Razvoj Vue.js aplikacije

Kokić, Marta

SVEUČILIŠTE U SPLITU

PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

RAZVOJ VUE.JS APLIKACIJE

Marta Kokić

Split, rujan 2024.

Temeljna dokumentacijska kartica

Završni rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

RAZVOJ VUE.JS APLIKACIJE

Marta Kokić

SAŽETAK

Rad se bavi: usporedbom različitih vrsta aplikacija i popularnih okvira za izradu web aplikacija. Nakon analize mogućih okvira, njihovih prednosti i nedostataka, detaljno se objašnjava razlog odabira Vue.js-a. Rad također prikazuje ključne značajke odabranog okvira te prikazuje njihovu primjenu unutar aplikacije za izradu bilješki. Uz to, prikazuje postupak izrade jednostavnog poslužitelja i baze podataka potrebnih za rad aplikacije.

Ključne riječi: Vue.js, frontend, poslužitelj, komponenta, SPA, direktive, npm, JSON web token, Options API

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 37 stranica, 41 grafičkih prikaza i 18 literaturnih navoda.

Izvornik je na hrvatskom jeziku.

Mentor: **Dr.sc. Divna Krpan**, docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Ocjenjivači: **Dr.sc. Divna Krpan**, docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Dr.sc. Monika Mladenović, docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Antonela Prnjak, asistentica Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad prihvaćen: Rujan, 2024.

Basic documentation card

Thesis

University of Split
Faculty of Science
Department of Computer Science
Ruđera Boškovića 33, 21000 Split, Croatia

DEVELOPMENT OF VUE.JS APPLICATION

Marta Kokić

ABSTRACT

This paper is concentrated on: comparing different types of applications and popular web application frameworks. After analyzing the possible frameworks, their advantages and disadvantages, the choise of using Vue.js is explained in detail. The paper also shows the key features of the selected framework and demonstrates their application within the note-taking application. In addition, it shows the process of creating a simple server and database necessary for the operation of the application.

Key Words: Vue.js, frontend, server, component, Single Page Application, directive, npm, JSON web token, Options API

Thesis deposited in library of Faculty of science, University of Split.

Thesis consist of: 37 pages, 41 figures and 18 references

Original language: Croatian

Supervisor: **Divna Krpan**, **Ph.D.** Assistant Professor of Faculty of Science, University of Split

Reviewers: **Divna Krpan**, **Ph.D.** Assistant Professor of Faculty of Science, University of Split,

Monika Mladenović, Assistant Professor of Faculty of Science, University of Split,

Antonela Prnjak Instructor of Faculty of Science, University of Split

Thesis accepted: September, 2024.

Sadržaj

Uvod

U današnje doba, internet je od iznimne važnosti. Pomoću njega možemo slušati glazbu, ostati u kontaktu sa svojim bližnjima, pronaći bitne i relevantne informacije, organizirati se… Kako bismo mogli obavljati takve radnje potrebne su nam aplikacije. Postoje različite vrste aplikacija, a to su nativne, web i hibridne. One koje su na fokusu u ovom završnom radu su web aplikacije, i to izrada istih u Vue.js okviru zajedno sa *backendom* i bazom podataka za tu aplikaciju.

Web aplikacije slične su nativnim aplikacijama, razlika je u tome što ih nije potrebno preuzimati ni instalirati. Ovakve aplikacije se koriste putem preglednika kao što su Chrome, Edge ili Firefox i nalaze se na poslužitelju, a ne na korisnikovom uređaju. Tijekom korištenja web aplikacije, korisnikovo računalo komunicira s poslužiteljem, koji zatim vraća tražene podatke, primjerice prikaz traženih podataka ili poruka da je izrada novog dokumenta uspješno izvedena.

Vue.js je okvir koji omogućuje izradu web aplikacija koristeći JavaScript sintaksu, s mogućnošću korištenja JSX-a ili TypeScript-a, što olakšava prijelaz programerima koji su naviknuli na određenu sintaksu. Vue.js pruža temelj za izradu aplikacija, kako bi se mogli fokusirati na razvoj, bez brige o detaljima. Nudi sve potrebne komponente za izradu aplikacija te omogućuje jednostavnu integraciju dodatnih biblioteka za specifične funkcionalnosti.

1. Vrste aplikacija

Nativne aplikacije (engl. *native applications*) su aplikacije izrađene za određenu platformu. Može se može pokrenuti samo na platformi na kojoj je razvijena. Izrađuju se pomoću SKD ili *software developement kit*: sadrže biblioteke i alate za izradu aplikacije na određenoj platformi [1]. Kako je napravljena za specifičnu platformu, imat će bolje performanse jer može iskoristit sve njene prednosti. No, nedostatci su da se takva aplikacija mora ponovno razviti za svaku platformu.

Web aplikaciji možemo pristupiti preko preglednika. Web aplikacija ne može pristupiti nekim značajkama uređaja, za razliku od nativne. Kako bi korisnik mogao koristit web aplikaciju potrebna mu je internetska veza. No, za razliku od nativne, ne treba ih preuzimati niti instalirati, izrada i ažuriranje je jednostavnije i nisu izrađene za jednu platformu. Također, mogu reagirati različito ovisno o platformi na kojoj se nalaze, npr. veličina web stranice i njenih elemenata se mijenja ovisno o veličini zaslona i mogu sadržavati različite vrste kontrole [2].

Hibridne mogu raditi na različitim platformama. One su kombinacija nativnih i web aplikacija. Izvedba tih aplikacija može biti ista kao i kod nativnih, dok je cijena razvoja niža. Mogu se preuzeti isto kao i nativne. Koriste HTML, CSS i JavaScript baš kao i web, a pokreću se unutar nativnog spremnika (engl. *native container*) koji omogućuje da aplikacija se izvršava na različitim platformama [1]. Web aplikacije ne mogu pristupiti nekim funkcionalnostima uređaja, dok hibridne im mogu pristupiti.

2. Usporedba frontend okvira za razvoj web aplikacija

2.1. Razlika između okvira i biblioteke

Uvijek je jednostavnije raditi projekt sa nekim temeljem, nego raditi sve „od nule“. Kada bismo izradili cijeli projekt bez temelja, potrebno je pisati čist, testiran kod bez grešaka inače će drugim programerima biti vrlo teško razumjeti napisano.

Okviri i biblioteke nam značajno olakšavaju razvoj projekta, omogućuju da se ne bavimo osnovnim problemima, već koristimo alate i elemente koji su unaprijed definirani. No koja je razlika između njih?

2.1.1. Okvir

Okvir (engl. *framework*) je temelj za razvoj aplikacija koji definira njenu strukturu. Sadrži funkcije koje programer uređuje kako bi izradio aplikaciju sa što manje grešaka. Okvir ima širi opseg od biblioteke, sadrži sve neophodno za izradu aplikacije. Okviri nam nudi [3]:

- Lakše testiranje koda
- Čist kod koji je jednostavan za razumijevanje
- Smanjenje redundantnosti koda
- Smanjenje vrijeme i troškova izrade projekta
- Značajke i funkcionalnosti koje nudi okvir mogu se mijenjati i proširivati.

Neki okviri su: Angular je okvir izrađen od strane Googlea, Express je Node.js okvir koji omogućuje jednostavan razvoj web aplikacija, Django je Python okvir otvorenog koda, i još mnogi.

2.1.2. Biblioteka

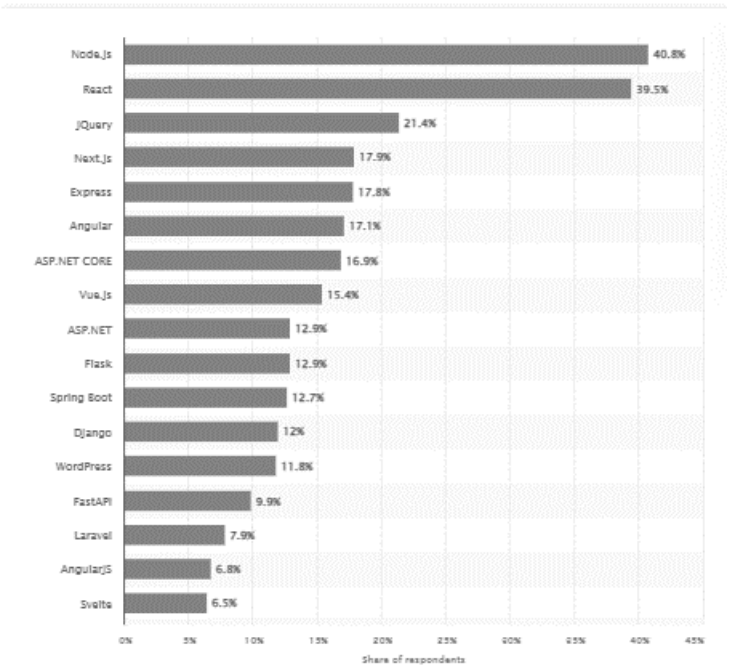
Biblioteka (engl. *library*) je skup značajki kao što su: funkcije, objekti i moduli, koje uvozimo kako bismo obradili specifične zadatke. Manje su od okvira jer se fokusiraju na manje područje, npr. nizovi ili IO ili pružaju pristup nekim funkcijama i konstantama. Biblioteka nam omogućuje pisanje koda sa što manje grešaka te brzu i jednostavnu implementaciju funkcionalnosti.

Neke od biblioteka su: NumPy što je Python biblioteka za strojno učenje za nizove (engl. *array-oriented*); Pandas Python biblioteka za manipulaciju podacima, TensorFlow je Pyton biblioteka za obučavanje neuronskih mreža, OpenCV je C++ biblioteka za obradu slika i računalni vid, i mnogi drugi.

Ključna razlika između biblioteke i okvira je u „inverziji kontrole“. Kada pozivamo element iz biblioteke, mi imamo kontrolu, no kod okvira je drugačije: okvir ima kontrolu i on naš kod poziva. [4]

2.2. Usporedba okvira

Na Statista web stranici postoji graf koji prikazuje najčešće korištene web okvire 2024. godine.



Sl. 2.1 - Najčešće korišteni web okviri među programerima širom svijeta, od 2024, izvor: *[5]*.

