single Page Application, SPA) koje će biti izrađene pomoću zadanih radnih okvira. Aplikacije trebaju sadržavati osnovne CRUD (create, read, update, delete) operacije i autentifikaciju. Treba analizirati mogućnosti, arhitekturu, potrošnju resursa i načine razvoja web aplikacija pomoću ova dva radna okvira. Za razvoj poslužiteljskog dijela web aplikacije treba koristiti Laravel radni okvir uz proizvoljno odabran sustav za upravljanje bazama podataka. Također, za realizaciju autentifikacije s klijentskom stranom, može se koristiti proizvoljan Laravel paket za autentifikaciju.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 26. ožujka 2023.

Mentor:

Doc. dr. sc. Marko Gulić

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Miroslav Joler

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2023.

_____

Antonio Monjac

Zahvala

Zahvaljujem se roditeljima i curi na podršci tijekom studiranja. Zahvale mentoru doc. dr. sc. Marku Guliću na strpljenju i pruženom vremenu pri izradi ovog završnog rada.

SADRŽAJ

Popis slika

1 Uvod

U današnjem digitalnom dobu, razvoj web aplikacija postao je neizostavan dio modernog softverskog inženjeringa. Sve više organizacija i pojedinaca prepoznaje važnost brzog, skalabilnog i pouzdanog razvoja web aplikacija kako bi ispunili zahtjeve tržišta i korisnika. Upravo iz tih razloga došlo je do potrebe razvoja raznih radnih okvira (eng. frameworks) koji uvode pravila pri pisanju i strukturu pri izgradnji web aplikacija.

Ovaj završni rad ima cilj usporediti dva radna okvira, Angular i Vue.js. Angular je besplatan radni okvir otvorenog koda baziran na programskom jeziku TypeScript. Prva inačica Angulara razvijena je 2009. godine od strane tvrtke Google, koji ga je podržavao i razvijao sve do 2022. godine. Google-ova dugoročna podrška za Angular prekinuta je u siječnju 2022., no Angular je i dalje dostupan. Vue.js također je radni okvir otvorenog koda, baziran je na JavaScript-u te se koristi za izradu jednostraničnih aplikacija i korisničkih sučelja. Razvio ga je 2014. godine Evan You koji ga i danas održava zajedno sa svojim timom. Za razvoj poslužiteljskog dijela web aplikacije korišten je Laravel radni okvir koji služi kao aplikacijsko programsko sučelje (eng. application programming interface, skraćeno API). Laravel je PHP radni okvir otvorenog koda koji služi za razvoj web aplikacija temeljenih na MVC (*model-view-controller*) arhitekturi.

Usporedba radnih okvira biti će odrađena tako što će se analizirati njihove prednosti i nedostatci, principi njihove strukture te u konačnici brzina izvedbe kako bi se dobio sveobuhvatan uvid u njihove mogućnosti i ograničenja. To će biti izvedeno na primjeru dvije identične *To-Do* RESTful jednostranične aplikacije (eng. Single Page Application, skraćeno SPA) izrađene pomoću zadanih radnih okvira. Aplikacije korisniku omogućuju osnovne CRUD (Create, Read, Update, Delete) operacije odnosno može dodavati, dohvaćati, uređivati i brisati svoje zadatke sa liste. Aplikacije u oba radna okvira omogućavaju registraciju i autentifikaciju korisnika izvedenu tako da svaki korisnik ima svoju listu zadataka povezanu preko identifikatora korisnika. Također, postoji i *admin* korisnik koji može upravljati svim zadacima sa svih lista korisnika. Aplikacija u oba radna okvira izgleda vrlo slično te koristi iste globalne postavke za stil i dizajn. Ovaj rad će pružiti detaljan uvid u karakteristike, prednosti i nedostatke Angulara i Vue.js-a.

2 Opis tehnologija

2.1 Angular

Angular je duboko ukorijenjen u svijetu razvoja web aplikacija i predstavlja moćan i kompleksan okvir za izradu korisničkih sučelja i SPA aplikacija. Razvijen od strane Googlea, ovaj okvir je evoluirao tijekom godina, a trenutna verzija, Angular 16, pruža širok spektar alata i koncepta za izradu aplikacija visokih performansi [1].

Jedna od ključnih karakteristika Angulara je njegova potpuna arhitektura. Angular pruža strukturu i organizaciju aplikacije kroz komponente, servise i module. Komponente predstavljaju temeljne gradivne jedinice aplikacije i omogućuju razdvajanje korisničkog sučelja na manje dijelove. Servisi se koriste za dijeljenje logike i podataka između komponenata, dok moduli omogućavaju organizaciju aplikacije i uključivanje vanjskih knjižnica.

2.1.1 Angular CLI

Angular CLI (eng. Command Line Interface) je alat koji olakšava razvoj i upravljanje Angular aplikacijama putem naredbenog retka. Samo neke od mogućnosti Angular CLI-a jesu:

a) Generiranje komponenti i modula - Angular CLI omogućuje brzo generiranje različitih komponenti, modula, usluga i drugih struktura projekta, moguće je izraditi novu komponentu jednostavnom naredbom poput *ng generate component componentName*
b) Razvojni poslužitelj: CLI dolazi s ugrađenim razvojnim poslužiteljem koji omogućuje pokretanje lokalne verzije Angular aplikacije
c) Upravljanje ovisnostima (eng. dependencies): CLI olakšava instalaciju i upravljanje ovisnostima projekta. Korištenjem naredbe *ng add* omogućuje dodavanje knjižnica, modula i drugih paketa u aplikaciju
d) Testiranje: CLI uključuje alate za testiranje Karma i Jasmine za jednostavno pisanje i izvođenje testova

e) Konfiguracija: CLI generira konfiguracijske datoteke kao što su *angular.json* i *tsconfig.json* koje olakšavaju početnu prilagodbu i konfiguraciju projekta [2]

2.1.2 TypeScript

Angular aplikacije izgrađene su korištenjem TypeScript jezika, superskripta za JavaScript, koji osigurava veću sigurnost budući da podržava tipove poput primitiva i sučelja (eng. primitives and interfaces). Pomaže otkriti i ukloniti pogreške rano u procesu pisanja koda ili prilikom njegovog održavanja. Također omogućuje izravno otklanjanje pogreške TypeScript koda u pregledniku ili uređivaču ukoliko postoji odgovarajuća struktura datoteka stvorena prilikom izgradnje aplikacije. TypeScript osigurava poboljšanu navigaciju kroz kod, refaktoriranje i usluge automatskog dovršavanja.

2.1.3 Komponente i DOM

Angular se temelji na konceptu komponenata, koje su temeljne gradivne jedinice aplikacije. Ovakav način strukturiranja aplikacije omogućuje ponovno korištenje funkcionalnih dijelova koda u obliku komponente na više mjesta u kodu.

Također, podržava direktive, koje omogućuju manipulaciju DOM-om i stvaranje dinamičkih aplikacija. *Document Object Model* (skraćeno DOM) predstavlja stranicu tako da programi odnosno skripte mogu promijeniti strukturu dokumenta (HTML-a), njihov stil i sadržaj. DOM predstavlja dokument kao čvorove i objekte te na taj način programski jezici mogu komunicirati sa stranicom [3].

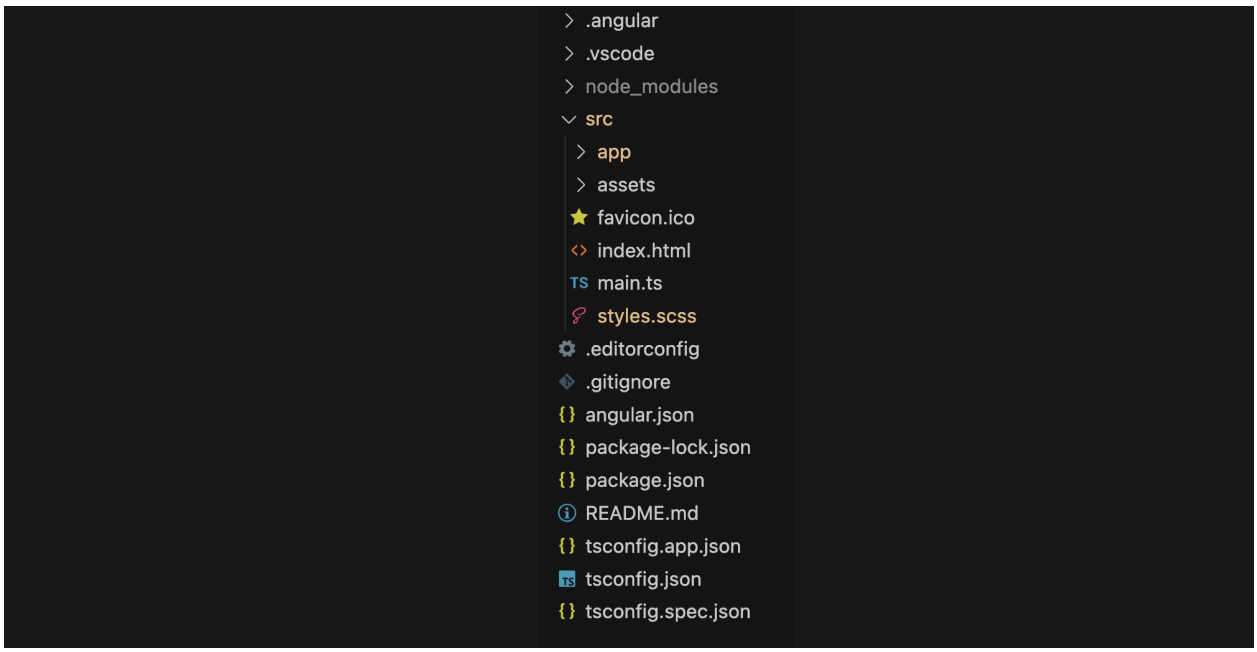
2.1.4 POJO (Plain Old JavaScript Object)

Pri korištenju Angular nije potrebno definirati *getter* i *setter* funkcije. To je zato što je svaki objekt koji Angular koristi POJO (*Plain Old JavaScript Object*) koji omogućuje manipulaciju objektom pružajući sve konvencionalne JavaScript funkcije.

2.1.5 Struktura Angular aplikacije

Pri izradi nove Angular aplikacije generira se struktura mapa i datoteka koja olakšava samo korištenje i snalaženje u radnome okviru. Struktura Angular aplikacije prikazana je na slici 2.1 te je od iznimne važnosti zato što je to jedna od razlika Angulara i drugih JavaScript radnih okvira. Svrha svake od mapa i datoteka je sljedeća:

a) *src* – osnovne izvorne datoteke i izvorni kod Angular aplikacije
b) *app* – sadrži datoteke komponenti i servisa u kojima je definirana logika aplikacije
c) *assets* – sadrži slikovne i druge datoteke sredstava koje se u aplikaciji pozivaju i koriste onakve kakve jesu
d) *index.html* – glavna HTML stranica koja se poslužuje pri posjeti stranice, Angular CLI u ovu datoteku automatski dodaje sve JavaScript i CSS datoteke prilikom izrade aplikacije
e) *main.ts* – glavna ulazna točka aplikacije, kompajlira aplikaciju s JIT kompajlerom i pokreće korijenski modul aplikacije (AppModule) za pokretanje u pregledniku
f) *styles.scss* – datoteka u kojoj se nalaze svi globalno primijenjeni stilovi aplikacije, proširenje (*.scss*) odražava pretprocesor stila koji je konfiguriran za projekt
g) tsconfig.app.json – TypeScript konfiguracija specifična za aplikaciju, uključuje opcije prevoditelja Angular predložaka i samog TypeScripta
h) tsconfig.spec.json – TypeScript konfiguracija za testiranje aplikacije [1]



Slika 2.1 Struktura Angular aplikacije

2.2 Vue.js

Vue.js, moderni JavaScript okvir za izradu korisničkih sučelja, zauzima značajno mjesto u svijetu web razvoja. Njegov razvoj započeo je 2014. godine, a glavni arhitekt i programer Evan You osmislio je Vue.js s naglaskom na jednostavnost, reaktivnost i lakoću upotrebe. Vue.js se često uspoređuje s drugim popularnim okvirima kao što su Angular i React. Jedan od glavnih razloga njegove popularnosti leži u njegovoj lakoći učenja. Programeri brzo usvajaju Vue.js zahvaljujući njegovoj jasnoj sintaksi i konceptima.

Jedna od ključnih karakteristika Vue.js-a je njegova reaktivnost. Vue.js koristi reaktivni sustav za praćenje promjena u podacima i automatsko ažuriranje korisničkog sučelja kad se ti podaci promijene. Ovaj pristup čini izradu dinamičkih aplikacija intuitivnom i efikasnom. Komponentna arhitektura još je jedna bitna karakteristika Vue.js-a. Aplikacija se razdvaja na manje komponente, što olakšava ponovnu upotrebu koda i održavanje aplikacije. Vue komponente su autonomne i komunikacija između njih olakšava se putem *propsa* i emitiranja događaja. Uz osnovni okvir, Vue.js dolazi s bogatim ekosustavom i dodacima. Vue *Router* omogućava jednostavno upravljanje rutama u aplikaciji, dok *Vuex* omogućava globalno upravljanje stanjima. Vue CLI pruža alate za brzo i jednostavno postavljanje projekta. [4]

2.2.1 Vue CLI

Vue CLI je alat za brzi razvoj Vue.js aplikacija koji pruža:

- Brzo i interaktivno kreiranje Vue aplikacija koristeći @vue/cli
- *Runtime dependency* koji omogućuje:
    - Nadogradnju
    - Konfiguraciju putem konfiguracijske datoteke unutar projekta
    - Proširivost putem dodataka (eng. plugins)

Cilj Vue CLI-a je biti standardna baza alata za Vue ekosustav. Osigurava da različiti alati za izradu rade nesmetano zajedno sa smislenim zadanim postavkama tako da se programer može usredotočiti na pisanje aplikacije umjesto konfiguriranja. No, još uvijek nudi fleksibilnost za podešavanje konfiguracije svakog alata. CLI *(@vue/cli*) je globalno instalirani *npm* paket i omogućuje korištenje naredbe *vue* u naredbenom retku ili terminalu. Pruža mogućnost brzog postavljanja novog projekta putem *vue create-a*. Također pruža mogućnost upravljanja projektima

koristeći grafičko korisničko sučelje putem *vue ui*. CLI dodaci su *npm* paketi koji pružaju opcionalne značajke Vue CLI projektima, kao što je kompilacija *Babel/TypeScript*, integracija *ESLinta*, jedinično testiranje i *end-to-end* testiranje. Lako je uočiti Vue CLI dodatke jer njihova imena počinju s *@vue/cli-plugin-* (za ugrađene dodatke) ili *vue-cli-plugin-* (za dodatke koje je stvorila zajednica) [5].

