Usporedba frontend okvira za razvoj web aplikacija

Nenadović, Kristian

SVEUČILIŠTE U SPLITU

PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

USPOREDBA FRONTEND OKVIRA ZA RAZVOJ WEB APLIKACIJA

Kristian Nenadović

Split, rujan 2021.

SVEUČILIŠTE U SPLITU

PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

USPOREDBA FRONTEND OKVIRA ZA RAZVOJ WEB APLIKACIJA

Kristian Nenadović

Mentor:

dr. sc. Goran Zaharija

Split, rujan 2021.

Temeljna dokumentacijska kratica

Završni rad

Sveučilište u Splitu

Prirodoslovno-matematički fakultet

Odjel za informatiku

Ruđera Boškovića 33, 21000 Split, Hrvatska

USPOREDBA FRONTEND OKVIRA ZA RAZVOJ WEB APLIKACIJA

Kristian Nenadović

SAŽETAK

U ovom završnom radu objašnjen je pojam modernih web aplikacija te koncept Single Page Application (SPA). Frontend područje se unazad posljednjih godina razvija velikom brzinom, a za to su djelom odgovorni JavaScript okviri. Stoga je tema ovog rada fokusirana na usporedbu i funkcionalnosti koje oni pružaju pri razvoju web aplikacija. Napravljen je pregled Angular, React i Vue okvira te su istaknute njihove prednosti i mane. U zadnjem poglavlju prikazana je web aplikacija izrađena u Vue okviru koja služi kao demonstracija značajki okvira.

Ključne riječi: web aplikacija, JavaScript, okvir, Angular, React, Vue

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

Rad sadrži: 32 stranica, 21 grafičkih prikaza, 0 tablica i 12 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: **Dr. sc. Goran Zaharija,** *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Ocjenjivači: **Dr. sc. Goran Zaharija,** *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Dr. sc. Saša Mladenović, *izvanredni profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Dr. sc. Divna Krpan, *viši predavač Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Rad prihvaćen: rujan 2021.

Basic documentation card

Thesis

University of Split

Faculty of Science

Department of informatics

Ruđera Boškovića 33, 21000 Split, Croatia

COMPARISON OF FRONTEND FRAMEWORKS FOR WEB APPLICATION DEVELOPMENT

Kristian Nenadović

ABSTRACT

In this thesis, the concept of modern web applications and the concept of Single Page Application (SPA) are explained. The Frontend field has been evolving at a rapid pace in recent years, and JavaScript frameworks are partly responsible for this. Therefore, the topic of this paper is focused on the comparison and the functionalities they provide in the development of web applications. An overview of the Angular, React and Vue frameworks was made and their advantages and disadvantages were highlighted. The last chapter shows a web application created in the Vue framework that serves as a demonstration of its features.

Key words: web application, JavaScript, framework, Angular, React, Vue

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

Thesis consists of: 32 pages, 21 figures, 0 tables and 12 references

Original language: Croatian

Mentor: **Goran Zaharija, Ph.D.** *Assistant Professor of Faculty of Science, University of Split*

Reviewers: **Goran Zaharija, Ph.D.** *Assistant Professor of Faculty of Science, University of Split*

Saša Mladenović, Ph.D. *Associate Professor of Faculty of Science, University of Split*

Divna Krpan, Ph.D. *Senior Lecturer of Faculty of Science, University of Split*

Thesis accepted: September 2021.

Sadržaj

UVOD

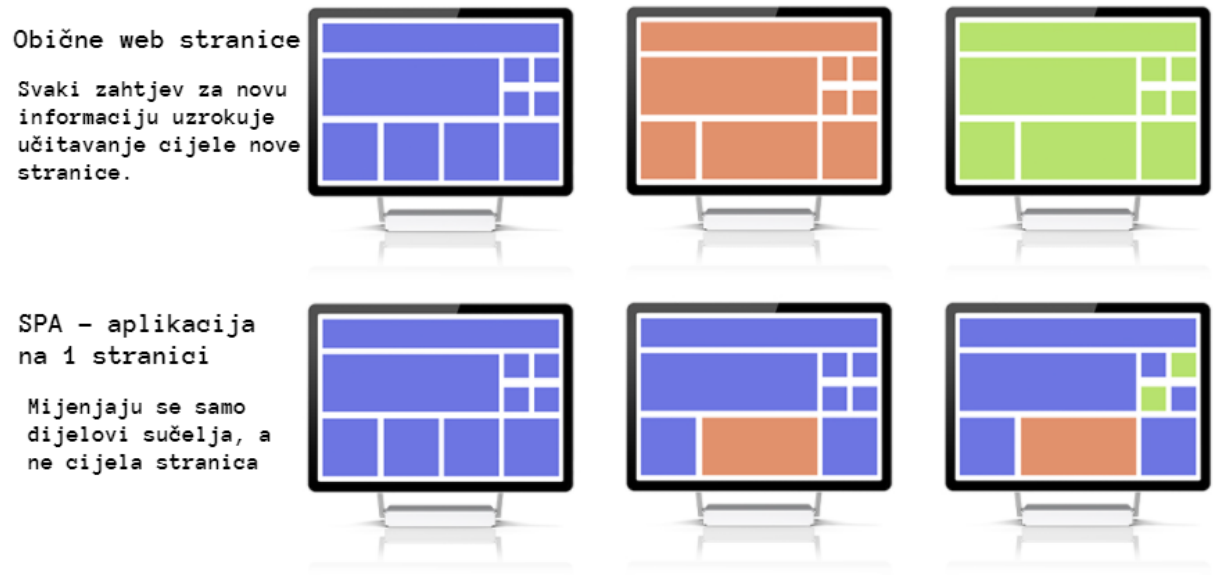
Posljednjih godina dizajn i razvoj web aplikacija je uveliko napredovao i evoluirao. Stvorene su nove potrebe ljudi i načini korištenja tehnologije što je za sobom povuklo i veću kompleksnost za programere. Frontend programeri, često referirani kao programeri klijentske strane, zaduženi su za različite tipove posla vezane uz interakciju korisnika sa sučeljem web aplikacije, od samog izgleda i prezentacije podataka pa sve do funkcionalnosti iste.