Na temelju ovog grafa možemo usporediti neke od mogućih i popularnih web aplikacija.

2.2.1. Angluar

Angular je JavaScript *frontend* okvir izrađen od strane Google-a, objavljen 2012. godine. Uveo je mnoge značajke koje olakšavaju programerima izradu *Single Page* aplikacija. U početku se zvao Angular JS: to su sve v1.x verzije Angulara [6].

Angular uvodi i direktive, koje omogućuju programerima dodavanje svojih atributa HTML elemenata i „*dependency injection*“: komponente aplikacije se spajaju što omogućava ponovnu upotrebu koda [7]. Koristi TypeScript, JavaScript nadogradnju u kojoj se mogu dodavati tipovi varijablama. Savršen je za velike i kompleksne projekte, sadrži alate koje možemo koristit odmah nakon stvaranja projekta.

2.2.2. React

Na službenoj stranici Reacta, on se definira kao JavaScript biblioteka za izradu sučelja aplikacije. Ono je biblioteka, a ne okvir jer ne nameće strukturu ni pravila za izradu web aplikacije, za razliku od okvira koji moraju upravljati svim dijelovima aplikacije, od baze podataka do osvježavanja preglednika [8]. React se fokusira na izradu i upravljanje interaktivnog korisničkog sučelja na računalu ili mobitelu. On je *Single Page* aplikacija i ima veliki fokus na korisničko sučelje, mnogo biblioteka, alata i veliku zajednicu.

Postoji ReactJS i React Native. Kada spominjemo React, najčešće se misli na ReactJs. On se koristi za izradu web aplikacija, no React Native se koristi za izradu nativnih aplikacija.

2.2.3. Svelte

Svelte je JavaScript *frontend* kompajler koji se koristi za izradu web aplikacija koje se mogu podijeliti u komponente. Objavljen je 2016. godine i ima manju zajednicu u odnosu na okvire kao što su Reacta ili Vue.js. Aplikacije izrađene sa Svelte će biti mnogo manje nego one izrađene u Reactu ili Vue.js baš zbog toga što je Svelte mnogo manji od njih.

Za razliku od nekih drugih popularnih okvira, Svelte ne sadrži virtualni DOM. Samim time se vrijeme učitavanja aplikacije smanjuje i one su brže i učinkovitije. Također, ovaj okvir se brine da se promjene automatski prikazuju unutar preglednika, bez dodatnih biblioteka i okvira. Jednostavan za naučiti i razumjeti zbog svoje jednostavne sintakse. Ovaj okvir se najčešće koristi kod aplikacija u kojima je brzina, količina i složenost koda od velike važnosti.

2.2.4. Vue.js

Vue.js ili skraćeno „Vue“, je *open-source* Javascript *frontend* okvir kojeg je izradio Evan You za razvoj web stranica koji koristi standardni HTML-a, CSS-a i JaveScript. Za razliku od

Angulara i Reacta, Vue omogućuje korištenje i TypeScripta i JSX-a što ga čini prikladno početnicima jer korisnici nisu prisiljeni učiti novu sintaksu, već mogu koristiti ono s čime su upoznati.

Vue.js se koristi za izradu samostalnih *widgeta* (na primjer traka za pretraživanje) i komponenti. Vue je *Single Page* aplikacija: on upravlja aplikacijom kako bi prikaz stranica bio brži. Ovaj okvir je i fleksibilan: što znači da možemo koristiti i biblioteke koje odgovaraju nekom specifičnom zadatku. Vue se koristi za izradu malih i srednjih web aplikacija.

2.3. Razlog odabira Vue.js

Budući da je ideja aplikacije bila slična prethodnom projektu izrađenom u Reactu, ovaj okvir je bio odbačen. Angular također eliminiran zbog njegove strme krivulje učenja u usporedbi s drugim okvirima.

Na prvi pogled Svelte i Vue imaju sličan način izrade komponenti: svi elementi vezani za komponentu se pišu unutar te komponente (HTML, CSS i JS). S obzirom na to što Svelte i Vue nude, nije bilo presudno koji od ta dva okvira odabrati: sve prednosti koje pružaju nisu bile presudne. Na kraju je odabran Vue zbog svoje malo strmije krivulje učenja.

Vue.js je jedan veoma interesantan okvir. Omogućava izradu složenih aplikacija na vrlo jednostavan način, pisanje svih relevantnih dijelova stranice unutar jedne datoteke. Daje nam mnogo slobode: iako sve možemo pisati unutar jedne datoteke, ovaj okvir nam omogućuje da ih pišemo u različitim datotekama i samo uvezemo. Vrlo je fleksibilan. Omogućuje korištenje TypeScripta ili JSX-a umjesto JavaScripta u slučaju da smo upoznati ili upoznati sa njima.

Ovaj okvir će odgovarati iskusnijim programerima zbog svoje fleksibilnosti, no zbog svoje detaljne dokumentacije, široke zajednice i jednostavnosti preporuča se i početnicima. Omogućuje ponovno korištenje komponenti, a njegova najnovija verzija uvodi nove metode za još efikasnije korištenje koda, kao što je *teleport*.

Stoji među okvirima koje podupiru veliki *tech* divovi kao što su Google i Facebook, dok Vue je podržan samo sa donacijama sponzora i zajednice otvorenog koda.

3. Vue.js

3.1. Single Page Application

Single Page Application ili skraćeno SPA, je vrsta web aplikacije koja učitava samo jedan web dokument i ažurira tijelo tog dokumenta, umjesto osvježavanja stranice [9]. Takve aplikacije obrađuju različite stranice u pregledniku umjesto na poslužitelju. Kod nekih okvira, kada bismo upisali URL stranice, šaljemo zahtjev serveru koji ga obrađuje i šalje HTML stranicu koja se onda prikazuje na pregledniku. Kada pritisnemo na neki drugi link na stranici, šalje se novi zahtjev poslužitelju, koji šalje novu stranicu. Vue.js ima drugačiji pristup: šalje početni zahtjev poslužitelju koji šalje HTML stranicu i Vue.js paket. Taj paket kontrolira preglednik i renderira komponente koje su na toj web stranici: kontrolira poveznice i usmjerava nas na željenu stranicu umjesto da se šalje zahtjev serveru, što poboljšava brzinu i odziv.

3.2. Vite

Vite je alat za izradu koji čini njihovo razvijanje brže i jednostavnije razvojno iskustvo za web aplikacije [10]. Kako aplikacija raste, raste i količina koda koja se nalazi unutar nje. To jako utječe na performanse i vrijeme potrebno za prikaz izmjena na pregledniku, dok Vite je dizajniran da bude brz.

Vite koristi alate kao što su Rollup.js i esbuild. Njegov originalan tvorac je Evan You, čovjek koji je izradio Vue. No, kako je rastao, počeo je podržavati i ostale okvire kao što su React i Svelte.

3.3. Virtualni DOM

Ideja virtualnog DOM-a je da se sve izmjene čuvaju u memoriji i, nakon usporedbe dvaju stabala, razlike primjenjuju na stvarni DOM. Proces provjere razlika i primjene promjena na stvarni DOM se naziva *patch* ili *diffing*.

Ideju je uveo React kojeg su kasnije implementirali mnogi okviri, pa čak i Vue. Umjesto direktne manipulacije DOM stabla, Vue će izvršavati manipulaciju DOM-a i na taj način je umanjiti.

3.4. Direktive

Direktive su atributi unutar HTML elementa koji daju dodatnu funkcionalnost tom elementu. Postoje različite direktive, neke od njih su:

- *v-on*: koji se koristi za događaje kao što su click, input ili mouse
- *v-for* sa kojim prolazimo po nizu sa „for“ petljom i izrađujemo HTML elemente
- *v-if, v-else-if* i *v-else* se koristi kako bismo prikazali element ovisno o stanju
- *v-show* koji izmjenjuje CSS svojstvo *visible*: umjesto da ga izbacuje iz DOM stabla, ono ga samo sakriva

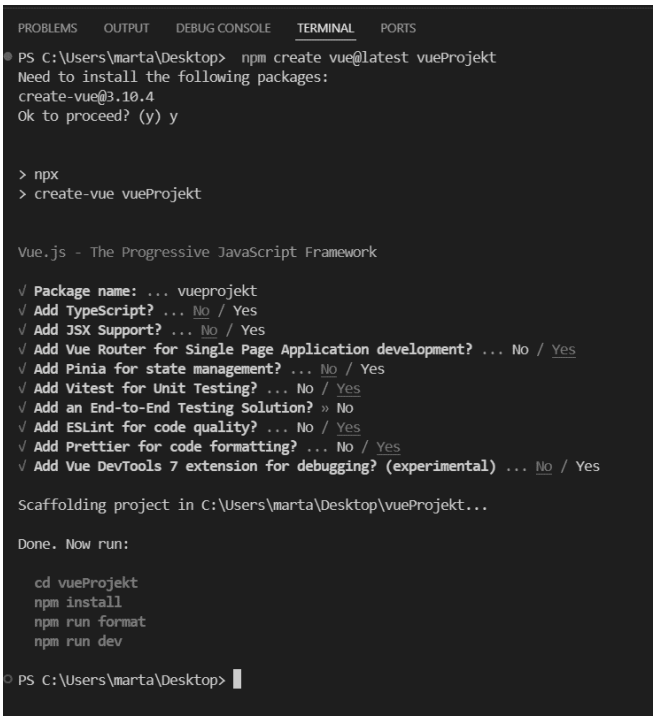
Svaka direktiva započinje sa *v-* malo slovo v označava Vue. Mogu se pisati i kraće, Na primjer, umjesto *v-on:click* se piše @.

3.5. Instalacija

Prije no što krenemo sa izradom web aplikacije, na računalu bi trebali imati Node.js instaliran. Prema Vue.js službenoj stranici, postoje 4 različita načina kako izraditi Vue.js projekt [11]:

- Uvesti ga kao CDN paket na stranicu
- Preuzeti JS datoteke i sami *hostirati*
- Instalacija pomoću npm
- Koristeći službeni CLI za izradu predloška

U Vue.js dokumentaciji, preporučena metoda za izradu velikih Vue.js aplikacija je pomoću npm (*Node Package Manager*). Kako bismo izradili Vue.js projekt, odabiremo direktorij unutar kojeg ćemo stvoriti projekt. Pokrećemo terminal i unosimo naredbu: *npm create vue@latest* kako bismo započeli instalaciju projekta.