2.2.2 Komponente s jednom datotekom

U većini Vue.js projekata, komponente se rade koristeći format datoteke nalik HTML-u koji se naziva komponenta s jednom datotekom (također poznat kao *.vue datoteke). Komponente s jednom datotekom, kao što im i ime sugerira, inkapsuliraju logiku komponente (JavaScript), predložak (HTML) i stilove (CSS) u jednoj datoteci. Ovakav način pisanja komponenti olakšava čitljivost i održivost samog koda. Na slici 2.2 vidi se kako su logika (*script*), predložak (*template*) i stil (*style*) smješteni u samo jednu datoteku, što bi kod većine drugih radnih okvira zahtijevalo tri ili više datoteka.

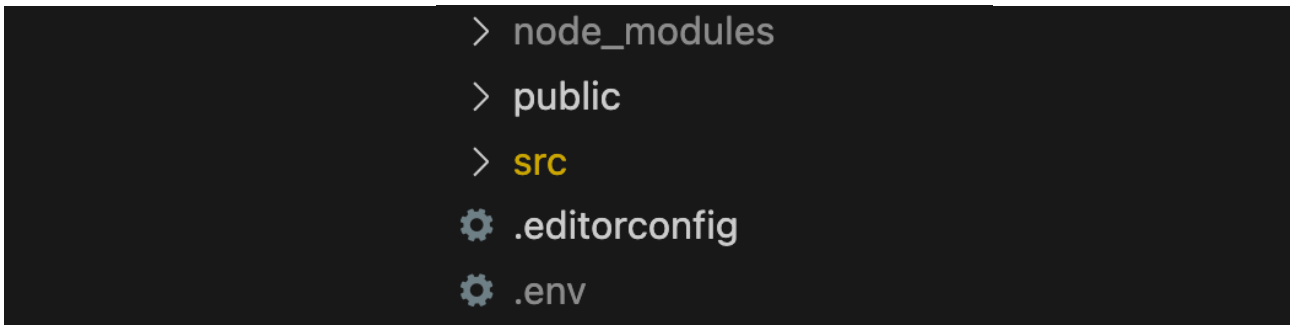
```
<script setup>
import { ref } from 'vue'
const count = ref(0)
</script>

<template>
  <button @click="count++">Count is: {{ count }}</button>
</template>

<style scoped>
button {
  font-weight: bold;
}
</style>
```

Slika 2.2 Komponenta s jednom datotekom

2.2.3 Struktura Vue.js aplikacije

Jednako kao i kod Angular, pri kreiranju nove Vue.js aplikacije datoteke i mape smještene su s obzirom na određenu strukturu. Ta struktura specifična je za aplikacije rađene u Vue.js radnom okviru, a bitni direktoriji i mape prikazani su na slici 2.3. Svrha svake od direktorija i datoteka je sljedeća:

a) *public* direktorij - Sadrži statičke resurse koji se ne obrađuju pomoću *Webpacka* ili bilo kojeg alata za izgradnju, iznimka je datoteka *index.html*
b) *node_modules* – Postoji u aplikaciji samo ako se izvrši naredba *npm install* u konzoli, a sadrži knjižnice poput *axiosa* i ostale instalirane module
c) *src* direktorij - Sadrži jezgru Vue.js aplikacije te se u njemu nalaze sljedeće važne datoteke:
    a. *main.js* - Ova datoteka inicijalizira Vue aplikaciju i označava kojem HTML elementu u datoteci index.html aplikacija treba biti priložena, ovdje se registriraju globalne komponente ili dodaju Vue knjižnice
    b. *App.vue* - Komponenta najviše razine u aplikaciji Vue [6]



Slika 2.3 Arhitektura Vue aplikacije

2.3 Laravel

Laravel je radni okvir za izradu poslužiteljske strane aplikacije koji se temelji na PHP-u. PHP je popularan jezik za izradu web aplikacija te omogućava obradu HTTP zahtjeva i upravljanje bazama podataka. Pri izradi To-Do aplikacije u ovome radu, Laravel je korišten kao API za komunikaciju radnih okvira sa bazom podataka.

2.3.1 Laravel Artisan

Artisan je sučelje naredbenog retka uključeno u sam Laravel. Artisan postoji u korijenu aplikacije kao artisan skripta i pruža brojne korisne naredbe koje programeru olakšavaju izgradnju Laravel aplikacije. Za pregled popisa svih dostupnih *artisan* naredbi, koristi se naredba *php artisan list*.

2.3.2 Skalabilni radni okvir

Zahvaljujući Laravelovoj ugrađenoj podršci za brze, distribuirane sustave predmemorije kao što je Redis, horizontalno skaliranje s Laravelom je iznimno jednostavno. Laravel aplikacije osmišljene su da budu lako skalirane za obradu stotina milijuna zahtjeva.

2.3.3 Paketi i ekosustav

Laravel ima bogat ekosustav paketa koji olakšavaju integraciju različitih funkcionalnosti u samu aplikaciju. Neki od paketa koje Laravel nudi jesu:

a) Laravel Cashier: Koristi se za jednostavno integriranje sustava za plaćanje, kao što su Stripe i Braintree
b) Laravel Socialite: Omogućava brzu integraciju društvenih mreža za autentifikaciju i omogućava korisnicima prijavu putem vlastitih društvenih profila
c) Laravel Dusk: Koristi se za pisanje automatiziranih testova za web aplikacije, uključujući testiranje korisničkog sučelja
d) Laravel Breeze: minimalna, jednostavna implementacija svih Laravelovih značajki provjere autentičnosti

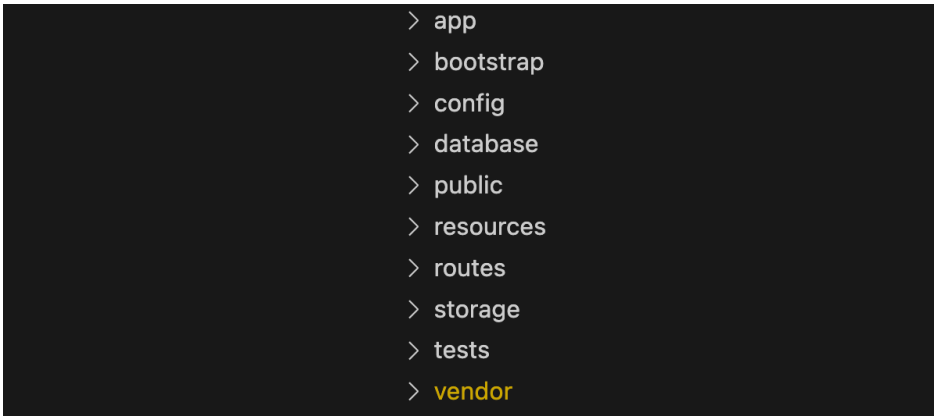
2.3.4 Eloquent

Laravel koristi Eloquent ORM sustav za objektno relacijsko mapiranje koji olakšava interakciju s bazom podataka. Pri korištenju Eloquenta, svaka tablica baze podataka ima odgovarajući model koji se koristi za interakciju s tom tablicom. Osim dohvaćanja zapisa iz tablice baze podataka, Eloquent modeli također omogućuju umetanje, ažuriranje i brisanje zapisa iz tablice.

2.3.5 Struktura Laravel aplikacije

Osnovna struktura Laravel aplikacije generirana pri samom kreiranju prikazana je na slici 2.4 te sadrži:

a) *app* direktorij – Sadrži osnovni kod aplikacije poput klasa, kontrolera i *middlewarea* (filtracija zahtjeva od strane klijenta upućenih serveru)
b) *bootstrap* direktorij – Sadrži datoteke za početno povezivanje ključnih komponenti Laravela i automatskog učitavanja. Također, sadrži "cache" direktorij s predmemoriranim podacima aplikacije
c) *config* direktorij – Sadrži datoteke za konfiguraciju općih postavki aplikacije, postavki za spajanje na bazu podataka i drugo
d) *database* direktorij – Sadrži generirane migracije, mehanizme za ispunjavanje tablica u bazi podataka te tvornice koje se koriste za generiranje testnih objekata
e) *node_modules* – Postoji u aplikaciji samo ako se izvrši naredba *npm install* u konzoli, a sadrži knjižnice poput Tailwinda i ostale instalirane module
a) *public* direktorij – U njemu se nalazi *index.php* datoteka koja je početna točka aplikacije pri njezinom učitavanju te svi ostali javni resursi poput datoteka za stilove, medije i JavaScript
b) *resources* direktorij – Sadrži poglede i nekompajlirane resurse
c) *routes* direktorij – Sadrži sve rute u aplikaciji; *api.php, web.php, console.php*
d) *storage* direktorij – U njemu se nalaze Blade predlošci, datoteke iz predmemorije i datoteke sesije te druge datoteke koje generira sam Laravel kao npr. zapise o pogreškama
e) *tests* direktorij – Sadrži skripte za testove
f) *vendor* direktorij – Sadrži aplikacijske zavisnosti [7]

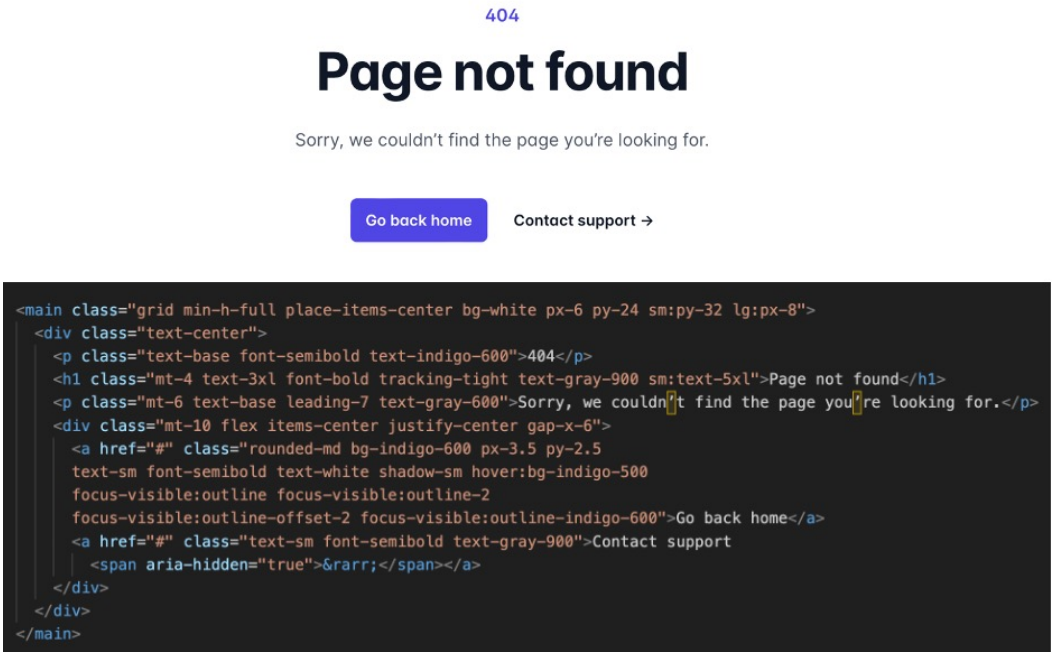


Slika 2.4 Struktura Laravel aplikacije

2.4 Tailwind

Tailwind CSS je popularan CSS okvir (eng. framework) koji se koristi za izradu modernih i prilagodljivih web sučelja. Ovaj okvir je dobio znatnu pažnju u zajednici web razvojnih programera zbog svoje jednostavne i ekspresivne sintakse koja omogućava brzo i učinkovito stiliziranje web stranica. Tailwind radi tako da skenira sve HTML datoteke, JavaScript komponente i sve druge predloške te u njima traži nazive klasa, generira odgovarajuće stilove i zapisuje ih u statičnu CSS datoteku. Tailwind automatski uklanja sav neiskorišteni CSS prilikom izrade za proizvodnju (eng. production build), što smanjuje broj neiskorištenih CSS klasa na nulu te tako štedi memoriju i olakšava snalaženje pri pisanju stilova.

Jedna od velikih prednosti Tailwinda je to da pruža enormnu količinu klasa koje se mogu primijeniti na HTML elemente. Ove klase često koriste kratke i opisne nazive kako bi se olakšalo stiliziranje elemenata. Na primjer, klasa *bg-blue-500* postavlja pozadinsku boju elemenata na svijetloplavu, dok klasa *text-xl* postavlja veličinu teksta na iznimno veliku. Fleksibilnost Tailwinda još je jedna od njegovih prednosti. Može se koristiti zajedno s drugim CSS okvirima ili pretprocesorima kao što je SASS. Tailwind se dakle ne nameće kao ekskluzivno rješenje te ne isključuje korištenje drugih načina stilizacije [8]. Primjer stranice za javljanje greške 404 stilizirane pomoću Tailwinda nalazi se na slici 2.5 gdje se može vidjeti kako su svi stilovi obrađeni isključivo preko naziva klasa.

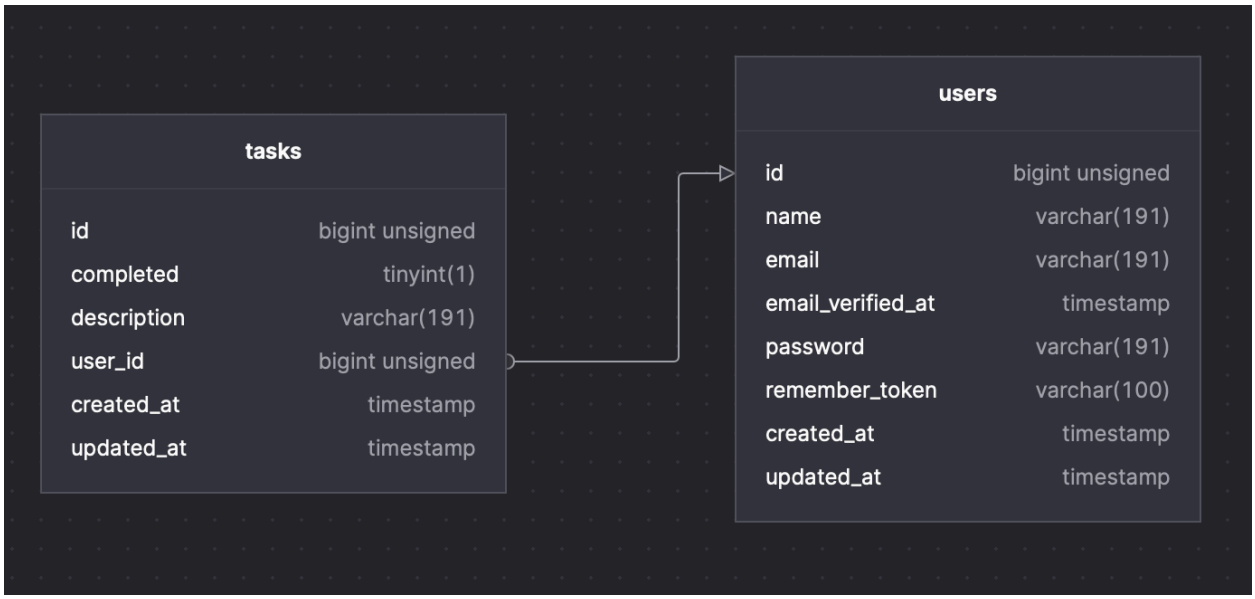


Slika 2.5 Primjer Tailwind komponente i njoj pripadnog koda [9]

2.5 MySQL

MySQL je sustav za upravljanje relacijskim bazama podataka (eng. ralational database management system) temeljen na SQL-u (*Structured Query Language).* MySQL ima široku primjenu, no najčešće se koristi u svrhu baze podataka za web aplikacije. Poznat je po svojoj pouzdanosti i performansama, što ga čini popularnim izborom za razvoj web aplikacija i platformi za upravljanje sadržajem (eng. Content Management System, skraćeno CMS) [10] . Često se koristi u kombinaciji sa web serverima kao što su Apache ili Nginx, i *server-side* skriptnim jezicima poput PHP-a, Pythona i Rubyja. To omogućava izradu dinamičkih web aplikacija koje mogu čitati, pohranjivati i upravljati podacima u bazi.