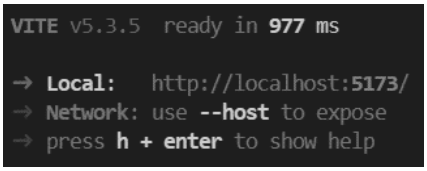
Sl. 3.1 - Proces instalacije Vue.js projekta.

- **TypeScript** - nadogradnja JavaScripta koja omogućuje dodavanje tipova varijabli
- **JSX Support** – JSX je ekstenzija JaveScript i XML.
- **Vue Router for Single Page Application developement** – omogućuje stvaranje navigacije unutar web aplikacije.
- **Pinia for state management** – sustav za upravljanje stanjem za Vue.js. Omogućuje dijeljene podataka između komponenti.
- **Vitest for Unit Testing** – okvir za unit testiranje, sadrži ESM, TypeScript i JSX podršku.
- **End-to-End Testing Solution** – testiranje funkcionalnosti aplikacije od početka do kraja, oponaša interakcije korisnika kako bi osigurali da aplikacija radi očekivano.
- **ESLint** - pomaže u potrazi grešaka u kodu.
- **Prettier** - omogućuje da kod ima dosljedan stil.
- **Vue DevTools** – poboljšava otklanjanje grešaka Vue.js aplikacije.

Kao što možemo vidjeti u slici (Sl. 3.1), prije instalacije možemo odabrati želimo li određene elemente unutar projekta. Unutar terminala upisujemo *cd nazivProjekta* (engl. *change directory*): želimo promijeniti svoje mjesto iz C:\Users\marta\Desktop u C:\Users\marta\Desktop\nazivProjekta. Kada smo izvršili tu naredbu, instaliramo sve biblioteke i zavisnosti pomoću *npm install* naredbe.

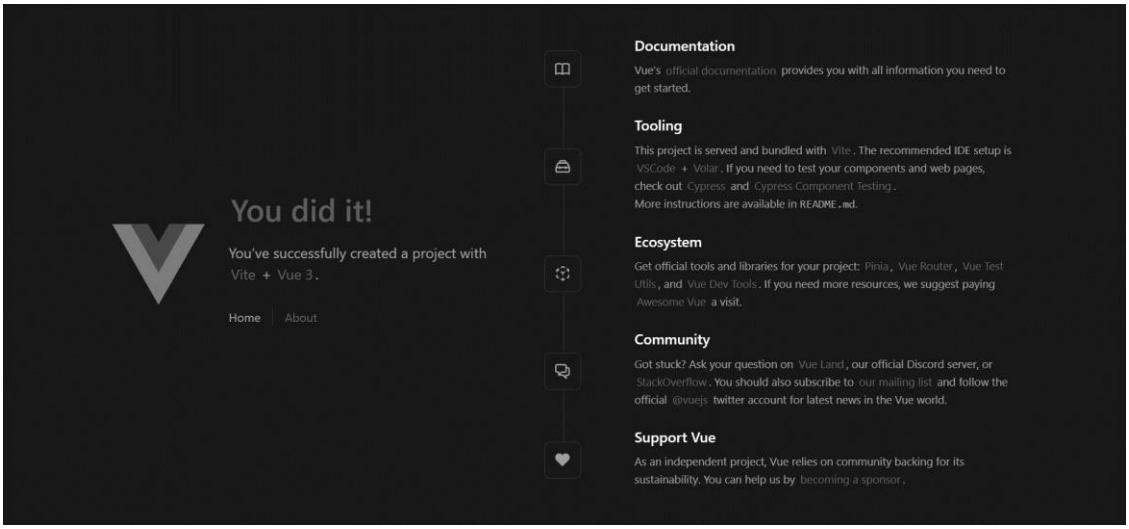
Na slici (Sl. 3.1) možemo vidjeti da je napisana još jedna naredba *npm run format*. Ova naredba je vezana za Prettier koji formatira svaku .js ili .jsx datoteku koje se nalaze unutar *src* direktorija. Tu naredbu bi trebali izvesti svaki put kada izvršimo neku promjenu unutar projekta. Bitno je za naglasiti da će se ova naredba samo prikazati ako smo prilikom konfiguracije projekta postavili da želimo preuzeti Prettier.

Slijedeća naredba se koristi za pokretanje lokalnog razvojnog poslužitelja koji će nam prikazivati aplikaciju u stvarnom vremenu dok je uređujemo. Unutar terminala upisujemo naredbu *npm run dev*.



Sl. 3.2 - Ispis unutar terminala nakon upisa *npm run dev* naredbe.

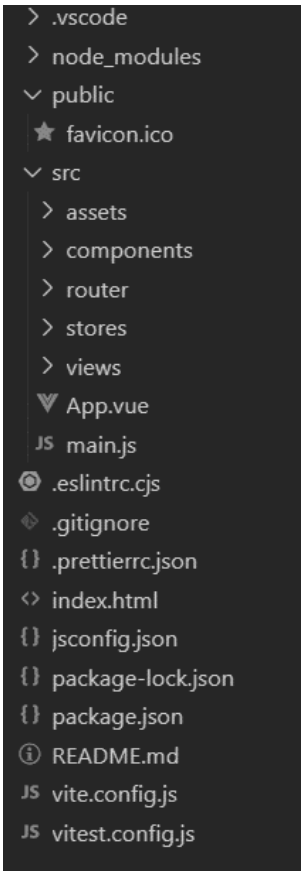
Adresa koja je ispisana u terminalu, kada tu adresu otvorimo u pregledniku, možemo vidjeti našu aplikaciju.



Sl. 3.3 - Izgled preuzetog predloška na web pregledniku.

3.6. Struktura predloška

Nakon što smo završili konfiguraciju, možemo pogledati što se sve nalazi u predlošku.



Sl. 3.4 – Struktura preuzetog predloška.

- **.vscode** - direktorij koji se koristi za postavljanje i održavanje dosljedne razvojne okoline.
- **public** - direktorij koji sadrži statičke elemente.
- **favicon.ico** – ikona aplikacije, trenutno je Vue.js logo.
- **package.json i package-lock.json** - dvije konfiguracijske datoteke za cijelu aplikaciju.
- **src** – (engl. *source code*) direktorij koji sadrži sve Vue.js komponente koje smo izradili
- **assets** – direktorij za statičke elemente, na primjer CSS datoteke i slike.
- **router** – direktorij koji sadrži rute, detaljnije u nastavku.
- **.eslintrc.cjs** – omogućuje konfiguraciju ESLint pomoću JavaScript koda.
- **.gitignore** – ako projekt postavimo na GitHub, unutar gitignore datoteke ćemo postaviti sve datoteke i direktorije koje ne želimo poslati na granu.
- **App.vue** – *root* komponenta.
- **main.js** – pokreće aplikaciju.
- **index.html** - unutar ove datoteke se ubacuje *root* komponenta, glavni HTML predložak.
- **jsconfig.json** - navodi JS korijenske datoteke i opcije za značajke.

- **vite.config.js**: konfiguracijska datoteka za Vite.

Unutar package.json možemo provjeriti verzije svih preuzetih komponenti pa čak i verziju Vue.js-a.

Sl. 3.5 - Izgled package.json datoteke, prikaz verzija svih preuzetih elemenata projekta.

Postoje dva direktorija u koja možemo spremati komponente, a to su *views* i *components*. No koja je razlika među njima? Svrha *views* direktorija je da pohranjujemo glavne stranice aplikacije, kao što je stranica za prijavu, početna stranica, itd. U *components* direktoriju pohranjujemo komponente koje možemo koristiti u više različitih komponenti. Iako, možemo pohraniti sve komponente unutar jednog direktorija, to neće utjecati na aplikaciju. Koristimo ih za organizaciju aplikacije. Ako radimo kompleksnu web aplikaciju, sa mnogo komponenti: moguće je izraditi direktorije unutar istih.

3.7. Komponente

Komponente su temelj Vue.js aplikacija. One omogućavaju da podijelimo aplikaciju na manje dijelove kojima možemo upravljati zasebno. Vue.js datoteke su sve datoteke koje završavaju sa .vue ekstenzijom. Unutar njih unosimo HTML, CSS i JavaScript kod umjesto izrade 3 različitih datoteka. Nazivaju se *Single File Component* ili skraćeno SFC. Ovo omogućuje izradu jednostavnih aplikacija: svi dijelovi komponente će se nalaziti unutar jedne datoteke što čini održavanje i razumijevanje koda jednostavnijim.

Prilikom izrade kompleksnijih aplikacija, javlja se potreba za ponovnim korištenjem koda. Jedno rješenje je samo kopirati i zalijepiti taj kod, ali tako se nepotrebno komplicira aplikacija. Vue.js nam omogućuje da taj dio koda postavimo u komponentu te uvezemo na stranice na kojima je potrebna, time pojednostavimo i olakšamo razumijevanje koda.

3.8. Vue.js osnove

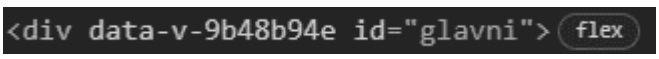
Unutar Vue.js-a unutar skripte možemo, ali i ne moramo koristiti točku-zarez. Unutar HTML-a možemo koristiti jednostruke i dvostruke navodnike za unos vrijednosti atributa.

Bitno je za napomenuti da svaka komponenta, osim App.vue, mora imati dvije riječi unutar svog naziva. Te riječi moramo podijeliti sa crticom ili započeti drugu riječ velikim slovom. Bilo bi poželjno da nazivi komponenti, ako počinju sa malim slovom, ne budu ključne riječi kako ne bi došlo do konflikta unutar aplikacije.

Komponenta se sastoji od 3 dijela: *script, template* i *style*. U tradicionalnim aplikacijama, ovi dijelovi bi bili podijeljeni u tri datoteke: komponenta.js, komponenta.css i komponenta.html. Međutim, u Vue.js-u sve te dijelove možemo pisati unutar jednog dokumenta.