Entity Relationship (skraćeno E-R) dijagram baze podataka korištene za izradu To-Do aplikacije u ovome radu prikazan je na slici 2.6. Na njemu se može vidjeti kako u bazi postoje dva entiteta, a to su *'users'* i *'tasks'*. Entitet *'users'* sadrži atribute *'id', 'name', 'email', 'email_vertified_at', 'password', 'password_token', 'created_at'* i *'updated_at'*. Entitet *'tasks'* sadrži atribute *'id', 'completed', 'description', 'created_at', 'updated_at'* te strani ključ *'user_id'* koji pokazuje na ID korisnika kojemu pripada određeni zadatak.



Slika 2.6 E-R dijagram

2.6 Axios

Axios je JavaScript knjižnica koja se koristi za izradu HTTP zahtjeva od strane node.js-a ili XMLHttpRequesta iz preglednika. Node.js izvorni http modul koristi na strani poslužitelja, dok

na strani klijenta (u pregledniku) koristi XMLHttpRequests. Može se koristiti za presretanje HTTP zahtjeva i odgovora te omogućuje zaštitu na strani klijenta od XSRF-a [11]. Izomorfan je što znači da se može izvoditi u pregledniku i node.js-u s istom bazom koda. Jedna od glavnih prednosti Axiosa je njegova jednostavna sintaksa i jasna upotreba. Zahvaljujući Axios-u, izrada HTTP zahtjeva, kao što su GET, POST, PUT ili DELETE, postaje relativno jednostavna. Također podržava različite HTTP metode i omogućava prilagodbu zaglavlja zahtjeva. Primjer u kojemu je vidljivo kako je *get* zahtjev mnogo čitljiviji i jednostavniji koristeći Axios umjesto slanja istog zahtjeva Fetch API-jem vidljiv je na slici 2.7.

```
// Fetch API
fetch('https://api.example.com/tasks')
  .then((response) => {
    if (!response.ok) {
      throw new Error('Network response error');
    }
    return response.json();
  })
  .then((data) => {
    console.log(data);
  })
  .catch((error) => {
    console.error('Fetch error:', error);
  });
```

```
// Axios
axios.get('https://api.example.com/tasks')
  .then((response) => {
    console.log(response.data);
  })
  .catch((error) => {
    console.error('Axios error:', error);
  });
```

Slika 2.7 Primjer get zahtjeva bez i sa korištenjem Axios knjižnice

Još jedna značajka koja čini Axios izvrsnim rješenjem je njegova kompatibilnost sa svim modernim preglednicima i okolinama. To znači da se može koristiti kako u klijentskoj, tako i u serverskoj strani aplikacije. Axios također pruža podršku za otkrivanje i automatsko translatiranje odgovora u različite formate, uključujući JSON i XML. To olakšava rad s različitim vrstama API-ja. Axios se također može koristiti s modernim okvirom za upravljanje stanjem kao što je Vuex u Vue.js-u ili Redux u React-u. To omogućava integraciju HTTP komunikacije s globalnim stanjem aplikacije. Axios daje sposobnost za praćenje napretka HTTP zahtjeva i omogućavanje poništavanje zahtjeva ako je to potrebno. Ovo je korisno za izbjegavanje nepotrebnih zahtjeva i poboljšanje performansi aplikacije [12].

Činjenica je da je Axios moćan alat za izvođenje HTTP zahtjeva u JavaScript aplikacijama. Njegova jednostavna sintaksa, podrška za različite platforme i napredne značajke čine ga popularnim izborom među programerima za komunikaciju s web servisima i API-jima.

2.7 Node package manager

Za izradu ovog projekta, prethodno je bila potrebna instalacija Node.js-a. Postavljanjem Node.js-a omogućeno je korištenje zadanih alata za upravljanje paketima u okruženju Node.js, poznatih kao Node Package Manager (skraćeno NPM).

Node Package Manager sastoji se od tri komponente. Prva komponenta obuhvaća službene web stranice, gdje programeri mogu istraživati i pronalaziti nove pakete i resurse. Druga komponenta je korisničko sučelje putem naredbenog retka (CLI) koje omogućava interakciju s *npm-om*. Treća komponenta je NPM registar, koji sadrži brojne JavaScript pakete korisne za razvoj web aplikacija.

NPM je izvorno razvijen zajedno s Node.js-om i postao je standard za upravljanje paketima u JavaScript okruženju. Njegova funkcionalnost obuhvaća pretragu, instalaciju, ažuriranje i brisanje JavaScript paketa te je ključan za rad s vanjskim modulima i knjižnicama. Upravljanje projektima također je pojednostavljeno putem NPM-a. Svaki projekt može imati svoju *package.json* datoteku koja definira ovisnosti, skripte i konfiguraciju. Ovo omogućava programerima lako dijeljenje i reproduciranje okoline između razvojnih timova i računala. NPM također podržava inačice paketa i omogućava programerima da specificiraju točnu verziju paketa koja će se koristiti u projektu. Ovo je ključno za osiguravanje stabilnosti aplikacija i izbjegavanje neočekivanih problema sa zastarjelim verzijama paketa [13]. Može se zaključiti da je *Node Package Manager* neizostavan alat u JavaScript ekosustavu koji pojednostavljuje upravljanje ovisnostima, dijeljenje koda i automatizaciju razvoja. Njegova moćna funkcionalnost i aktivna zajednica čine ga krucijalnim dijelom modernog JavaScript razvoja.

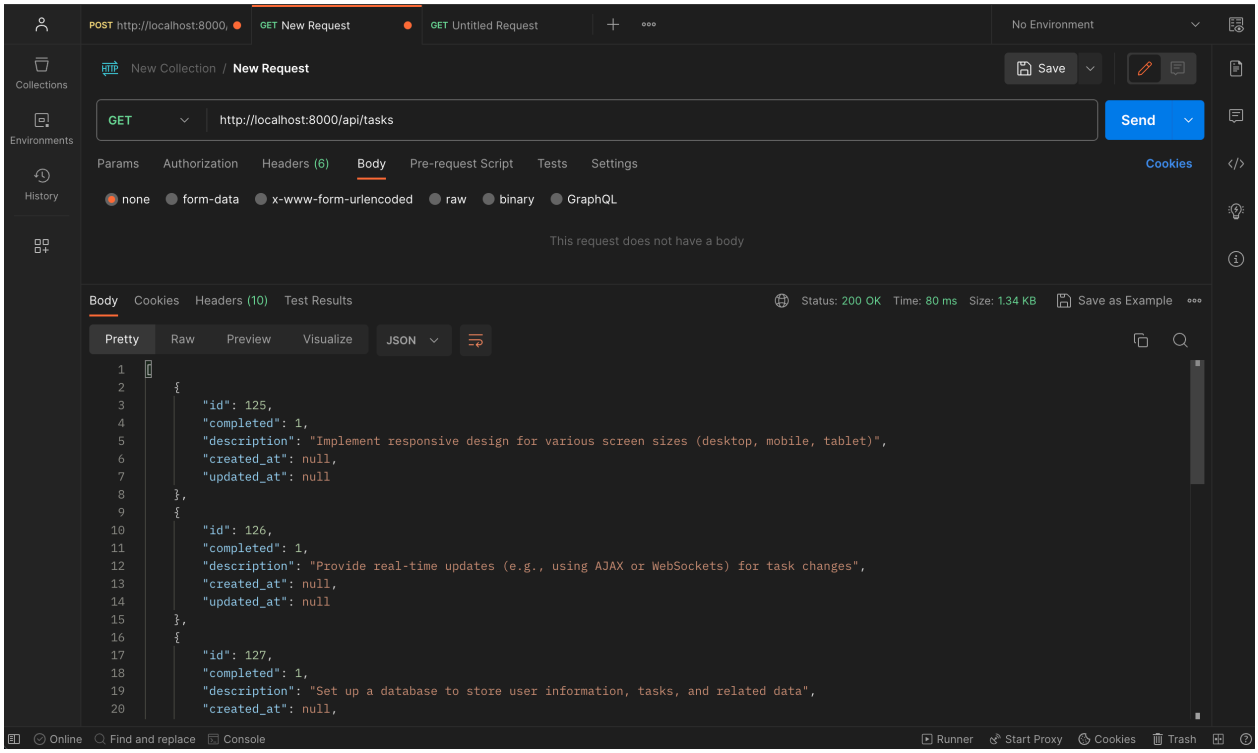
2.8 Postman

Postman je snažan i široko korišten alat za testiranje i upravljanje API-jima koji ima ključnu ulogu u razvoju i održavanju web aplikacija. Svojim intuitivnim sučeljem i raznovrsnim značajkama, olakšava programerima interakciju s API-jima, učinkovito dijagnosticiranje problema i samim time poboljšavanje kvalitete softverskih proizvoda.

Jedna od glavnih karakteristika Postmana je mogućnost izrade HTTP zahtjeva prema različitim API-jima i resursima. Ovo omogućava programerima da testiraju funkcionalnost, provjere odgovore i analiziraju ponašanje API-ja, što je ključno u razvoju aplikacija kao i u otklanjanju pogrešaka u istima. Postman podržava različite vrste HTTP zahtjeva, uključujući GET, POST, PUT

i DELETE, te omogućava dodavanje parametara, tijela zahtjeva i zaglavlja kako bi se precizno simulirali stvarni scenariji uporabe. Također podržava autentifikaciju kako bi se moglo pristupiti zaštićenim resursima [14].

Značajke poput automatskog generiranja dokumentacije, praćenja promjena i integracije s popularnim alatima za upravljanje verzijama čine Postman sveobuhvatnim alatom za razvoj i testiranje API-ja. S njegovim rastućim ekosustavom i aktivnom zajednicom korisnika, Postman nastavlja biti ključan alat u modernom softverskom inženjeringu. Prikaz alata Postman pri slanju HTTP zahtjeva na poslužiteljsku stranu pri testiranju To-Do aplikacije za ovaj rad vidljiv je na slici 2.8.



Slika 2.8 Alat Postman

2.9 Prikaz funkcionalnosti kroz korištene tehnologije

Radi lakšeg shvaćanja korištenih tehnologija u ovome radu, na primjeru spremanja imena i prezimena korisnika u bazu podataka, biti će prikazan životni ciklus jedne funkcionalnosti.

Pri odlasku na web aplikaciju u svrhu kreiranja korisnika prikazuje se početni pogled naziva „*Register*“ koji postoji u obliku komponente namijenjen za registraciju korisnika. Sadržaj tog pogleda napisan je u HTML jeziku te je smješten unutar komponente radnog okvira u za to

predviđeno mjesto, ukoliko se radi o Angularu onda je to datoteka s proširenjem *.html*, a ukoliko se radi o Vue.js-u onda je to datoteka s proširenjem *.vue* namijenjena za izgradnju komponente. Obje datoteke smještene su u *src* direktorij unutar strukture radnog okvira.

Pogled o kojemu je riječ u oba je slučaja stiliziran koristeći Tailwind i još pokoju dodatnu CSS klasu. Tailwind je primijenjen tako što su dodani prikladni nazivi klasa unutar sadržaja te se na temelju toga automatski generira njegov stil. U slučaju Angulara, dodatne CSS klase smještene su u datoteku s proširenjem *.css* unutar direktorija same komponente dok Vue.js te stilove dohvaća iz klasa napisanih u oznaci <*style*> koja se nalazi unutar istog dokumenta kao i sadržaj pogleda.

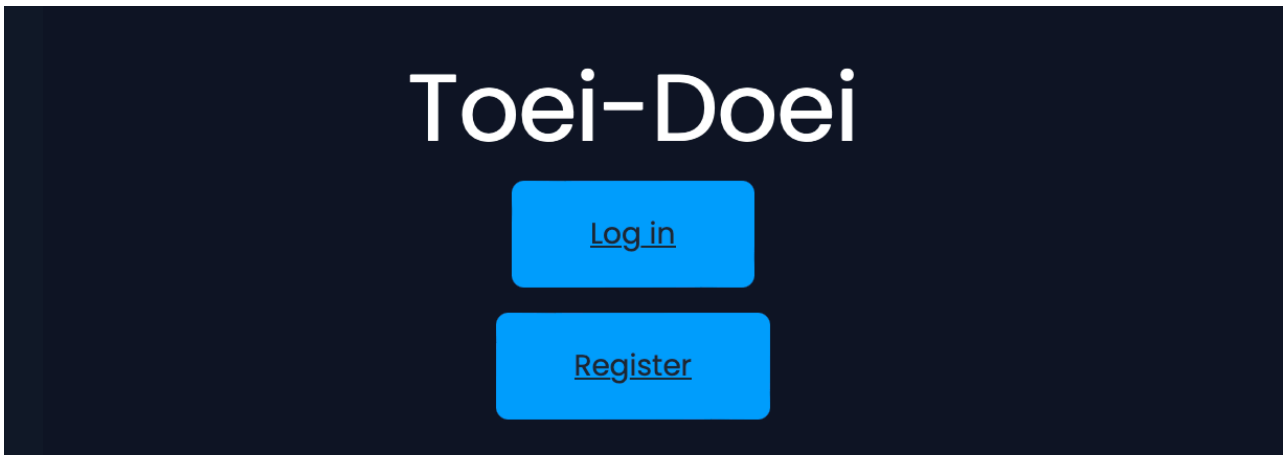
Taj pogled u svome sadržaju poziva drugu komponentu „*Form*“ koja u sebi implementira formu sa četiri polja za unos i jednom tipkom za podnošenje forme. Kada korisnik ispuni formu i klikne tipku za njeno podnošenje, logika napisana u komponenti pozvati će funkciju koja šalje zahtjev na stranu poslužitelja, u Angularu pomoću sadržane knjižnice za upravljanje zahtjevima, a u Vue.js pomoću knjižnice Axios. Ta funkcija u Angularu biti će napisana u datoteci s proširenjem *.ts* dok će kod Vue.js-a biti unutar iste datoteke kao i sadržaj i stil. Zatim *endpoint* u Laravelu dohvaća taj zahtjev te ga obrađuje u skladu s ponašanjem definiranim u za to predviđenom kontroleru. Nakon toga sprema podatke iz zahtjeva u bazu pomoću MySQL-a te samim time kreira novog korisnika. Nakon uspješnog unosa u bazu, funkcija iz kontrolera u Laravelu vraća odgovor sa statusom 201 te time potvrđuje dodavanje novog korisnika u bazu.

3 OPIS APLIKACIJE

U svrhu ovog rada razvijena je To-Do aplikacija s implementiranim funkcionalnostima autentifikacije te kreiranja, dohvaćanja, uređivanja i brisanja zadataka. Aplikacija je razvijena u oba radna okvira, Angular i Vue.js te izgleda identično. Iz tog razloga, u ovome poglavlju primjeri će biti odrađeni na slikama aplikacije iz jednog od radnih okvira.

3.1 Početni pogled

Pri prvom učitavanju aplikacije prikazuje se početni pogled koji korisniku pruža dvije opcije u obliku tipki: *Login* i *Register (Slika3.1)*. Sam početni pogled napravljen je u obliku komponente koja je rutama povezana s ostalim pogledima, također komponentama.



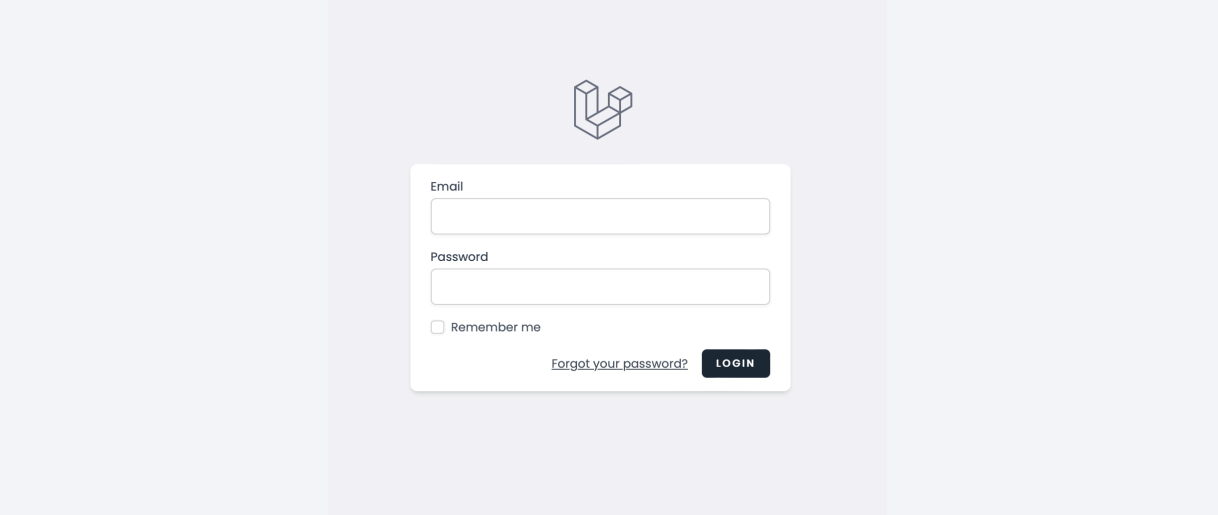
Slika 3.1 Početni pogled

3.2 Pogledi autentifikacije

Nadalje su prikazani pogledi koji se prikazuju u procesu autentifikacije odnosno prijave i registracije.

3.2.1 Pogled za prijavu

Klikom na prvu opciju (*Log in*) početnog pogleda otvara se pogled koji prikazuje slika 3.2 gdje je korisniku dostupna forma za unos podataka potrebnih pri autentifikaciji odnosno *Email* i *Password* te tipka za podnošenje te iste forme.

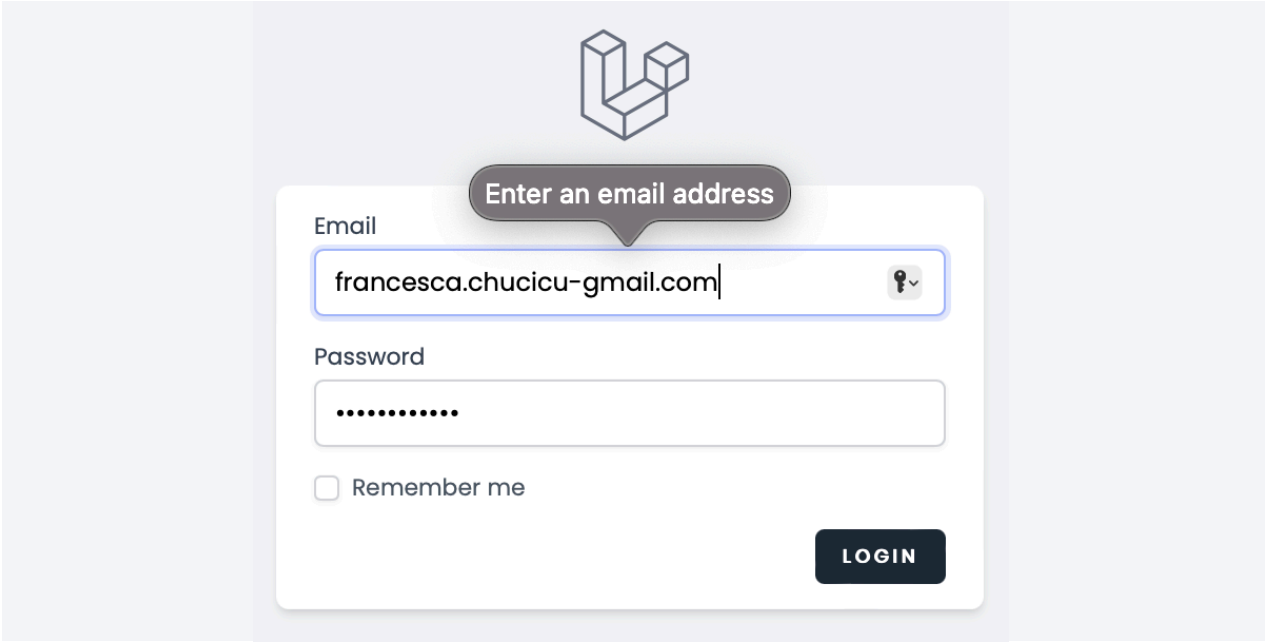


Slika 3.2 Pogled za prijavu korisnika

Ukoliko korisnik klikne tipku *login*, a nije ispunio jedno od polja u formi, sučelje javlja grešku u obliku upozorenja koje označava korisniku da je potrebno ispuniti oba polja prije podnošenja forme. Prikaz tog *upozorenja* vidljiv je na slici 3.3.

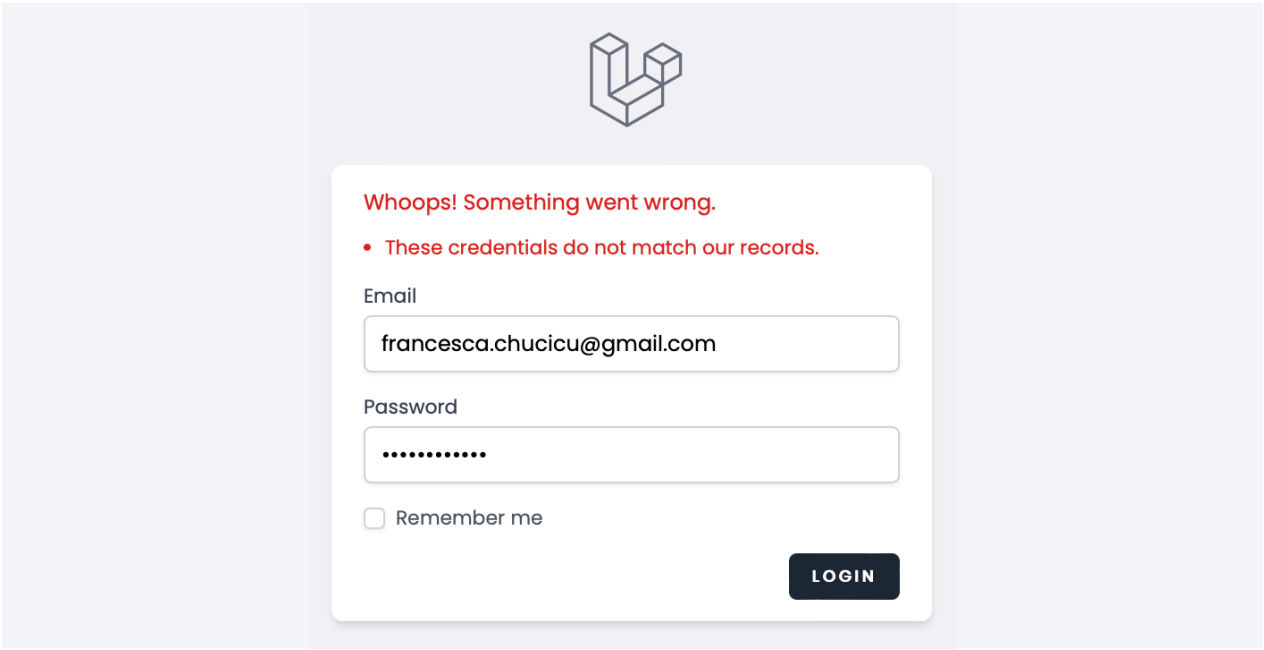
Slika 3.3 Polje u formi ostavljeno prazno

Ukoliko korisnik ispuni formu, ali ne unese email adresu u pravome obliku, sučelje će izbaciti upozorenje da je potrebno ubaciti email adresu što je vidljivo na slici 3.4.



Slika 3.4 Unesena email adresa u krivom obliku

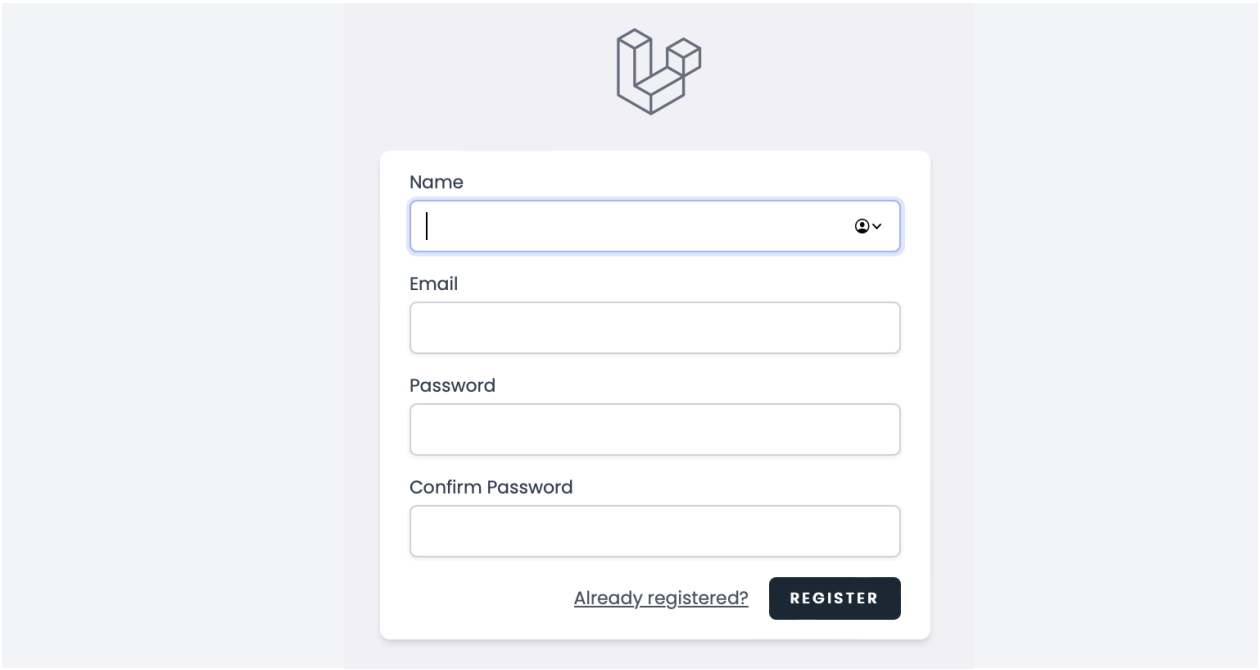
Ukoliko korisnik unese email adresu u pravome obliku i ispuni formu, poslati će se zahtjev za prijavu. Ukoliko uneseni podaci nisu pronađeni u bazi podataka odnosno ne odgovaraju niti jednom stvorenom *useru*, oni se smatraju netočnima te sučelje ispisuje pogrešku iznad forme koju prikazuje slika 3.5.



Slika 3.5 Pogreška pri pokušaju autentifikacije

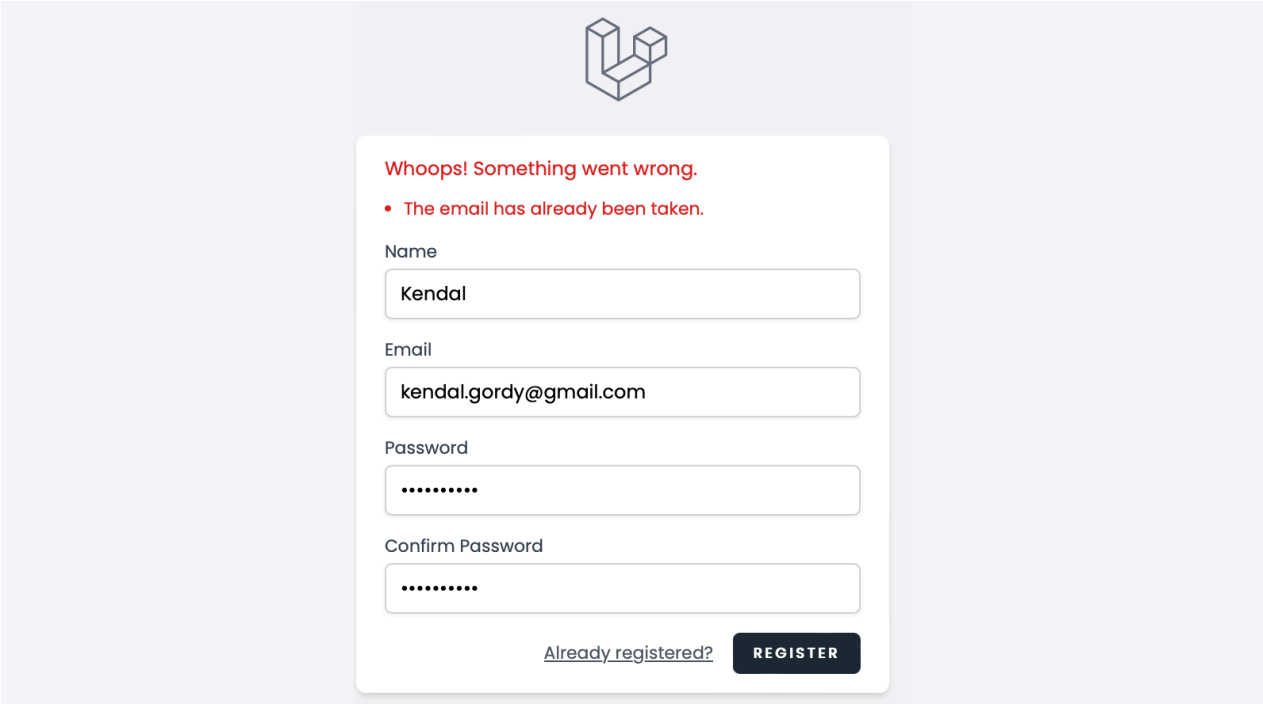
3.3 Pogled za registraciju

Klikom na drugu opciju (*Register*) početnog pogleda otvara se pogled koji prikazuje slika 3.6 gdje je korisniku dostupna forma za samu registraciju odnosno kreiranje računa. Prilikom dolaska na pogled za registraciju korisniku se prikazuje forma sa četiri polja za kreiranje računa. Iznad svakog polja piše oznaka koja opisuje što je potrebno unijeti u to polje. Kako bi korisnik kreirano novi račun potrebno je ispuniti sva polja u skladu sa pravilima opisanima u nastavku koja vrijede za svako od njih. Na dnu forme postoji buton REGISTER na čiji se klik forma, ukoliko je ispravno ispunjena, šalje putem HTTP zahtjeva na poslužitelja.