Unutar *template* definiramo strukturu komponente, a unutar *style* stil komponente. Redoslijed ovih dijelova nije bitan, a komponenta ne mora nužno sadržavati *script* i *style*, ali mora template. Brisanjem svih elemenata unutar *template* elementa uzrokovat će pogrešku, jer taj element mora imati barem jedno dijete elemente. Ovo vrijedi za Vue 3. Unutar Vue 2, *template* mora sadržavati samo jedno dijete, koje naravno može sadržavati mnogo djece.

Stilovi su globalni, što znači da ih možemo definirati unutar jedne komponente, čak i ako ta komponenta sama ne koristi te stilove. Oni će se automatski primijeniti na sve komponente koje ih koriste. Stilovi su spremljeni unutar *head* elementa. Ključna riječ *scoped* označava da će se taj CSS izvršiti samo na razini te komponente, ne na cijeloj aplikaciji. Vue dodjeljuje nasumičan *data* atribut komponenti i CSS stilu kada koristimo ključnu riječ *scoped*, kako bi osigurao da će se taj stil primijeniti samo na željene elemente. Ako smo postavili stil unutar druge komponente koja koristi *scoped*, stil neće biti primijenjen van komponente.



Sl. 3.6 - Vue dodjeljuje nasumičan *data* atribut kako bi osigurao da samo elementi unutar te komponente koriste određeni stil.

Ako želimo izbjeći promjenu izgleda unutar svake pojedine komponente, taj stil možemo smjestiti u datoteke kao što su base.css ili main.css koje se nalaze unutar *src/assets* direktorija.

U element *script* možemo dodati ključnu riječ *setup* kako bi se kod izvršavao svaki put kada se kreira nova instanca komponente. Ova sintaksa je uvedena u Vue 3.

Unutar web aplikacije postojat će mnoge varijable u kojima ćemo spremati bitne podatke. U Vue moramo napisati funkciju koja vraća objekt kako bismo mogli spremiti vrijednost. Kada želimo prikazati vrijednost neke varijable, unutar HTML dijela koristimo dvostruke vitičaste zagrade, na primjer: *{{naziv_varijable}}*. Napišemo li krivo naziv varijable, stranica će raditi, ali ništa se neće prikazati. Taj objekt, u kojem smo spremili varijablu, moramo spremiti unutar *export default* koji je neophodan za Vue komponente jer osigurava da se komponenta može pravilno uvesti i renderirati.

```
export default {
    korisnikData() {
        return {
            naslov: "",
            opis: ""
        }
    }
}
```

Sl. 3.7 - Način na koji inicijaliziramo varijable u Vue: definiramo funkciju koja vraća objekt.

Na stranici postoje različite interakcije kao što su: pritisak dugmeta, unos podataka, *mouse hover*, itd. Kako bi aplikacija izvela određeni dio koda koristimo *v-on* unutar određenog elementa. To je direktiva koja omogućuje da aplikacija reagira na različite tipove događaja. Može se skratiti pisanje sa @, na primjer *@click*, *@mouseover*, *@keyup*...

```
<button @click="DodajNoviZadatak">Ok</button>
<button @click="Izlaz">Cancel</button>
```

Sl. 3.8 - Korištenje *@click*.

Unutar navodnih znakova upisujemo kod koji će se izvršiti kada se dogodi određeni događaj. Tu možemo upisivati čisti JS kod, kao što je povećavanje vrijednosti varijable ili poziv funkcije. Funkciju deklariramo unutar *script* elementa, unutar *methods{}*. Vrijednost možemo proslijediti funkciji koristeći sintaksu *@click="funkcija(vrijednost)"*. Ako želimo pristupiti vrijednosti varijable koja se nalazi unutar komponente, koristimo sintaksu *this.naziv_varijable*.

```
methods: {
    exitTask() {
        const res_token = this.token;
        this.$router.push({
            name: 'allTasks',
            state: { res_token }
        });
    },
}
```

Sl. 3.9 - Deklaracija funkcije u Vue.js unutar *methods*.

Možemo uvjetno renderirati određene komponente na temelju vrijednosti pomoću *v-if* ili *v-show*. Razlika između ova 2 načina je u tome što *v-if* će izbaciti element iz DOM stabla, dok *v-show* samo izmjenjuje njegovo CSS svojstvo.

```
<p v-if="filteredTasks.length === 0" id="tip">{{ tip }}</p>
```

Sl. 3.10 - Uvjetno renderiranje, ako je duljina niza jednaka 0, ovaj element će se prikazati na web stranici te će prikazati vrijednost varijable tip.

Prilikom izrade složene web aplikacije, koristit ćemo bazu podataka za pohranu korisničkih podataka. Kako bismo prikazali popis tih podataka, koristimo *v-for="element in array*“, gdje je element varijabla koja predstavlja svaki element dok prolazimo kroz niz. Za prikaz podataka koristimo *{{ element.item }}*.

```
<Cards v-for="item in filteredTasks" :key="item._id" :podaci="item" />
```

Sl. 3.11 - Korištenje for petlje kako bismo poslali komponenti Cards vrijednosti, pod nazivom podaci, koristimo *v-bind* ili skraćeno **:**

Vrijednost *:key* atributa mora biti jedinstvena jer olakšava praćenje i identifikaciju svih stavki istog tipa, na primjer zadataka unutar niza.

Neki zadaci će tražiti prikaz nekih drugih podataka u dijete komponenti. Kako bismo poslali podatke, koristimo *props* ili *state*. Ono nam omogućuje ponovno korištenje komponente jer podaci unutar nje će se moći izmijeniti. *Props* se definira tako da unutar komponente kojoj prosljeđujemo podatke dodajemo atribut s nazivom po izboru i dodjeljujemo odgovarajuću vrijednost. U dijete komponenti, unutar skripte, pristupamo tim podacima preko objekta *props*. Nije potrebno eksplicitno definirati *props* u dijete komponenti jer ga ona automatski prima od roditelja. Također, možemo urediti komponentu na temelju proslijeđene vrijednosti.

```
<Komponenta msg="This is a message" header="This is header"/>
```

Sl. 3.12 - Slanje podataka djetetu.

```
props: ["podaci", "token", "finished"],
```

Sl. 3.13 - Primanje podataka od strane roditelj komponente, njima pristupamo pomoću ključne riječi *this.*

Kako bismo dinamično izmijenili vrijednost atributa kao što su *src*, *href*, *value*, *class*, *style*, itd, koristimo *v-bind* ili skraćeno samo *:naziv_atributa*, kao što možemo vidjeti na slici (Sl. 3.14). Ako promijenimo podatak, element se ažurira, no, ako se element promijeni zbog npr. korisničkog unosa, Vue ga ne može automatski ažurirati. Zato koristimo *v-model*.

```
<input type="text" name="naslov" v-model="this.naslov" placeholder="Title" />
```

Sl. 3.14 - Kako bismo povezali varijablu sa *input* poljem, koristimo *v-model*. Svaki put kada se dogodi neka izmjena unutar tog polja, vrijednost se mijenja unutar te varijable.

Vue.js sadrži izračunata svojstva (engl. *computed properties*) za izračun i prikaz na temelju nekih vrijednosti. Ne primaju ulazne argumente, automatski se ažuriraju kada se ovisnost promijeni. Možemo ga koristiti ako filtriramo podatke pomoću input polja.

```
computed: {
    Complexity is 3 Everything is cool!
    label_button1() { return this.finished ? "Untick" : "Edit" },
    Complexity is 3 Everything is cool!
    label_button2() { return this.finished ? "Delete" : "Finish" }
}
```

Sl. 3.15 - Vrijednost unutar ove dvije varijable ovisi o *boolean* varijabli *finished.*

Vue Template Refs koriste se za upućivanje na različite DOM elemente. Ono može biti zamjena za *getElementById()* ili *querySelector()*. Unutar elementa na kojeg se želimo referencirati dodajemo atribut *ref=“naziv“*, a u funkciji pišemo *this.$ref.naziv*.

Ako želimo poslati HTML elemente unutar komponente, koristimo *slot* kako bismo ih postavili na odgovarajuća mjesta. Ono nam omogućuje slanje elemenata, ali nije zamjena za *props*: s njim ne šaljemo, npr. vrijednosti tipa *string* ili broj. Možemo postaviti elemente unutar *slota* koji će se prikazati ako proslijeđeni elementi nisu dostupni; prikazat će se ili zadani elementi unutar *slota* ili samo oni koji su poslani.

```
<template>
    <Child_component >
        <p>Element koji šaljemo djetetu</p>
    </Child_component>
</template>
```

Sl. 3.16 - Roditelj komponenta. Unutar djeteta postavljamo element koji želimo poslati djetetu.

```
<template>
    <h1>Ovo je dijete!</h1>
    <slot></slot>
    <button>Ok</button>
</template>
```

Sl. 3.17 - Dijete komponenta. Unutar nje postavljamo *slot* element koji označava gdje će se primljeni element postaviti.

Vue 3 uvodi *teleport* značajku koja omogućuje premještanje elemenata komponente prilikom renderiranja, na drugu lokaciju u DOM stablu. Elementima se ne mijenja njihova funkcionalnost nakon premještanja. Unutar atributa *to* upisujemo gdje želimo prebaciti elemente, možemo upisati *id* elementa ili klasu.

```
<Teleport to="body">
  <div class="popUp">
    <p>Ovo je popup </p>
    <button @click="_popup">Close</button>
  </div>
</Teleport>
```

Sl. 3.18 - Korištenje *teleport* značajke, svi element koji se nalaze unutar tog elementa će se prikazati unutar tijela.

Kada izrađujemo stranicu za prijavu korisnika, sve input elemente možemo postaviti unutar *form* spremnika, kako bismo im mogli dodati *require* atribut.

U slučaju da korisnik mora odabrati element iz izbornika, elementima unutar izbornika postavljamo *value* atribut, dok *select* element postavljamo *v-model*. Svaki put kada korisnik izmijeni svoj odabir, vrijednost će se promijeniti.