Slika 3.6 Pogled za autentifikaciju

Ukoliko korisnik pokuša kreirati račun sa email adresom koja je već zauzeta od strane nekog drugog korisnika, sučelje će nakon što dobije odgovor od poslužitelja da je adresa zauzeta javiti prikladno upozorenje koje prikazuje slika 3.7.



Slika 3.7 Email adresa se već koristi

Kao i kod prijave, u polje za unos email adrese potrebno je unijeti adresu oblika *tekst@tekst* kako sučelje ne bi javilo grešku. Isto kao i polje za email adresu, polja za unos zaporki imaju uvjete validacije. Ukoliko korisnik unese zaporku kraću od osam znakova, sučelje će javiti prikladnu pogrešku.

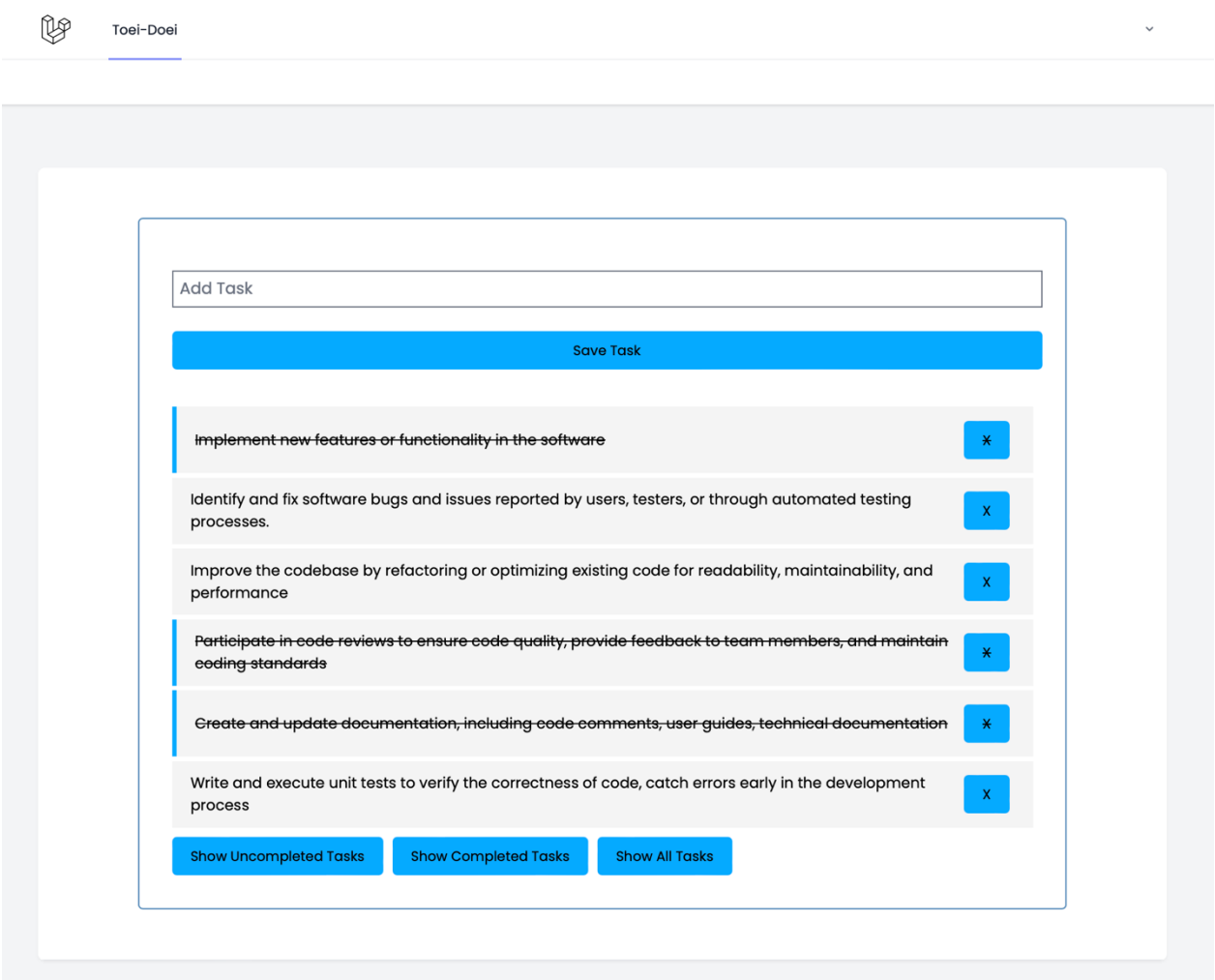
3.4 Pogledi liste zadataka (Dashboard)

U slučaju da je forma ispunjena u skladu sa pravilima svakog polja za unos te da su uneseni podatci jedinstveni i ispravnog oblika, kreira se novi korisnik, sprema se u bazu podataka te se otvara novi pogled *Dashboard* prikazan na slici 3.8.

Pri vrhu u lijevom kutu prikazani su Laravelov logotip i naziv aplikacije odnosno *Toei-Doei* te klik na bilo koji od ta dva elementa vodi natrag na početni pogled. Pri gornjem desnom vrhu nalazi se strelica koja pokazuje prema dolje te se klikom na nju otvara padajući izbornik s opcijom *Logout* na čiji se klik korisnik odvodi na početni pogled. Sljedeće što se nalazi na ovome pogledu je okvir unutar kojega se prikazuje polje za unos novog zadatka te tipka za podnošenje tj. spremanje novog zadatka. U polju za unos zadatka nalazi se *placeholder „Add Task“*.

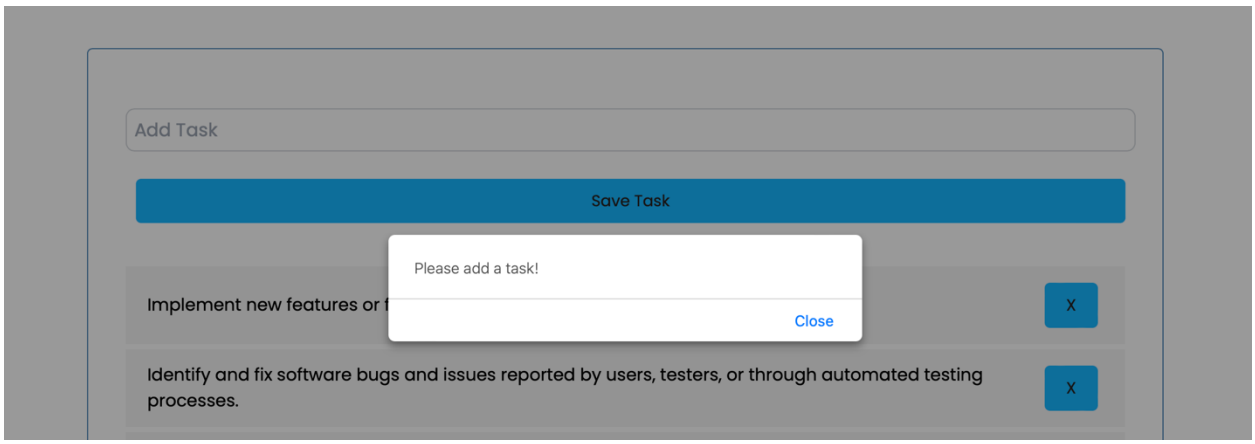
Ispod toga prikazuje se lista svih zadataka specifična za trenutno ulogiranog korisnika. Korisnik ovdje može manipulirati svojim zadacima odnosno može ih brisati ili označiti kao završene. U

polju zadatka nalaze se tekst zadatka i tipka za njegovo brisanje. Na dnu liste zadataka nalaze se tri tipke, redom *Show Uncompleted tasks, Show Completed Tasks* i *Show All Tasks.* Sve navedene operacije upravljanja zadacima obavljaju se bez osvježavanja ekrana.



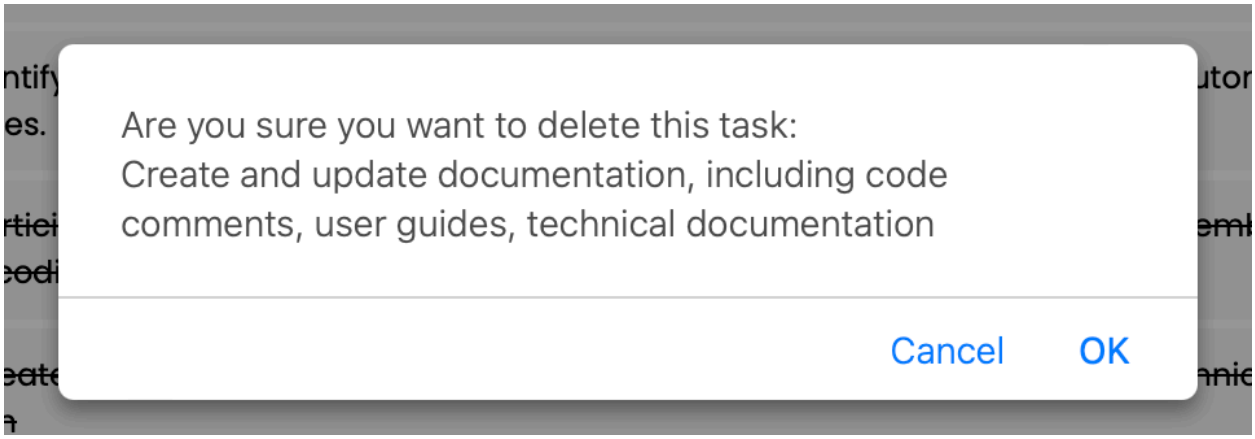
Slika 3.8 Pogled Dashboard

Dodavanje zadataka u listu korisnik izvršava tako što u polje za unos *„Add task“* upisuje željeni opis zadatka te pritišće tipku *Save task* ili tipku *Enter* na tipkovnici. Zadatak se zatim sprema u bazu podataka te se automatski prikazuje korisniku u njegovoj listi zadataka. Ukoliko je tipka za spremanje zadatka pritisnuta, a u polje za opis zadatka nije upisano ništa, sučelje javlja upozorenje u obliku *alerta* koje govori korisniku da ispuni polje. To upozorenje sadrži opciju *Close* te se pritiskom na nju *alert zatvara.* Navedeno upozorenje prikazano je slikom 3.9.



Slika 3.9 Upozorenje za ostavljeno prazno polje opisa zadatka

Ukoliko želi izbrisati zadatak sa svoje liste, korisnik klikće buton označen sa „X“ u produžetku zadatka kojeg želi obrisati. Zatim iskače prozor za potvrdu vidljiv na slici 3.10. koji upita korisnika je li siguran da želi obrisati zadatak kako ga ne bi obrisao slučajno te mu prikazuje i opis zadatka na kojeg je kliknuo da ga obriše. Ukoliko korisnik pritisne OK sučelje potom šalje DELETE zahtjev na poslužitelja te se zadatak pritom briše iz baze podataka i ujedno makne sa korisnikove liste zadataka.

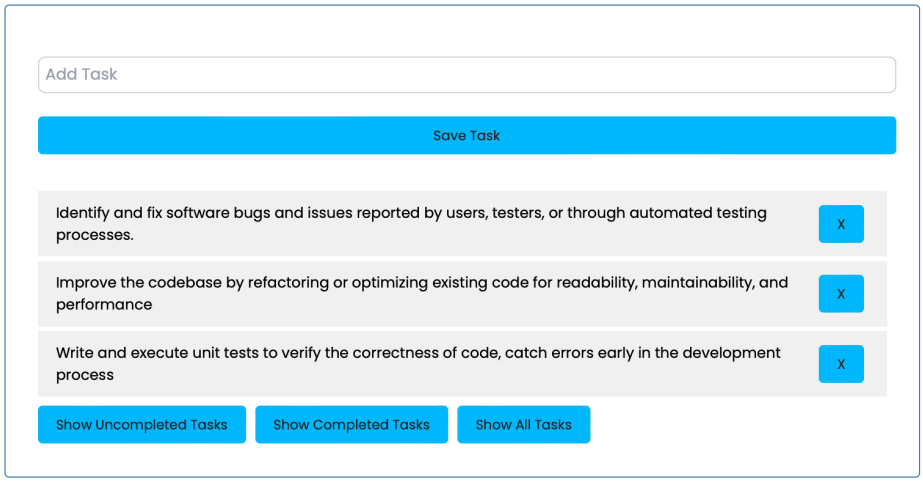


Slika 3.10 Potvrdni prozor pri brisanju zadataka

Korisnik ima mogućnost i označiti zadatak kao završen. To radi dvostrukim klikom na polje zadatka te se, nakon uspješnog zahtjeva sučelja i povratka odgovarajućeg odgovora od poslužitelja, tekst zadatka precrta cijelom svojom duljinom i na lijevom kraju polja zadatka pojavi se obrub kako bi korisnik lakše razlikovao završene zadatke od nezavršenih. Također, dvostrukim klikom na zadatak koji je već označen kao završen, taj zadatak ponovno postaje nezavršen te poprima izgled nezavršenog zadatka. Prikaz nezavršenog zadatka te istog tog zadatka označenog kao završen vidljiv je na slici 3.11.