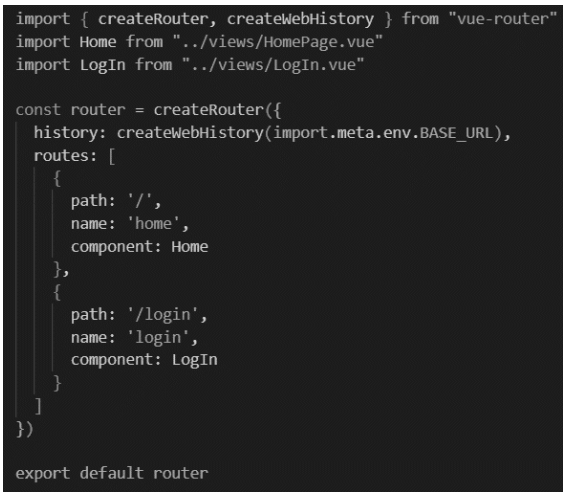
```
<script>
export default {
  data() {
    return {
      email: "",
      password: "",
      role: ""
    }
  }
}
</script>


<template>
  <div>
    <input type="text" name="email" placeholder="Email" v-model="email" />
    <input type="password" name="password" placeholder="Password" v-model="password" />
    <select v-model="role">
      <option value="admin">Admin</option>
      <option value="user">User</option>
    </select>
  </div>
</template>
```

Sl. 3.19 - Korištenje *select* elementa.

Prilikom korištenja *checkboxa*, unutar *v-modela* se postavlja varijabla koja može imati *boolean* vrijednost, vrijednost unutar varijable će biti njegova zadana vrijednost. Nije potrebno raditi metodu kojom će se vrijednost varijable promijeniti. Ako imamo više *checkbox*-ova, dodajemo im *value* atribut. Unutar roditelj komponente *select* postavljamo *v-model* (Sl. 3.19). Kako ih budemo označavali, dodavat ćemo ili uklanjati njihove vrijednosti unutar niza.

Ako smo postavili podatke unutar *form* spremnika, možemo koristiti *@submit*. No, prvo što će se dogoditi nakon što „podnesemo“ podatke je osvježavanje stranice. Kako bismo to priječili, moramo upisati *@submit.prevent*.

Nakon što smo izradili stranice, moramo izraditi njihove rute kako bismo ih mogli prikazati. Rute se definiraju unutar *src/router* direktorija, unutar *indeks.js* datoteke. Prije nego što izradimo rutu, moramo uvesti komponentu, tj. stranicu.



Sl. 3.20 - Izgled index.js datoteke unutar koje se definiraju rute za Vue.js stranice.

Pomoću *createRouter* stvaramo *router*. Sa *createWebHistory(import.meta.env.BASE_URL)* stvaramo *router* sa povijesti kako bismo mogli otvoriti prethodne stranice. Svaka ruta mora sadržavati naziv, putanju i komponentu sa kojom je povezana.

Otvaranje drugih stranica možemo postići sa *router-link* elementom. Unutar atributa *to* upisujemo putanju stranice koju slijedeću otvaramo ili možemo poslati objekt koji sadrži svojstvo neke rute, npr. naziv. Vue.js je SPA, što znači da neće slati zahtjev poslužitelju da mu pošalje tu stranicu. No, ako koristimo elemente <*a href*=“”>, on mora poslati zahtjev.

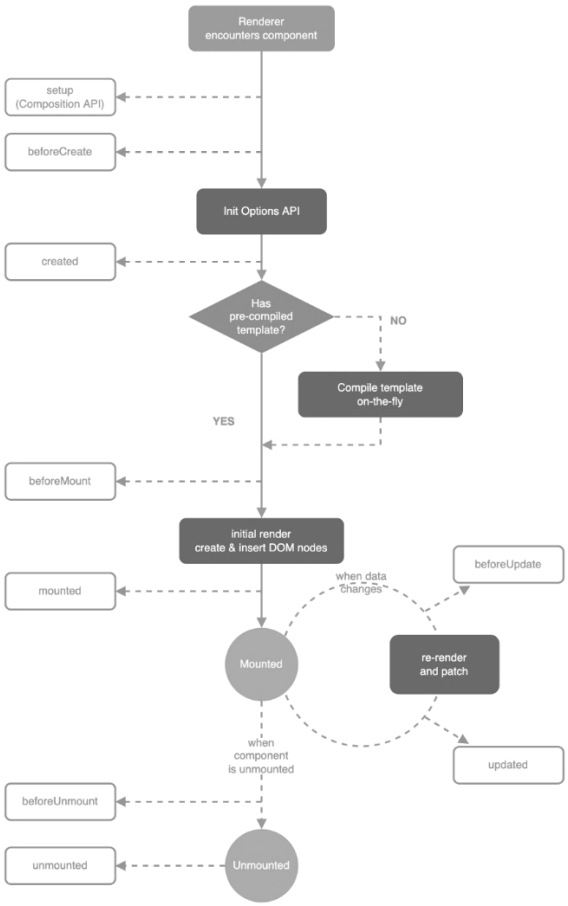
```
<router-link :to="{ name: 'newTask', params: { id: this._id } }"></router-link>
```

Sl. 3.21 - *RouterLink*, umjesto upisivanja putanje unutar atributa, može se postaviti objekt unutar kojeg možemo poslati parametre koje šaljemo.

Kako bismo izbjegli grešku u slučaju da korisnik pokušava pristupiti ruti koja više ne postoji, unutar indeks.js možemo upisati *{path: "/staraRuta", redirect:"/novaRuta"}*. U slučaju da je korisnik upisao nepostojeću rutu, prikazat će se samo ono što se nalazi unutar App.vue. Stoga možemo izraditi stranicu koju ćemo prikazati za svaki krivo upisani URL. Unutar indeks.js-a postavljamo novu rutu koja za putanju ima *"/:catchAll(.*)"* kojom se hvata svaka ruta koja nije unutar te datoteke. Redoslijed ruta nije bitan.

Također možemo preusmjeriti korisnika na prethodne stranice koje je posjetio unutar aplikacije sa: *this.$router.go(1)* ili *this.$router.go(-1)*. Možemo postaviti znak $ prije korištenja *routera* kako ne bismo uvezli cijeli *router* već samo oni potreban dio.

Svaka komponenta ima životni ciklus. Vue.js nam omogućuje da pristupimo tim fazama kroz *lifecycle hooks*. Na primjer, možemo koristiti *mounted hook* unutar kojeg definiramo funkciju, taj hook će se izvesti nakon što se komponenta renderirala i postavila unutar DOM stabla. Ili možemo koristiti *created() hook*, sa kojim možemo dohvatiti podatke prije. Postoje još *destroyed()* i *updated() hook*. Možemo ih uvesti prije korištenja, pa umjesto *mounted()* koristimo *onMounted()*.



Sl. 3.22 – Dijagram životnog ciklusa komponente, izvor: *[12]*

3.9. Options API

Options i *Composition* API-ji su dva različita načina na koja možemo napisati skriptu Vue komponente. *Options* API se najčešće koristi za pisanje skripti jer je jednostavniji. Prije Vue 3 bio je i jednini način pisanja.

Kada koristimo Options API za izradu skripte, koristimo opcije kao što su: *data()*, *methods*, *lifecycle hookove*, izračunata svojstva (engl. *computed properties*) i *state* podaci [13].

4. Izrada web aplikacije

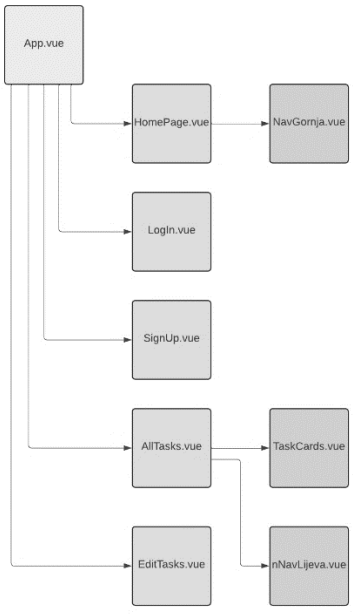
4.1. Ideja web aplikacije

Unutar ovog završnog rada, proći ćemo kroz izradu web aplikacije za male bilješke. Glavne stranice koje će postojati su: Početna, Prijava i Registracija, Bilješke (dovršene i nedovršene) Uređivanje/Izrada bilješke.

Prilikom otvaranja web aplikacije prikazuje se početna stranica na kojoj će biti dva dugmeta: za prijavu i registraciju. Prilikom prijave unosi se adresa korisnika i lozinka. Bit će postavljeno dugme u slučaju da korisnik odabere krivu opciju. Nakon prijave, server izrađuje i šalje token korisnika na stranicu koja prikazuje sve njihove zadatke.

Moguće je pretražiti zadatke, izraditi novi, urediti i obrisati postojeće, pregledati riješene zadatke, filtrirati i označavati ih. Aplikacija će se prilagođavati na temelju veličine prozora.

Kada postavljamo komponente unutar stranica, ta postavljena komponenta je dijete, a ona druga je roditelj. Skupa čine stablo, hijerarhija komponenti. Na početku stabla je korijen, ta komponenta može imati podređene komponente (djecu) koje mogu imati djecu…



Sl. 4.1 - Stablo komponenti za ovaj projekt.

4.2. Smjernice za projekt i izmjene predloška

Radi organizacije: stranice se mogu spremiti unutar *views* direktorija, dok su komponente uvezene unutar stranica možemo spremiti unutar *components* direktorija. Pokretanjem aplikacije i izmjenom nekog elementa, u pregledniku se odmah prikazuje promjena, nije potrebno ručno osvježavanja. No, prilikom izmjene logike, preporuka je ručno osvježavanje.

Svaka komponenta ima dvije riječi unutar svog naziva zbog toga što Vue.js sprječava da imenujemo komponente sa jednom riječi kako ne bi došlo do konflikta unutar koda, kao što sam objasnila prije.

Prilikom uvoza komponente nije potrebno pisati „.vue“, ali je dobra praksa.

Stilove ne trebamo postavljati unutar samih komponenti, možemo ih upisivati unutar base.css ili main.css. Prilikom pokretanja web aplikacije se svi stilovi grupiraju te postaju globalni. No, ako postoje neki specifični stilovi koje želimo primijeniti na određenu komponentu, možemo koristit *style scoped, id* ili *class* atribute. No, moguće je postaviti ih sve unutar jedne datoteke: base.css ili main.css

Za ovaj projekt, obrisani su svi stilovi unutar base.css i main.css, izmijenjen je App.vue i obrisane sve komponente predloška.

Unutar App.vue možemo postaviti neki sadržaj koji želimo da svaka stranica ima. Za ovaj primjer potreban je samo *router-view* , koji postavlja komponentu unutar App.vue, koja odgovara trenutnoj ruti.

4.3. Izrada komponenti

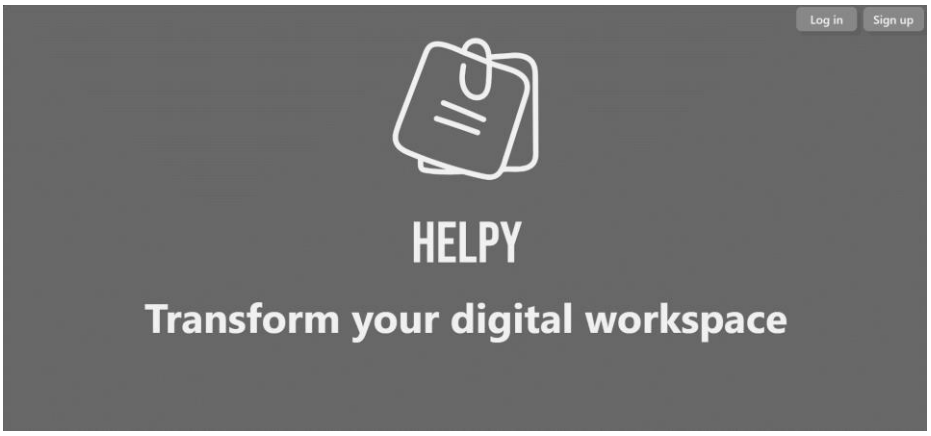
U direktoriju *views* izradit ćemo HomePage.vue. Početna stranica sadržava logo, naslov i gornju navigacijsku traku. Navigacijska traka je spremljena u *components* direktoriju. Svaka komponenta koju želimo prikazati unutar stranice prvo mora biti uvezena u *script* elementu roditelja, navesti je unutar *components* pa postavljena u template elementu. Vue 3 omogućuje korištenje uvezenih komponenti i varijabli automatski, bez definiranja u objektu, tj. unutar *components* elementa. Kako bismo to mogli koristiti moramo navesti ključnu riječ *setup* unutar *script* elementa.

```
<RouterLink to="/login"><button>Log in</button></RouterLink>
<router-link to="/signup"><button>Sign up</button></router-link>
```

Sl. 4.2 – NavGornja.vue sadrži dva dugmeta, za prijavu i registraciju. *RouterLink* se koristi kako bi se prikazale odgovarajuće stranice

Na slici (Sl. 4.2) možemo uočiti da postoje dva načina kako stvoriti vezu sa web stranicom unutar Vue.js aplikacije: *RouterLink* i *router-link*. Zapravo su ista stvar, ali drugačije napisani. *RouterLink* se koristi za navigaciju: kada pritisnemo na taj element, prikazuje se odgovarajuća komponenta sa kojom je povezan, bez ponovnog učitavanja stranice. Unutar tog atributa postavljamo rutu koju želimo prikazati. Rute se izrađuju unutar indeks.js-a, koji se nalaze unutar src/router direktorija.

Za svaku izrađenu stranicu potrebno je napraviti rutu kako bi joj se moglo pristupiti. Svaka ruta mora sadržavati putanju, ime i komponentu koju želimo prikazati. Kada otvorimo aplikaciju, otvorit će se ona stranica koja ima „/“ rutu, u ovom slučaju to je HomePage.vue.



Sl. 4.3 - Izgled početne stranice projekta

Unutar Login.vue i SignUp.vue može se koristiti *form* spremnik koji će sadržavati 2 polja za unos korisnikovih podataka: email i lozinka. Ideja prijave i registracije je ista, zato je izgled identičan. No logika iza njih je različita. Možemo postaviti dugme koje će preusmjeriti korisnika na drugu stranicu za prijavu, u slučaju da je korisnik odabrao krivu rutu.

Sl. 4.4 - Izgled stranice za prijavu

Sl. 4.5 - Izgled komponente za registraciju, SignUp.vue

Kada korisnik pritisne dugme za prijavu, poziva se funkcija koja šalje *GET* zahtjev serveru. Kod SignUp.vue komponente potrebno je izraditi korisnika, za što se koristi *POST* zahtjev.

Unutar servera provjera poslane vrijednosti te postoji li korisnik, te ako su uvjeti zadovoljeni: šalje token koji se šalje komponenti AllTasks.vue. Token se provjerava odmah nakon što je stranica postavljena unutar DOM stabla.

Umjesto korištenja fetch API-ja, možemo koristiti axios koji, ako primi odgovor servera sa statusom od 400 do 599, smatrat će greškom. Fetch te odgovore ne smatra greškama jer je server uistinu odgovorio, iako to nije odgovor kojim smo se nadali. [14]

S obzirom da za prijavu koristimo *GET* zahtjev, kako bismo poslali podatke možemo koristiti upitni (engl. *query*) parametre. Kako bismo označili početak parametara, unutar URL-a se postavlja „?“. Oni se pišu u obliku „ključ=vrijednost“ te se odvajaju sa &.

```
(`http://localhost:3000/korisnici/login?email=${this.email}&password=${this.password}`)
```

Sl. 4.6 - *Query* parametri unutar URL-a

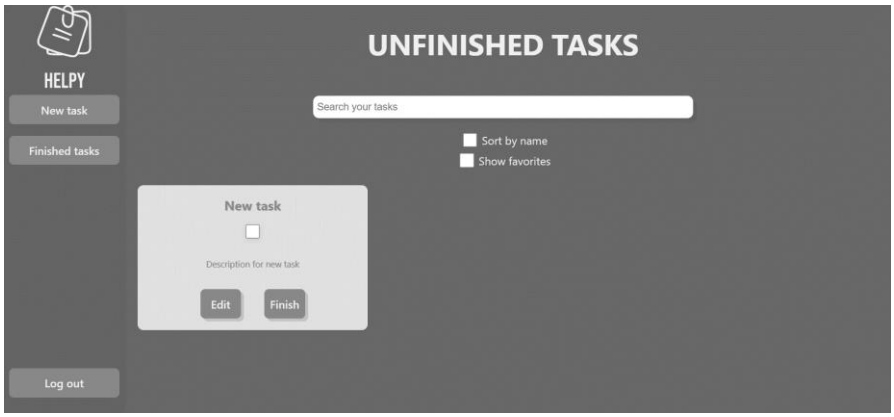
Poslani podaci se provjeravaju na poslužitelju, a u slučaju greške prikazat će se *alert box* s opisom greške. U slučaju da nije bilo greške, novoizrađen token se šalje slijedećoj stranici.

Nakon uspješne prijave, otvara se AllTasks.vue, stranica za prikaz svih dovršenih / nedovršenih zadataka. Prilikom prikaza stranice, poziva se funkcija koja prima sve korisnikove zadatke. Stranica sadrži komponente NavLijeva.vue i TaskCards.vue. Ova stranica prima token te provjerava njegovu postojanost i valjanost, u protivnom usmjerava korisnika na login stranicu.

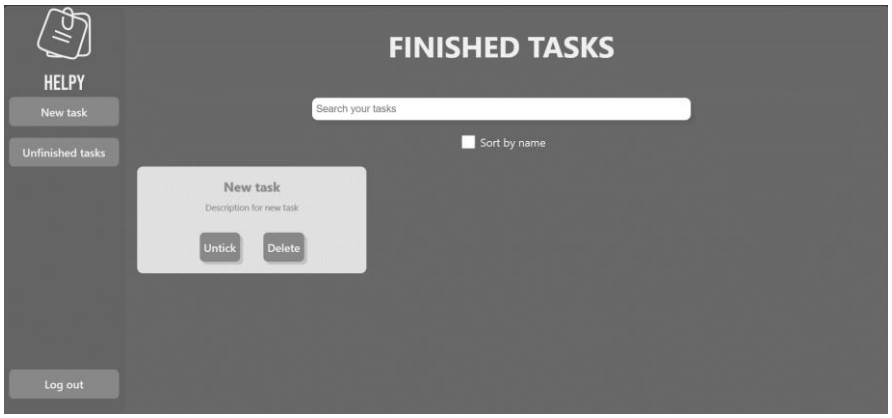
Nedovršeni zadaci se mogu uređivati ili završiti, zadaci se brišu tek onda kada su dovršeni. Završene zadatke možemo vidjeti pritiskom na dugme s oznakom „*Finished tasks*“ koji se nalazi na lijevoj navigacijskoj traci. Kada pritisnemo to dugme, mijenja se vrijednost *boolean* varijable što poziva funkciju za prikaz zadataka i mijenja vrijednosti varijabli unutar *computed* dijela. Ako postoje varijable čija vrijednost ovisi o drugim varijablama, možemo ih postaviti unutar *computed* dio.

Pomoću *window.history.state* možemo poslati potrebne podatke stranicama. To omogućuje korisnicima da prolaze kroz povijest preglednika i da ne šaljem token pri svakom prikazu ove stranice.

Ova stranica je napravljena da prikazuje sve zadatke, dovršene i nedovršene, ovisno o *boolean* varijabli naziva *isFinished*. Izmjena ove varijable pokreće izmjenu drugih varijabli te prikaz drugih zadataka.



Sl. 4.7 - Izgled AllTasks.vue unutar preglednika, izgled kada je varijabla *isFinished* jednaka *false*



Sl. 4.8 - Izgled AllTasks.vue kada je varijabla *isFinished=true*

Unutar ove stranice moguće je filtrirati i sortirati po nazivu te prikazati označene zadatke. Filtrirati i sortirati se mogu dovršeni i nedovršeni zadaci, no označavati se mogu samo nedovršeni. Kada se zadatak dovrši, on više neće biti označen. Kod filtriranja najbolje je koristiti *@input* kako bi se funkcija za filtriranje zadataka pozivala nakon svake izmjene. Za sortiranje i prikaz označenih zadataka koristit ćemo *checkbox* i *@change* koji se poziva nakon svake promjene unutar *checkboxa*. *Checkbox* automatski mijenja vrijednost varijable ovisno o tome je li označen. Kako bi se mogli prikazati zadaci ovisno o stanju unutar *checkboxa*, moguće je postaviti podatke unutar dva niza: sortirani i nesortirani podaci.