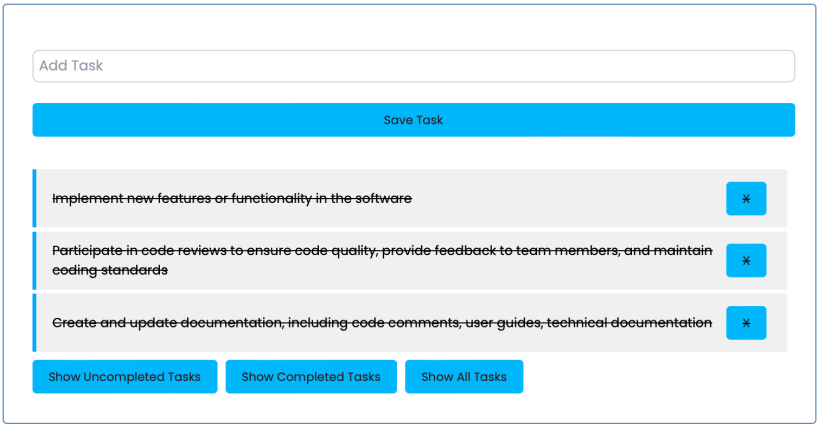
Slika 3.11 Prikaz zadatka prije (gore) i nakon (dolje) što je označen kao završen

Klikom na tipku *Show Uncompleted Tasks* iz liste zadataka filtriraju se zadaci označeni kao nezavršeni te se samo oni prikazuju na ekranu kao što je vidljivo na slici 3.12.



Slika 3.12 Prikaz liste nezavršenih zadataka

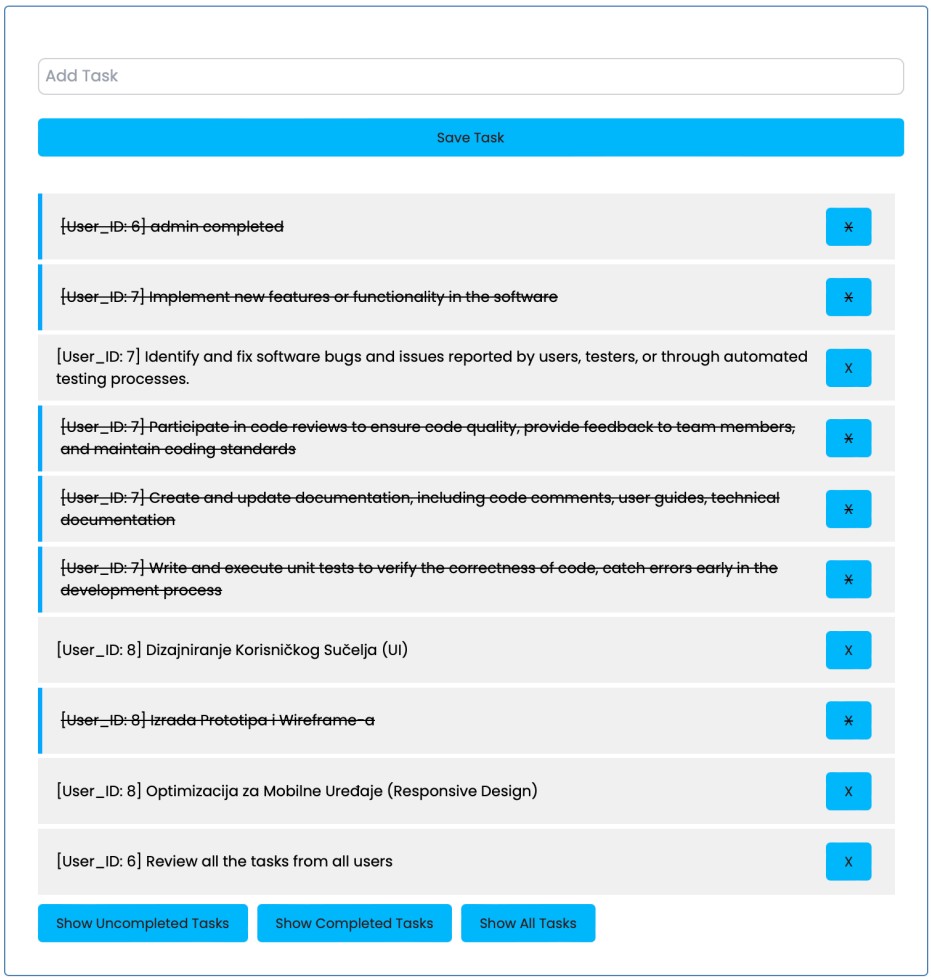
Pritiskom na sljedeću tipku označenu sa *„Show Completed Tasks“* iz liste svih zadataka izdvajaju se i prikazuju samo oni koji su označeni kao završeni, a taj je slučaj prikazan na slici 3.13.



Slika 3.13 Prikaz liste završenih zadataka

3.4.1 Pogled liste zadataka administratora

Ukoliko se prijavi korisnik kojemu je dodijeljena oznaka administratora, na listi prikaza zadataka biti će vidljivi zadaci svih korisnika te će prije opisa zadataka biti vidljiv ID korisnika kojemu pripada taj zadatak. Administrator ima iste mogućnosti upravljanja zadacima kao i korisnici, no razlika je u tome što on ima pristup i mogućnost upravljanja svojim, ali i brisanja te dohvaćanja tuđih zadataka. Prikaz kako administrator vidi listu zadataka vidljiv je na slici 3.14.



Slika 3.14 Administratorov prikaz liste zadataka

4 USPOREDBA RADNIH OKVIRA ANGULAR I VUE.JS

Angular i Vue.js, radni su okviri za izradu web aplikacija, najčešće jednostraničnih. Iako dijele mnoge sličnosti, ovi radni okviri ipak imaju svoje specifičnosti koje su katkad presudne pri odabiru jednog od njih za izradu projekata i aplikacija. Kako bi bilo lakše uvidjeti specifičnosti za svaki radni okvir, potrebno je prvo sagledati njihove sličnosti koje su sljedeće:

a) Komponentna arhitektura: Oba okvira podržavaju komponentnu arhitekturu, što znači da aplikaciju grade s više manjih, samostalnih komponenti koje se mogu ponovno koristiti
b) Reaktivnost: I Angular i Vue.js koriste reaktivni pristup, što znači da automatski reagiraju na promjene u sadržaju te osvježavaju korisničko sučelje
c) Razvojni alati: Oba okvira dolaze s bogatim setom razvojnih alata koji pomažu programerima pri izradi i održavanju aplikacija
d) *Routing*: Angular i Vue.js imaju gotova rješenja za upravljanje rutama unutar aplikacije (Angular Router i Vue Router), što uvelike olakšava navigaciju između različitih dijelova aplikacije
e) Prilagodljivost: Oba okvira omogućuju prilagodbu pomoću dodataka i *pluginova* [15]

Kada je riječ o njihovim razlikama, valja govoriti o komponentama, složenosti i krivulji učenja, DOM-u, zahtjevima na API, performansama te zajednici i ekosustavu. U nastavku slijedi pregled razlika radnih okvira Angular i Vue.

4.1 Komponente

I Angular i Vue.js koriste komponente kao osnovne gradivne jedinice za strukturiranje svojih aplikacija. Međutim, sama struktura tih komponenti ima svoje razlike u ova dva radna okvira. Naime, komponenta u Angularu sastoji se od četiri datoteke, a njihova proširenja zajedno sa svrhama su sljedeća:

a) <ime-komponente>.*component.html* – HTML predložak koji deklarira što se prikazuje na stranici

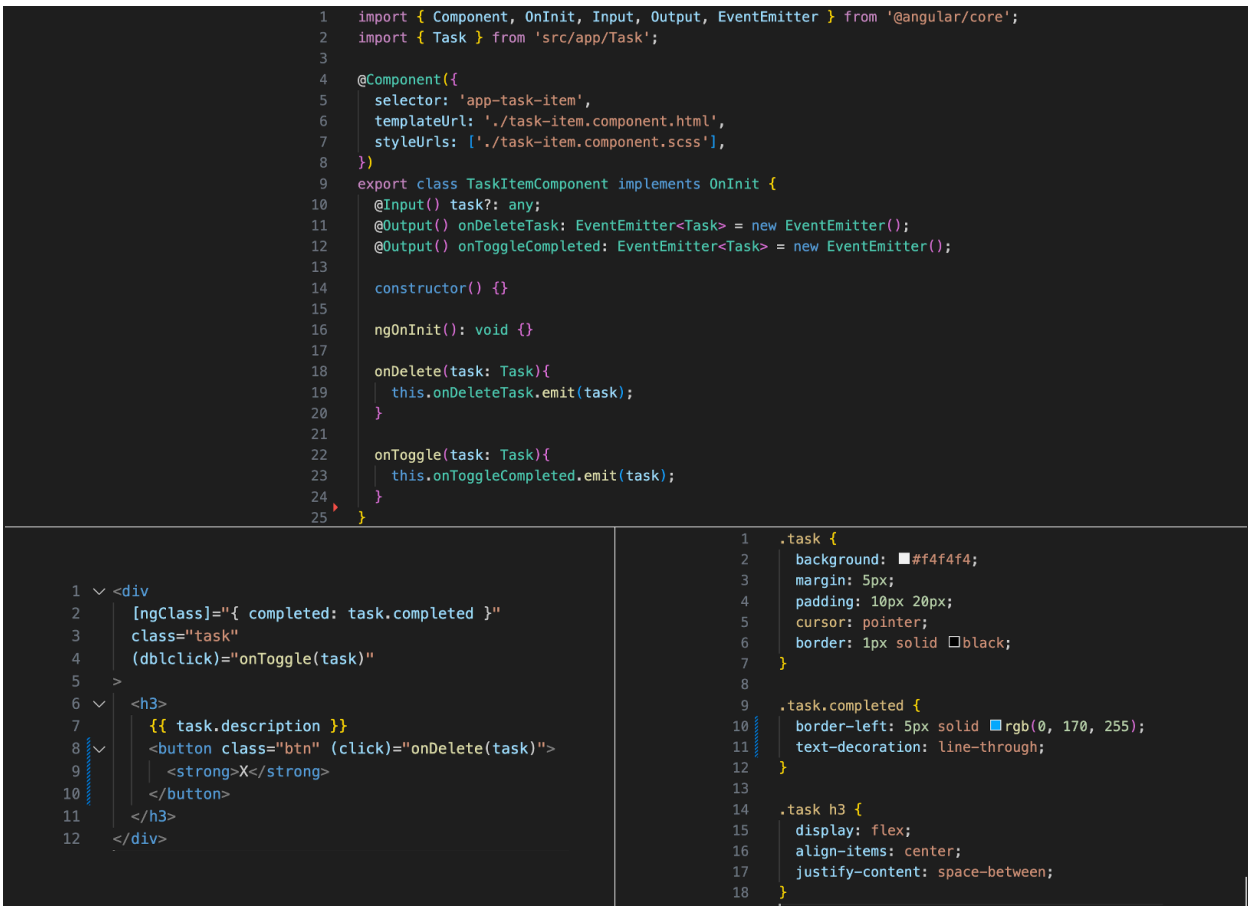
b) <ime-komponente>.*component.css* – CSS selektor koji definira stilove komponente
c) <ime-komponente>.*component.spec.ts* – Datoteka za specifikaciju testiranja komponente
d) <ime-komponente>.*component.ts* – TypeScript klasa koja definira ponašanje odnosno logiku korištenu u komponenti

Kako bi Angular znao da je riječ o komponenti, iznad definicije klase u *.ts* datoteci potrebno je dodati *@Component* anotaciju. Time ona dobiva posebna svojstva, a unutar nje definiraju se i ostale postavke komponente. Ovdje je definiran *selector* – jedna od najvažnijih postavki koja označava kako će se unutar HTML-a pozivati komponenta. Zatim postoji atribut *templateUrl* koji omogućuje navođenje putanje do eksternog HTML predloška koji će se koristiti za prikazivanje komponente. Posljednje se definira *styleUrls* gdje su zapisane putanje koje pokazuju na stilove koji će se koristiti u ovoj komponenti. Definicija komponente prikazana je na slikom 4.1.

```
4  @Component({
5    selector: 'app-task-item',
6    templateUrl: './task-item.component.html',
7    styleUrls: ['./task-item.component.scss'],
8  })
```

Slika 4.1 Definicija komponente u Angularu

Na slici 4.2. prikazan je sadržaj datoteka komponente koja predstavlja jedan zadatak u listi zadataka. Vrijedi uočiti kako su dijelovi komponente odvojeni u zasebne datoteke, ali svaki dio međusobno komunicira i poziva jedan drugoga, no bez potrebe da se dodatno uveze kao što se trebaju uvesti druge komponente ili knjižnice pomoću ključne riječi *import*.



Slika 4.2 Komponenta u radnom okviru Angular

Za razliku od Angulara, ista ova komponenta u radnom okviru Vue sadržana je u samo jednoj datoteci s proširenjem *.vue* prikazanoj na slici 4.3. Vidi se kako su predložak, logika i stil smješteni u oznake *template*, *script*, i *style*.

```vue
<template>
    <div :class="[task.completed ? 'task completed' : 'task']">
      <h3 @dblclick="toggleCompleted">
        {{ task.description }}
        <button class="btn" @click="deleteTask()">X</button>
      </h3>
    </div>
</template>

  <script>
    import axios from 'axios';

    export default {
        name: 'task-item',
        props: ['task'],
        methods: {
          toggleCompleted(){
              axios.put('http://localhost:8000/api/toggleCompleted/' + this.task.id, {
                task: this.task }).then(response => {
                if(response.status == 200) {
                  this.$emit('onToggle') }})
              .catch(error => {
                console.log(error) })
            return },
          deleteTask(){
            axios.delete('http://localhost:8000/api/delete/' + this.task.id, {
              task: this.task })
            .then(response => {
              this.$emit('onDelete')})}}
    }
  </script>

<style>
  .task {
    background: #f4f4f4;
    margin: 5px;
    padding: 10px 20px;
    cursor: pointer;
  }
  .task.completed {
      border-left: 5px solid rgb(0, 170, 255);
      text-decoration: line-through;
  }
  .task h3 {
      display: flex;
      align-items: center;
      justify-content: space-between;
  }
</style>
```

Slika 4.3 Komponenta u radnom okviru Vue.js

Može se zaključiti kako komponente u Angularu i Vue.js-u, makar izvršavale istu funkciju, ipak imaju razlike u svojoj strukturi i načinu rada. Na prvi pogled, moglo bi se reći da je struktura komponente u Vue.js-u puno preglednija radi toga što se svi njeni dijelovi nalaze u istoj datoteci, međutim to može postati veoma kaotično ukoliko veličina komponente poraste i uz veću količinu

sadržaja krene implementirati veliki broj metoda, atributa i stilova. Valja uzeti u obzir to da bi komponenta trebala biti i ostati osnovna gradivna jedinica aplikacije i kao takva trebala bi implementirati što je moguće uži raspon funkcionalnosti kako bi se olakšalo njezino ponovno korištenje na drugim mjestima unutar aplikacije i time zadržao smisao korištenja komponenti.

Kada je riječ o slanju podataka roditelja djetetu, Angular koristi anotaciju *@Input* unutar komponente koja je dijete. Slika 4.4 prikazuje korištenu anotaciju u svrhu primanja sučelja *task* u komponenti *task-item*.

```
4   @Component({
5     selector: 'app-task-item',
6     templateUrl: './task-item.component.html',
7     styleUrls: ['./task-item.component.scss'],
8   })
9   export class TaskItemComponent implements OnInit {
10
11    @Input() task?: any;
12
13
14    @Output() onDeleteTask: EventEmitter<Task> = new EventEmitter();
15    @Output() onToggleCompleted: EventEmitter<Task> = new EventEmitter();
```

Slika 4.4 Korištenje @Input i @Output anotacije u Angularu

Način na koji roditeljska komponenta šalje to isto sučelje *task* svojem djetetu, komponenti *task-item*, prikazan je sljedećom linijom koda na slici 4.5. Ta linija koda dodaje se pri pozivu komponente djeteta unutar oznaka selektora za pozivanje njezinog predloška.

```
6   [task]="task"
```

Slika 4.5 Slanje sučelja task u komponentu djeteta

U aplikaciji To-Do, nakon što korisnik pritisne tipku za brisanje zadatka, točno taj zadatak treba se poslati nazad u komponentu roditelja kako bi se ondje izvršila funkcija za brisanje zadatka odnosno slanje DELETE zahtjeva na poslužitelja. Način na koji funkcija dijete šalje konkretni *task* roditelju opisan je u nastavku.

Kao i kod primanja podataka, za slanje podataka koristi se određena anotacija, no ovoga puta *@Output* anotacija kao što je to prikazuje slika 4.4. U HTML dijelu komponente *task-item* pri

kliku na buton za brisanje zadatka, pomoću koda sa slike 4.6, poziva se funkcija *onDelete(task)* koja prima *task* koji se treba obrisati.

```
<button class="btn" (click)="onDelete(task)">X</button>
```

Slika 4.6 Funkcija koja se poziva klikom na buton za brisanje zadataka u Angularu

Ta je funkcija definirana na slici 4.7 te je njena zadaća da emitira poslani *task* natrag u komponentu roditelja.

```
21  onDelete(task: Task){
22    this.onDeleteTask.emit(task);
23  }
```

Slika 4.7 onDelete(task) funkcija

Roditeljska komponenta prima poslani događaj (eng. event) koristeći kod sa slike 4.8.

```
5   (onDeleteTask)="deleteTask(task)"
```

Slika 4.8 Primanje eventa u roditeljskoj komponenti u Angularu

U prvom dijelu ove linije napisan je naziv *eventa* kojeg prima, dok je u drugom dijelu navedeno što će se desiti pri primanju tog *eventa*, a u ovom slučaju pozvati će se funkcija *deleteTask(task)* koja prima *task* poslan u *eventu*.

Komunikacija roditelja i djeteta u kontekstu komponenti, kod radnog okvira Vue.js slična je kao i u Angularu, ali ipak malo jednostavnija. Tipka za brisanje također na klik poziva funkciju za emitiranje događaja prikazanu slikom 4.9, međutim u Vue.js sučelje za *task* nije primano putem anotacije *@Input,* već se svi podaci koje komponenta prima spremaju u tzv. rekvizite (eng. props). Vue.js komponente zahtijevaju eksplicitnu deklaraciju rekvizita kako bi Vue.js znao koje vanjske podatke proslijeđene komponenti treba tretirati kao prolazne atribute.

```
<button class="btn" @click="deleteTask(task)">X</button>
```

Slika 4.9 Funkcija koja se poziva klikom na buton za brisanje zadataka u Vue.js-u

Nakon ovoga, funkcija *deleteTask(task)* šalje *emit* roditeljskoj komponenti kodom sa slike 4.10.

```
30          deleteTask(){
31            this.$emit('onDelete', task)
32          }
```

Slika 4.10 Funkcija koju poziva klik na buton za brisanje u Vue.js-u

Roditeljska komponenta *tasks* zatim prima *event* pomoću koda na slici 4.11.

```
7           v-on:onDelete="deleteTask(task)">
```

Slika 4.11 Primanje eventa u roditeljskoj komponenti u Vue.js-u

Kod za objašnjenu funkcionalnost brisanja zadatka iz liste zadataka, u Angularu je rasprostranjen na šest datoteka dok je u Vue za to potrebno samo dvije. Angular za međusobnu komunikaciju komponenti uvodi anotacije i *event emittere* dok Vue.js sve odrađuje bez toga.

4.2 Složenost i krivulja učenja

Angular je poznat po svojoj složenosti. Radi se o sveobuhvatnom okviru sa strmom krivuljom učenja, posebno za programere koji su novi u razvoju web aplikacija ili imaju pozadinu rada u JavaScriptu. Nameće korištenje kompleksnije strukture te katkad zahtjevnih konvencija, što može biti korisno za velike i kompleksne projekte, ali za manje aplikacije često zna biti jednostavno previše.

U slučaju To-Do aplikacije izrađenu za ovaj rad, Angular se pokazao kao nepotrebno kompleksan za izradu tako jednostavnih komponenti s malim brojem funkcionalnosti. Ponajviše radi same strukture komponenti, no i pri komunikaciji između njih. Najveće prepreke pri učenju Angulara nalaze se na samom početku učenja gdje treba razumjeti koncepte kao što su poveznica između TypeScript klase i HTML predloška ili način korištenja i nazive odnosno sintaksu direktiva koje se koriste i jedinstvene su za Angular.

S druge strane, Vue.js je dizajniran da bude pristupačniji i lakši za učenje, što ga čini izvrsnim izborom kako za početnike tako i za iskusne programere. Pruža fleksibilniji i manje strog pristup,

omogućujući programerima postupno usvajanje i integraciju u postojeće projekte bez strmih krivulja učenja [16].

4.3 Document Object Model (DOM)

Razlika između Angulara i Vue.js-a kada je riječ o DOM-u odnosi se na njihov pristup upravljanju i manipulaciji dokumentnim objektima u web aplikacijama. Angular koristi svoj prilagođeni virtualni DOM (VDOM) sustav koji koristi tzv. *Angular Change Detection* proces. Ovaj sustav prati promjene u komponentama te automatski ažurira DOM kako bi odražavao te promjene. *Angular Change Detection* radi asinkrono i automatski detektira promjene u aplikaciji, omogućavajući programeru da se manje brine o ručnom ažuriranju DOM-a. U Angularu, rijetko je potrebna direktna manipulacija DOM-om. Umjesto toga, definiraju se komponente i koriste Angularove direktive za upravljanje tim komponentama i vezama s DOM-om. To omogućuje Angularu vrlo učinkovito ažuriranje DOM-a i održavanje konzistentnosti između stanja komponenti i prikaza na ekranu. [17]

Vue.js također koristi virtualni DOM (VDOM), ali pristup je nešto drugačiji u usporedbi s Angularom. U Vue.js-u, programeri često definiraju komponente koje uključuju HTML predloške (*template*) i JavaScript logiku. Vue.js automatski generira virtualni DOM za svaku komponentu, a zatim ga uspoređuje sa stvarnim DOM-om kako bi utvrdio promjene koje treba primijeniti. Vue.js također omogućuje programeru da ručno manipulira DOM-om kada je to potrebno putem svojih posebnih direktiva i referenci na elemente (*refs*). To pruža veću fleksibilnost za specifične slučajeve u kojima je potrebna izravna interakcija s DOM-om. U usporedbi s Angularom, Vue.js je često percipiran kao jednostavniji za upotrebu kada je riječ o manipulaciji DOM-om, što ga čini popularnim među programerima koji preferiraju više kontrola nad DOM-om u svojim komponentama.

4.4 Performanse

Na performanse u web aplikacijama mogu utjecati razni faktori. Angular i Vue.js oboje omogućavaju izgradnju aplikacija visokih performansi, ali postoje razlike u tome kako se nose s određenim aspektima koji mogu utjecati na performanse.

Kada je riječ o veličini snopova (eng. bundle size), Angular aplikacije obično imaju veće početne veličine snopova u usporedbi s Vue.js zbog svog bogatog skupa značajki i alata. To dovodi do

dužih vremena početnog učitavanja, posebno na sporim mrežnim vezama. Vue.js ima manju početnu veličinu snopova, što ga čini bržim za učitavanje, posebno za manje aplikacije. Vue 3 je također uveo tzv. *tree-shaking* odnosno eliminaciju neiskorištenog koda i bolju optimizaciju za manje veličine snopova.

Govoreći o performansama važno je napomenuti i način detekcije promjena (eng. change detection). Angular koristi složeniji mehanizam detekcije promjena na temelju zona (eng. zones). To može dovesti do dodatnog opterećenja u praćenju promjena, osobito u velikim aplikacijama s čestim ažuriranjima korisničkog sučelja. Vue.js koristi virtualni DOM opisan u poglavlju 4.3. i jednostavan i učinkovit sustav reaktivnosti. To može dovesti do bržeg renderiranja i ažuriranja, posebno u aplikacijama s velikim brojem komponenata.

Pri uspoređivanju renderiranja (eng. rendering) valja reći da Angular koristi pristup dvostrane veze podataka (eng. two-way data binding), što može uvesti dodatno opterećenje u nekim scenarijima i potencijalno usporiti performanse renderiranja. Za razliku od Angulara, Vue.js koristi jednosmjerni tok podataka (eng. one-way data flow), što ga može učiniti predvidljivijim u pogledu renderiranja.

Performanse tijekom izvođenja (eng. runtime performance) predstavljaju način na koji web aplikacija radi dok je pokrenuta, za razliku od onoga dok se učitava. Performanse tijekom izvođenja u Angularu mogu biti odlične, posebno kada je dobro optimiziran. Prikladan je za aplikacije velikih razmjera i složenih scenarija. Vue.js je poznat po dobrim performansama izvođenja, što ga čini pouzdanim izborom za širok raspon aplikacija [16].

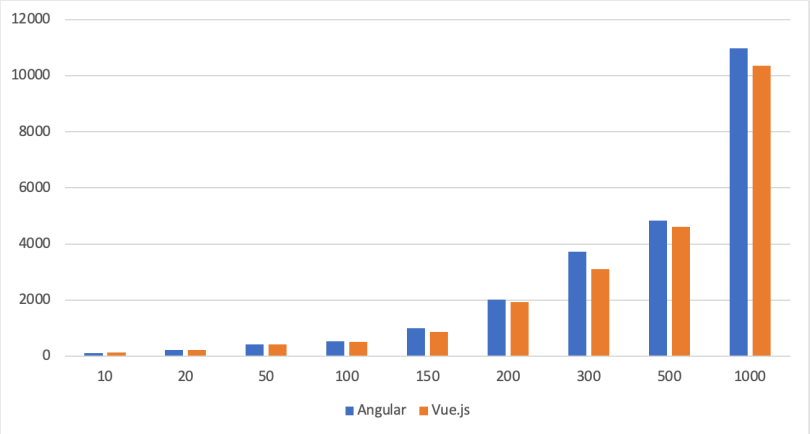
Kao zaključak, može se reći da i Angular i Vue.js mogu pružiti visoko performantne web aplikacije, ali specifične karakteristike performansi mogu varirati ovisno o veličini aplikacije, složenosti i uzorcima korištenja mogućnosti koje ovi radni okviri nude. Prije odabira jednog od njih, važno je razmotriti zahtjeve projekta, stručnost tima i ograničenja razvoja. Profiliranje i optimizacija koda također su ključni za postizanje optimalnih performansi u oba okvira.

4.4.1 Testiranje

U okviru ovog istraživanja, provedeno je i testiranje koje uključuje slanje razčičitog broja HTTP zahtjeva kako bi se analizirale i usporedile performanse Angulara i Vue.js-a u aspektu brzine izvršavanja. Ovaj eksperiment temelji se na realnom scenariju dodavanja zadataka na listu

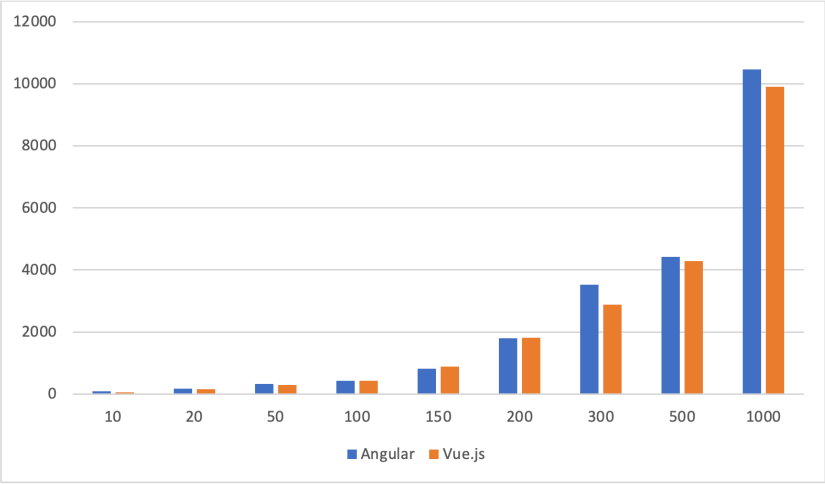
zadataka putem izrađenih web aplikacija, a rezultati će pružiti uvid u to kako se ova dva radna okvira nose s rastućim brojem HTTP zahtjeva.

Na slici 4.12 prikazan je graf koji na svojoj vertikalnoj osi pokazuje broj milisekundi potrebnih za izvođenje zahtjeva dok na svojoj horizontalnoj osi poprima vrijednosti broja zahtjeva koji su poslani. Za svaki broj zahtjeva napravljeno je 10 testiranja te je uzet prosjek svih vremena mjerenja. Iz grafa se vidi da radne performanse ovih radnih okvira nemaju neke značajne razlike pri ovako malom broju poslanih zahtjeva, međutim vrijedi primijetiti kako Vue.js ipak izvršava ovaj zadatak neznatno brže od Angulara za gotovo svaki testni slučaj.



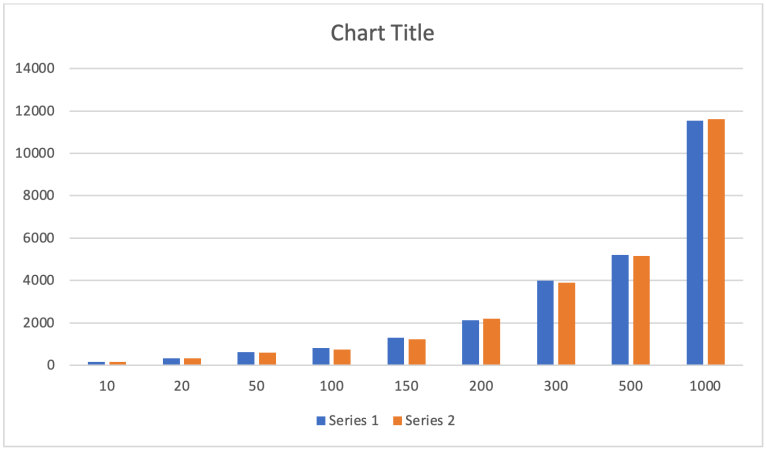
Slika 4.12 Grafički prikaz brzine obrade zahtjeva

Na slici 4.13 prikazan je graf s najboljim vremenima od 10 mjerenja pri obradi zahtjeva oba radna okvira.