Komponenti NavLijeva potrebno je poslati *boolean* vrijednost kojom se mijenja vrijednosti varijabli unutar stranice AllTasks.vue, kao što su oznake i prikazani zadaci. Ona sadrži dugme za izradu novog zadatka, tako da joj je potrebno proslijediti i token. Moguće se pokrenuti funkciju unutar roditelja iz uvezene komponente pomoću *this.$emit()*.

Kako bi se prikazali svi zadaci, unutar komponente za kartice se unosi *v-for="item in tasks_list"*. Na taj način šalju se podaci svakog zadatka toj komponenti. Ovisno o varijabli isFinished mijenjaju se oznake i funkcionalnosti na karticama.

Sl. 4.9 - Nedovršen zadatak, oznake na dugmadi su „*Edit*“ i „*Finish*“

Zadaci se mogu označiti samo ako su nedovršeni. Ako je dovršen, vrijednost te varijable se postavlja na zadanu vrijednost, *false*.

Sl. 4.10 - Dovršen zadatak sa oznakama na dugmadi „*Untick*“ i „*Delete*“

Svaka kartica ima 2 funkcionalnosti: uređivanje i brisanje. Obrisati se može samo onaj zadatak koji je završen. Uređivati se mogu naslov i opis, te odabrati je li zadatak dovršen ili ne.

Stranica EditTasks.vue se poziva ako se izrađuje novi zadatak ili uređuje postojeći. Ako se uređuje, postavljaju se podaci zadatka unutar input elemenata, za lakše uređivanje. Izgled je identičan LogIn.vue i SignUp.vue komponentama, no logika je drugačija.

Stranici kao što je EditTask.vue je potreban token jer sadrži *id* korisnika, a ako je poslan onda je potreban i naslov i opis. Stranica EditTasks.vue je izrađena tako da se unutar nje mogu izrađivati ili uređivati zadaci, u protivnom bi postojale stranice sa istim stilom i izgledom, a sličnom logikom. Stoga ovisno o poslanim podacima, stranica se prilagođava: ako je poslan *id* zadatka znači da ga uređujemo, dok odsutnost *id*-a označava da ne postoji.

Zadaci imaju ograničenje: naziv može biti do 20 znakova dug i ne smije biti prazan, dok opis zadatka može biti do 100 znakova dug i može biti prazan. Korisnici mogu uređivati naslov i opis samo nedovršenim zadacima. Moguće je izići iz stranice klikom na dugme s oznakom „*Cancel*“. S obzirom da se ova komponenta koristi za dodavanje ili izmjenu zadatka, možemo postaviti oznake elemenata, npr. naslov ili dugme, unutar varijabli tako da ih možemo izmijeniti.

Sl. 4.11 - Izgled EditTasks.vue u slučaju da je nije poslan *id.*

Sl. 4.12 - Izgled EditTasks u slučaju da je poslan *id* zadatka.

Ako korisnik upisuje krivi URL, na stranici će se prikazati sve što se nalazi unutar App.vue, a s obzirom da je ta komponenta samo sadrži *router-view* neće prikazati ništa. Ovdje postoje dvije opcije: možemo ga preusmjeriti na drugu stranicu ili izraditi i prikazati stranicu sa dugmetom za povratak.

4.4. Izrada servera

Server možemo postaviti unutar novog direktorija koji se nalazi izvan src direktorija. Sa naredbom *npm init -y* izradit ćemo novi package.json koji sadrži sve bitne informacije o serveru. Nakon preuzimanja paketa potrebnih za server, možemo ih vidjeti unutar te datoteke. Unutar naredbe možemo upisati *–y* kako bi označili da želimo projekt sa zadanim vrijednostima [15]. Unutar server.js datoteke definirat ćemo sve rute potrebne za ovu aplikaciju.

```
import { createRequire } from 'module';
const require = createRequire(import.meta.url);
```

Sl. 4.13 – Kod koji omogućava korištenje zahtjeva CommonJS u ES modulu.

Unutar package.json možemo upisait *„type":"module"* i pri početku servera postaviti kod na slici (Sl. 4.13). Ovo je potrebno kako bi se uvezli svi potrebni elementi bez greške, inače ćemo dobiti greške prilikom korištenja riječi „require".

Unutar ovog servera, korišteni su paketi: Express.js, CORS, Nodemon, Bcrypt, JSON web token i dotenv pomoću naredbe „npm install nazivPaketa". Pomoću riječi require se uključuju unutar servera.

Express.js je Node.js *open-source* okvir koji pojednostavljuje izradu poslužitelja tako što omogućuje izradu *middlewarea*, usmjerava zahtjeve na određene metode, omogućava definiranje rute za rukovanje HTTP zahtjevima.

CORS ili puni naziv *Cross-Origin Resource Sharing*, je sigurnosna značajka koja omogućuje preglednicima korištenje podataka s drugih stranica i određuje kako im se može pristupiti. [16]

Nodemon omogućuje da se poslužitelj prilikom svake izmjene, automatski ponovno pokrene.

Bcrypt je kripografska *hash* funkcija s kojom možemo *hashirati* osjetljive podatke tako da im je teže pristupiti ako ikada „procure". Prije *hashiranja* lozinki, potrebno je odrediti parametar koji se zove *rounds*. On određuje trošak i vrijeme potrebno za izračunati tu *hash* vrijednost..

JSON Web Token ili skraćeno JWT, je JSON objekt pomoću kojeg možemo na jednostavan i siguran način prenijeti bitne informacije između slojeva [17]. Sastoji se od zaglavlja, *payload* i potpisa. Pomoću Express.js možemo stvoriti *middleware* pomoću kojeg ćemo provjeriti token prije svake rute.

Dotenv omogućuje učitavanje .env datoteke u *process.env* [18] kako bismo im mogli pristupiti. Kako bismo izbjegli neovlašten pristup, unutar njih postavljamo osjetljive podatke, npr.: *port* servera, adresa baze ili ključ kojim (de)šifriramo token. Možemo unutar .env datoteke odrediti koji podaci će se koristiti ovisno o fazi aplikacije [14]. Svi nazivi .env varijabli moraju sadržavati samo velika slova.

Možemo postaviti *middleware* koji će se pozvati prije svakog poziva rute; inače bismo je trebali pozivati unutar svake rute. Pozivamo ih pri početku servera sa *app.use(nazivFunkcije)*. *Middleware* koje želimo pozvati su: cors i express json. Funkcija express.json će pretvarati sve

podatke unutar tijela zahtjeva u JSON format tako da ih ne trebamo pretvarati unutar *front-end* dijela.

MongoDB je NoSQL, *document-oriented* baza podataka. Ova baza podataka sprema dokument u obliku JSON objekta, zvan BSON, tj. binaran JSON. Unutar njega možemo spremati različite vrste podataka, kao što su: nizovi, *stringovi*, brojevi ili *boolean*.

Mongoose omogućuje korištenje operacija pomoću kojih možemo dobiti, izraditi, urediti ili obrisati elemente unutar baze podataka; CRUD operacije.

Kako bismo mogli koristiti MongoDB unutar web aplikacije, u direktoriju u kojem se nalazi server, moramo instalirati Mongoose (*Object Data Modeling*) biblioteku koja će se koristiti za interakciju sa MongoDB-om.

Pomoću Mongoose paketa možemo napraviti strukturu dokumenata u kojoj određujemo svojstva podataka. Nakon što smo napravili shemu, sa *mongoose.model()* stvaramo model. Prvi argument je naziv modela, nakon toga shema pa treći, opcionalan, je pluralan naziv modela. U slučaju da unutar MongoDB već postoje ovi modeli, ponovno definiranje neće izbrisati postojeće podatke. Razlog zašto radimo model je u tome što nam je shema samo „nacrt“, pravila kojih se svaki podatak treba držati. Pomoću modela možemo instancirati podatke.

```
const { Schema } = mongoose;

const korisnikShema = new Schema({
    email: { type: String, required: true, unique: true },
    password: { type: String },
});

const zadatakShema = new Schema({
    korisnikId: { type: String, required: true },
    naslov: { type: String, required: true },
    opis: { type: String },
    zavrsen: { type: Boolean, required: true, default: false }
})

const Zadatak = mongoose.model("Zadatak", zadatakShema, "Zadaci");
const Korisnik = mongoose.model("Korisnik", korisnikShema, "Korisnici");
```

Sl. 4.14 - Izrada dvaju shema, strukture podataka.

```
mongoose.connect(process.env.ADRESA_BAZE);
const db = mongoose.connection;

db.on("error", error => {
    console.error("Greška pri spajanju:", error);
});
db.once("open", function () {
    console.log("Spojeni smo na MongoDB bazu");
});
```

Sl. 4.15 – Povezivanje na bazu te logika osluškivanja događaja.

Sa ovakvom bazom, *id* korisnika i zadataka će generirati MongoDB. Kao mjera opreza, prilikom izmijene podataka, bilo to uređivanje ili brisanje, možemo provjeriti sadrži li korisnik taj zadatak.

Nakon povezivanja s bazom, možemo izraditi *middleware*-e koje ćemo pozivati unutar određenih unutar ruta. Oni su: provjeriToken, provjeriKorisnika te provjeriZadatak. *Middleware* provjeriKorisnika se poziva prilikom prijave ili registracije, kako bi se provjerila veličina podataka koji su poslani. Isto vrijedi i za provjeriZadatak, samo što se poziva na rutama gdje se dodaje novi ili izmjenjuje postojeći zadatak. *Middleware* provjeriToken se poziva u svim rutama koje imaju veze sa zadacima: provjerava je li token, koji je poslan u zaglavlju, važeći.

```
app.post('/korisnici/signup', provjeriKorisnika, async (req, res) => {
    try {
        const korisnikBaza = await Korisnik.findOne({ email: req.query.email });
        if (korisnikBaza == null) {
            const hashLozinka = await bcrypt.hash(req.query.password, saltRunde);
            const noviKorisnik = new Korisnik({ ...req.query, password: hashLozinka });
            await noviKorisnik.save();

            const token = jwt.sign({ idKorisnika: noviKorisnik._id }, process.env.TAJNI_KLJUC, { expiresIn: "1h" });
            res.status(201).json(token);
        }
        else res.status(406).send("Email is already in use.");
    }
    catch (error) { res.status(500).send(error.message); }
})
```

Sl. 4.16 – Primjer rute za izradu novog korisnika.