Slika 4.13 Grafički prikaz najboljih vremena obrade zahtjeva

Na slici 4.14. prikazan je graf s najlošijim vremenima od 10 mjerenja pri obradi zahtjeva oba radna okvira.



Slika 4.14 Grafički prikaz najlošijih vremena obrade zahtjeva

4.5 Zahtjevi na API

Sljedeća stavka usporedbe ova dva radna okvira biti će način na koji se nose sa zahtjevima poslanima na API.

Angular omogućava komunikaciju s API-jem putem HTTP zahtjeva kako bi dobavio, poslao, ažurirao i brisao podatke iz vanjskog izvora. Glavni alat koji Angular koristi za ove operacije je ugrađeni Angular *HttpClient* modul. Kako bi se ostvarila komunikacija s API-jem, potrebno je uvesti *HttpClient* modul u komponentu ili uslugu koja ga koristiti. Zatim se mogu koristiti metode poput *get, post, put* i *delete* za slanje HTTP zahtjeva [18].

Pri dohvaćanju podataka s API-ja, koristi se *get* metoda, specificirajući URL resursa koji se dohvaća. Rezultat ovog poziva je tzv. Observable koji se može prenijeti u podatke koristeći *subscribe* metodu. Ovo se često izvodi u *ngOnInit* metodi komponente kako bi se dohvatili podaci pri inicijalnom učitavanju. Na slici 4.15 vidi se način korištenja HTTP servisa za dohvaćanje svih zadataka određenog korisnika. Valja napomenuti kako je proces zahtjeva razdvojen na dva dijela odnosno prvi dio smješten je u *ngOnInit* metodi kako bi se pozvao pri samom učitavanju određene komponente aplikacije te se ondje poziva drugi dio koji zapravo izvršava zahtjev, a odijeljen je u datoteku sa servisima radi lakšeg održavanja i čitljivosti koda. Zadatci koje vraća poslužitelj

dohvaćaju se metodom *subscribe* te se spremaju u polje zadataka koje se kasnije upotrebljava za njihov prikaz.

Uz prethodno navedeno, Angular omogućuje i postavljanje zaglavlja zahtjeva, što je korisno za slanje tokena za autentifikaciju ili drugih meta podataka koji su potrebni za API komunikaciju. Zaglavlja se obično postavljaju pomoću *HttpHeaders* objekta.

```
ngOnInit(): void {
    this.taskService.getTasks(this.user_id).subscribe((tasks) => (this.tasks = tasks));
}
getTasks(user_id?: number): Observable<Task[]>{
 const apiUrl = "http://localhost:8000/api/tasks";
 const url = `${apiUrl}/${user_id}`;
 return this.http.get<Task[]>(url);
}
```

Slika 4.15 Primjer dohvaćanja podataka HTTP zahtjevom koristeći Angularov API

Vue.js također omogućuje komunikaciju s API-jem putem HTTP zahtjeva kako bi se dobavili, poslali, ažurirali i brisali podaci iz vanjskog izvora. Glavni alat koji Vue.js koristi za ove operacije je Axios, popularna JavaScript knjižnica za rad s HTTP zahtjevima. Za korištenje Axios knjižnice u Vue.js projektu, prvo je potrebno instalirati Axios i zatim ga uključiti u svoj projekt. Za dohvaćanje podataka s API-ja, koristi se Axios *get* metoda te je potrebno specificirati URL resursa koji se dohvaća. Rezultat ovog poziva je *response*.

Primjer zahtjeva za dohvaćanje zadataka te njihovog spremanja iz odgovora zahtjeva u polje zadataka nalazi se na slici 4.16. Knjižnica Axios također omogućuje postavljanje zaglavlja zahtjeva. Zaglavlja se obično postavljaju pomoću konfiguracijskog objekta koji se prosljeđuje prilikom slanja zahtjeva.

```
methods: {
    getTasks(){
        axios.get(`http://localhost:8000/api/tasks/{user_id}`)
        .then(response => {
            this.tasks = response.data
        })
    },
},
created(){
    this.getTasks();
}
```

Slika 4.16 Primjer dohvaćanja podataka HTTP zahtjevom koristeći Axios u Vue.js-u

4.6 Ekosustav i zajednica

Ekosustav i zajednica su značajni aspekti prilikom uspoređivanja Angulara i Vue.js-a, jer utječu na dostupnost resursa, podršku i održavanje ovih JavaScript okvira. Angular, razvijen i podržavan od strane Googlea, ima snažan korijen u industriji i korporativnoj zajednici. Ovo pruža stabilnost i kontinuiranu podršku, ali također može rezultirati formalnijim i često strožim putem razvoja. Službena dokumentacija, razni alati i resursi su opsežni i često se ažuriraju, osiguravajući programerima obilje materijala za učenje i rad.

S druge strane, Vue.js je razvijen i održavan od strane manje formalne zajednice. Ovo pruža veći stupanj fleksibilnosti i otvorenosti za raznolike pristupe i koncepte. Unatoč manjoj podršci od strane velikih korporacija, Vue.js je uvelike dobio na popularnosti i ima predanu zajednicu programera.

Što se tiče ekosustava, Angular ima širok spektar zvaničnih i ne zvaničnih dodataka, modula i alata za razvoj. Ovo uključuje različite knjižnice za usluge, rute, upravljanje stanjem i mnoge druge potrebe u razvoju aplikacija. Angular je često preferiran za velike i kompleksne projekte, gdje su skalabilnost i organizacija ključne. Vue.js također ima svoj ekosustav dodataka, iako je manji u usporedbi s Angularom. Ipak, zbog jednostavnosti i fleksibilnosti Vue.js-a, mnogi programeri preferiraju ga za manje i srednje projekte gdje brza izrada prototipova i jednostavnost razvoja igraju važnu ulogu.

Kada je riječ o dokumentaciji, Angular se ističe kao iznimno dobar. Na službenoj stranici nudi čak i lekcije za izgradnju aplikacije koja koristi najbitnije dijelove Angulara korak po korak. S druge strane, dokumentacija Vue.js-a vrlo je temeljita, no ne nudi posebne lekcije već objašnjava svoje koncepte na nezavisnim primjerima. Pri izradi projekta kao što je aplikacija To-Do za ovaj rad, veoma je lako pomoću dokumentacije usvojiti osnovne koncepte, sintaksu i specifičnosti kako Angulara tako i Vue.js-a.

5 ZAKLJUČAK

Angular i Vue.js dva su radna okvira koja se ističu među najkorištenijima u industriji razvoja web aplikacija. Kroz izradu To-Do aplikacije u oba okvira uspješno je odrađena temeljita usporedba njihovih svojstava. Kada je riječ o sličnostima, najbliskije točke u načinu funkcioniranja ovih okvira jesu to što dijele komponentnu arhitekturu, reaktivni pristup te oba imaju vrlo snažne razvojne alate.

S druge strane, iako se veoma često uspoređuju i koriste u iste svrhe, ovi radni okviri itekako imaju svoje razlike i specifična svojstva. Kao najveći odmak Angulara od Vue.js-a pokazala se njegova složenija arhitektura te poštivanje strožih konvencija radi samog TypeScripta kojeg Angular koristi. Rastavljanje svake komponente u više datoteka s obzirom na njihovu svrhu i stroga tipizacija čine ga težim za razumjeti od Vue.js-a. Također, pri nekim osnovnim operacijama kao što su prosljeđivanje podataka iz komponente roditelja u komponentu dijete, dolazi do potrebe korištenja raznih anotacija koje pri razvoju istih funkcionalnosti u Vue.js-u nisu potrebne. Iako se nešto od navedenog na prvi pogled čini kao mana Angulara, korištenje takvih konvencija može biti vrlo korisno u snalaženju projektom kada su u pitanju aplikacije velikog obima komponenti odnosno klasa i njihovih međusobnih interakcija.

Jedna od velikih prednosti Angulara je ta što nudi već ugrađena rješenja za gotovo sve potrebe izrade jednostraničnih aplikacija dok Vue.js ipak zahtjeva uključivanje dodatnih knjižnica kako bi omogućio cjelokupan razvoj aplikacije. Već pri jednoj od osnovnih funkcija web aplikacije, slanju HTTP zahtjeva poslužitelju, uočljivo je kako Angular za to osigurava svoj Angular *HttpClient* modul dok je za to u Vue.js-u potrebno uvesti knjižnicu kao što je Axios.

Važna stavka pri odabiru radnog okvira za izradu web aplikacije je i sam ekosustav ranog okvira te zajednica koja se formirala oko njega. Kada je riječ o Angularu, na njegovim službenim stranicama postoji vrlo opisna i temeljita dokumentacija koja uz lekcije koje nudi može biti izvrsna za početnike koji se po prvi puta žele okušati u razvoju jednostraničnih web aplikacija. To što ga razvija i održava Google može biti samo vjetar u leđa onima koji se odluče za njegovo korištenje. Može se reći da Vue.js, iako je nešto manji po ovom pitanju, nudi dostojnu dokumentaciju, a

njegova gotova rješenja za funkcionalnosti kao što su autentifikacija i plaćanje mogu biti presudne pri odabiru Vue.js-a preko Angulara.

Konačni izbor između Angulara i Vue.js-a ovisit će o specifičnim potrebama i obujmu projekta, no i o tome postoji li predznanje poput poznavanja TypeScripta. Oba okvira nude veoma dobre alate za razvoj web aplikacija, a poznavanje njihovih različitosti može samo pomoći programerima da odaberu bolji radni okvir za svoje projekte.

BIBLIOGRAFIJA

[1] »Angular,« [Mrežno]. Available: https://angular.io/cli. [Pokušaj pristupa 29. 8. 2023.].

[2] »What Is Angular CLI,« [Mrežno]. Available: https://mindmajix.com/what-is-angular-cli. [Pokušaj pristupa 27. 8. 2023.].

[3] »DOM Manipulations in Angular,« [Mrežno]. Available: https://dev.to/imsabodetocode/dom-manipulations-in-angular-1dh1. [Pokušaj pristupa 27. 8. 2023.].

[4] »Vue,« [Mrežno]. Available: https://vuejs.org/guide/introduction.html. [Pokušaj pristupa 1. 9. 2023.].

[5] »Getting Started With the Vue CLI,« [Mrežno]. Available: https://stackabuse.com/getting-started-with-the-vue-cli/. [Pokušaj pristupa 28. 8. 2023.].

[6] »Introduction to Vue,« [Mrežno]. Available: https://vuejs.org/guide/introduction.html. [Pokušaj pristupa 29. 8. 2023.].

[7] »Laravel Documentation,« [Mrežno]. Available: https://laravel.com/docs/10.x. [Pokušaj pristupa 2. 9. 2023.].

[8] »Tailwind,« [Mrežno]. Available: https://tailwindcss.com. [Pokušaj pristupa 29. 8. 2023.].

[9] »Tailwind,« [Mrežno]. Available: https://tailwindui.com/components/marketing/feedback/404-pages. [Pokušaj pristupa 29. 8. 2023.].

[10] »Oracle,« [Mrežno]. Available: https://www.oracle.com/mysql/what-is-mysql/. [Pokušaj pristupa 2. 9. 2023.].

[11] »Difference between Fetch and Axios.js for making http requests,« [Mrežno]. Available: https://www.geeksforgeeks.org/difference-between-fetch-and-axios-js-for-making-http-requests/.. [Pokušaj pristupa 4. 9. 2023.].

[12] »Axios Documentation,« [Mrežno]. Available: https://axios-http.com/docs/intro. [Pokušaj pristupa 2. 9. 2023.].

[13] »npm Docs,« [Mrežno]. Available: https://docs.npmjs.com. [Pokušaj pristupa 3. 9. 2023.].

[14] »Postman,« [Mrežno]. Available: https://www.postman.com/product/what-is-postman/. [Pokušaj pristupa 1. 9. 2023.].

[15] »Vue vs Angular: Neck-to-Neck Comparison,« [Mrežno]. Available: https://www.bacancytechnology.com/blog/vue-vs-angular. [Pokušaj pristupa 6. 9. 2023.].

[16] »Which Framework to Choose,« [Mrežno]. Available: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/#gref. [Pokušaj pristupa 7. 9. 2023.].

[17] »How to do DOM Manipulation properly in Angular,« [Mrežno]. Available: https://medium.com/weekly-webtips/how-to-do-dom-manipulation-properly-in-angular-e6fdb70a6e0e. [Pokušaj pristupa 1. 9. 2023.].

[18] »Make API Calls the Right Way in Angular,« [Mrežno]. Available: https://www.knowledgehut.com/blog/web-development/make-api-calls-angular#use-httpclient-library. [Pokušaj pristupa 4. 9. 2023.].

[19] »Difference between Fetch and Axios.js,« [Mrežno]. Available: https://www.geeksforgeeks.org/difference-between-fetch-and-axios-js-for-making-http-requests/. [Pokušaj pristupa 8 2023].

Sažetak

Cilj ovog rada bio je usporediti Angular i Vue.js radne okvire za razvoj web aplikacija. Za potrebe usporedbe izrađene su dvije identične To-Do aplikacije, svaka u svom radnom okviru. Potom je odrađena usporedba radnih okvira na temelju sljedećih stavki: komponente, složenost i krivlja učenja, DOM, zahtjevi na API, performanse te zajednica i ekosustav. Zaključak ovog rada je da je Vue.js jednostavniji i pogodniji okvir za manje aplikacije i početnike u njihovom razvoju, no to ne umanjuje njegove mogućnosti makar katkad zahtjeva uključivanje dodatnih knjižnica. Prednosti Angulara očituju se u njegovoj ponudi cjelokupnog rješenja za izradu aplikacija, a njegova arhitektura pogoduje većim aplikacijama. Na kraju, odabir okvira ovisi o potrebama projekta i predznanju programera koji će ih koristiti. Međutim, oba okvira snažno konkuriraju u ponudi radnih okvira za izradu web aplikacija.

Ključne riječi **– Angular, Vue.js, Laravel, radni okvir, Jednostranična web aplikacija**

Abstract

This paper aims to compare Angular and Vue.js frameworks for web application development. For comparison, two identical To-Do applications were created, each using its respective framework. Subsequently, the frameworks were compared based on the following criteria: components, complexity and learning curve, DOM, API requests, performance, and community and ecosystem. This paper concludes that Vue.js is a more straightforward and more suitable framework for smaller applications and beginners in development, although it occasionally requires the import of additional libraries. The advantages of Angular are evident in its offering of a comprehensive solution for application development, with its architecture particularly suitable for larger applications. In the end, the choice of framework depends on the project's requirements and the developer's expertise. However, both frameworks are strong competitors in the framework market for web application development.

Keywords **– Angular, Vue.js, Laravel, Framework, Single Page Application**