Na slici (Sl. 4.16) možemo vidjeti jednu od mnogih ruta korištenih u ovom projektu. Korištenjem *try* i *catch* bloka, uhvatit ćemo bilo koju neočekivanu grešku. Jedna od CRUD operacija korištenim je *findOne()* koja, kako joj i ime nalaže, traži samo jedan podatak koji sadrži poslana svojstva. U ovom slučaju to je email poslan unutar tijela zahtjeva. Traži se samo jedan zato što nije moguće izraditi više od jednog računa sa istim emailom jer unutar sheme to polje je jedinstveno. Ako korisnik pokuša izraditi račun sa emailom koji se koristi, MongoDB neće dozvoliti spremanje tih podataka. Ako ne postoji niti jedna osoba sa tim emailom, *hashira* se lozinka. Email i *hashirana* lozinka spremaju unutar baze u kojoj se generira *id* koji. Nakon toga se šalje novoizrađen token nazad pregledniku.

Unutar rute za provjeru postojećih korisnika, prvo se poziva *middleware* u kojem se provjerava duljina poslanih podataka. Nakon toga provjerava se postoji li korisnik sa tim emailom i ako postoji, jesu li im lozinke iste. Ako jesu, izrađuje se i šalje token pregledniku. Unutar tokena se sprema korisnikov *id*, kojeg je generirao MongoDB.

Token je potreban stranici AllTasks.vue koja će prikazati sve zadatke koje je prijavljeni korisnik izradio. Prije nego što se pošalju svi njegovi zadaci, taj token se mora provjeriti i dešifrirati. Ovo je bitno jer bez korisnikovog *id*-a neće se moći prikupiti svi njegovi zadaci.

Nakon prijave, korisnik može izraditi, izmijeniti i obrisati zadatke. Prilikom izrade zadatka, provjeravamo token korisnika i duljinu poslanih vrijednosti. Moguće je postaviti ograničenje na duljinu podatka prilikom izrade sheme. Token se šalje jednom, prilikom prijave. U slučaju da je token istekao, korisnika se vraća stranicu za prijavu. Nije moguće dodavati, uređivati ili brisati zadatke sa takvim tokenom. Prilikom izrade, uređivanja ili brisanja, koristim se CRUD operacija: *find(), save(), findByIdAndUpdate(), findByIdAndDelete()*.

Na kraju, moramo postaviti server na port preko kojeg će osluškivati zahtjeve web aplikacije. Port je spremljen unutar .env datoteke, a toj varijabli pristupamo preko *process.env*". Prilikom odabira porta, bitno je za napomenuti da *portovi* od 0 do 1023 se ne smiju koristiti. Ako se *port* koji smo odabrali koristi, potrebno je izmijeniti .env varijablu, inače se server neće moći pokrenuti.

Kako ne bismo dobili greške pri korištenju *process.env*, preuzet ćemo ESLint pomoću naredbe *npx eslint –init*.

Potrebno je izmijeniti liniju unutar datoteke eslint.config.js kako bih ESLint prepoznao da se radi o serveru u Node.js okruženu i da se koriste njegove globalne varijable.

```
import globals from "globals";
import pluginJs from "@eslint/js";
import pluginVue from "eslint-plugin-vue";

export default [
  { files: ["**/*.{js,mjs,cjs,vue}"] },
  { files: ["**/*.js"], languageOptions: { sourceType: "commonjs" } },
  { languageOptions: { globals: { ...globals.browser, ...globals.node } } },
  pluginJs.configs.recommended,
  ...pluginVue.configs["flat/essential"],
];
```

Sl. 4.17 - Izgled eslint.config.js, dio linije koji je podcrtan je izmijenjen

Zaključak

Vue.js se pokazao kao jedan dobar i pouzdan okvir. Brzina prikaza stranica u Vue.js aplikaciji je iznenađujuća, no to je i očekivano s obzirom da je riječ o *Single Page* aplikaciji. Vrlo je fleksibilan. Moguće je pisati sav relevantan kod unutar jedne datoteke, no moguće je i pisanje u zasebnim datotekama. Moguće je pisati logiku komponente sa JS, JSX ili TypeScriptom.

Okvir ima određene restrikcije poput one vezane za naziv komponente, na taj način pomažu početnicima da izbjegnu trivijalne greške. Također, pozitivna strana ovog okvira je što pri izradi web aplikacije se mogu birati neki dodaci za aplikaciju umjesto automatskog preuzimanja bez programerovog znanja. Budući da nudi alate prije izrade predloška, programer može znati što ovaj okvir nudi i birati želi li predložen alat unutar aplikacije ili neki drugi.

Dolaskom Vue 3, ovaj okvir omogućuje novi načina pisanja logike. Moguće je koristiti i stari i novi način unutar jednog projekta, no samo ako koristimo Vue 3. *Options API*, originalni način za izradu logike je jednostavan za naučiti, intuitivan, no preporuča se za male komponente. Za razliku od njega, *Composition API* omogućuje bolju organizaciju i jednostavnije dijeljenje logike komponente. Velika je razlika između njih, novi način je uveden zbog stvari koje originalan način nije omogućavao.

Jedina zamjerka je što, ako se izmijeni logika komponente, aplikacija se mora ručno osvježiti. No, ako se izmijeni struktura ili stil komponente, automatski se prikaže.

Literatura

[1] S.-H. Lim, »Experimental Comparison of Hybrid and Native Applications for Mobile Systems,« *International Journal of Multimedia and Ubiquitous Engineering,* 2015.

[2] B. Messenlehner i J. Coleman, Building web apps with WordPress : WordPress as an application framework, Sebastopol: O'Reilly Media, 2019.

[3] »What is a Framework?,« GeeksforGeeks, 10. Svibanj 2024.. [Mrežno]. Available: https://www.geeksforgeeks.org/what-is-a-framework/.

[4] »Software Framework vs Library,« GeeksforGeeks, 19. Rujan 2023.. [Mrežno]. Available: https://www.geeksforgeeks.org/software-framework-vs-library/.

[5] L. S. Vailshery, »Most used web frameworks among developers worldwide, as of 2024,« Statista, 7. Kolovoz 2024.. [Mrežno]. Available: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/. [Pokušaj pristupa 31. Kolovoz 2024.].

[6] I. Bodrov-Krukowski, M. M, B. Houwens, J. Aden, C. Ribeiro, K. Kolev, J. Wilken, M. Wanyoike, A. Bouchefra, V. Softic i D. Aden, Learn Angular : your first week, Collingwood: SitePoint Pty. Ltd., 2018..

[7] B. Dayley, B. Dayley i C. Dayley, Learning Angular : a hands-on guide to Angular 2 and Angular 4, Addison-Wesley Professional, 2018..

[8] M. Schwarzmüller, React Key Concepts, Birmingham: Packt Publishing, 2022..

[9] »SPA (Single-page application),« [Mrežno]. Available: https://developer.mozilla.org/en-US/docs/Glossary/SPA. [Pokušaj pristupa 11. Kolovoz 2024.].

[10] »Vite - Getting Started,« Vite, [Mrežno]. Available: https://vitejs.dev/guide/. [Pokušaj pristupa 1. Rujan 2024].

[11] »Installation,« Vue.js, [Mrežno]. Available: https://vueframework.com/guide/installation.html. [Pokušaj pristupa 4. Kolovoz 2024.].

[12] »Lifecycle Hooks,« Vue.js, [Mrežno]. Available: https://vuejs.org/guide/essentials/lifecycle.html. [Pokušaj pristupa 30. Kolovoz 2024.].

[13] C. Minnick, JavaScript all-in-one, Hoboken: John Wiley & Sons, Inc.,, 2023. .

[14] G. Zaharija, »Okviri i alati za razvoj web aplikacija,« 2023.. [Mrežno]. Available: https://pmfst-oarwa.netlify.app. [Pokušaj pristupa 14. Kolovoz. 2024.].

[15] A. D. Scott, JAVASCRIPT COOKBOOK programming the web, [S.l]: O'REILLY MEDIA, 2021..

[16] »Use of CORS in Node.js,« GeeksforGeeks, 12. Lipanj 2024.. [Mrežno]. Available: https://www.geeksforgeeks.org/use-of-cors-in-node-js/.

[17] »Introduction to JSON Web Tokens,« GeeksforGeeks, [Mrežno]. Available: https://jwt.io/introduction. [Pokušaj pristupa 18. Kolovoz 2024.].

[18] »dotenv,« Veljača 2024. [Mrežno]. Available: https://www.npmjs.com/package/dotenv. [Pokušaj pristupa 21. Kolovoz 2024.].

Sažetak

U ovom završnom radu analizirat ću različite tipove aplikacija, a to su: nativne, hibridne i web aplikacije. Usporedit ću različite okvire za razvoj web aplikacija, detaljno opisati njihove prednosti te objasniti zašto je upravo Vue.js odabran kao okvir za izradu web aplikacije za male bilješke. Detaljno ću prikazati sintaksu Vue.js-a i sve njegove pogodnosti. Prikazat ću proces izrade aplikacije za bilješke, zajedno s razvojem poslužitelja i korištenjem dodataka koji olakšavaju razvoj web aplikacija.

Summary

In this final paper, I will analyze different types of applications, namely: native, hybrid and web applications. I will compare different web application development frameworks, detail their advantages, and explain why Vue.js was chosen as the web application framework for small notes. I will show in detail the syntax of Vue.js and all its benefits. I'll show the process of building a notes application, along with developing a server and using plugins that make web application development easier.