Svaku web stranicu čine tri različita sloja koja su međusobno povezana, a to su: strukturni sloj koji čini sam sadržaj (HTML) stranice, prezentacijski sloj zadužen za stil (CSS) te sloj koji korisniku omogućuje interakciju za što je odgovoran skriptni jezik JavaScript. Prijava korisnika, navigacija različitim opcijama, prijenos informacija te razne druge akcije zahtijevale su poprilično puno vremena za implementaciju, a također zbog velike količine ponavljajućeg Javascript kôda bilo je otežano i održavanje. Kako bi se doskočilo tom problemu, počeli su se razvijati frontend okviri (eng. frameworks) kojima su se proširile mogućnosti JavaScript jezika te olakšalo kreiranje elemenata aplikacije. Odabir najboljeg okvira za rješavanje specifičnih problema unutar projekta osigurava brži razvoj i bolje iskustvo korisnika, ali i programera.

Cilj ovoga rada je prikazati osnovne principe frontend okvira, na koji način funkcioniraju i koje mogućnosti pružaju programerima u odnosu na običan JavaScript. Proučiti će se značajke koje bi se trebale uzeti u obzir prilikom odabira frontend okvira i zbog čega su u posljednje vrijeme postali standard za razvoj modernih web aplikacija. U posljednjem dijelu rada razvijena je i aplikacija koja služi za demonstraciju opisanih koncepata.

1 WEB APLIKACIJE

Pod web aplikacijom smatra se program pohranjen na udaljenom serveru kojemu korisnik pristupa preko preglednika (eng. browsera) uz aktivnu internetsku vezu [1]. Razlog zašto su one u posljednje vrijeme postale trend je zbog toga što korisnicima pružaju interakciju, za razliku od običnih statičnih web stranica s kojih se mogao samo čitati sadržaj (slika br. 1). Aplikacije na jednoj stranici (eng. Single Page Application - SPA) kao što su Facebook, Gmail ili Twitter primjeri su web aplikacija koje uz svoju raznovrsnost pružaju ugodno korisničko iskustvo i brzinu korištenja kao da je riječ o lokalnim desktop aplikacijama. Također su pogodne zbog toga što ne zahtijevaju mnogo procesorske snage poput nekih lokalnih aplikacija te su kompatibilne za različite preglednike što ih čini široko primjenjivima. Ideja SPA je da se cijela aplikacija učita u preglednik odmah pri prvom pokretanju, a onda ovisno o informacijama koje korisnik zatraži, sadržaj stranice se mijenja. Time se postiže osjećaj interaktivnosti, odnosno sadržaj nestaje i kontinuirano se zamjenjuje novim, a sve to bez ponovnog učitavanja. Uzmimo na primjer Gmail, gdje se prilikom učitavanja aplikacije prikazuje početna stranica pristigle pošte, no ukoliko korisnik pritisne na neku opciju u bočnoj navigacijskoj traci, URL (eng. Uniform Resource Locator) adresa i sadržaj stranice će se promijeniti, no neće se slati zahtjev na server. Sam taj proces gdje se URL i prikaz aplikacije mijenjaju, a da se zapravo ostaje na istoj stranici naziva se preusmjeravanje, o čemu će se detaljnije u nastavku ovoga rada.



Slika 1 Razlika izmjene sadržaja tradicionalnih i modernih web aplikacija

2 OPĆENITO O FRONTEND OKVIRIMA

Bitno je naglasiti da učenje jednog okvira nije samo učenje tog specifičnog već je to učenje zajedničkih koncepata, arhitektura SPA te dizajniranja i stvaranja komponenti. Također, dobiva se uvid kako podaci putuju kroz aplikaciju, kako se upravlja stanjem i povezuju podaci (eng. data binding), kako se odvija upravljanje rutama sa klijentske strane te komunikacija s backend API (eng. Application Programming Interface).

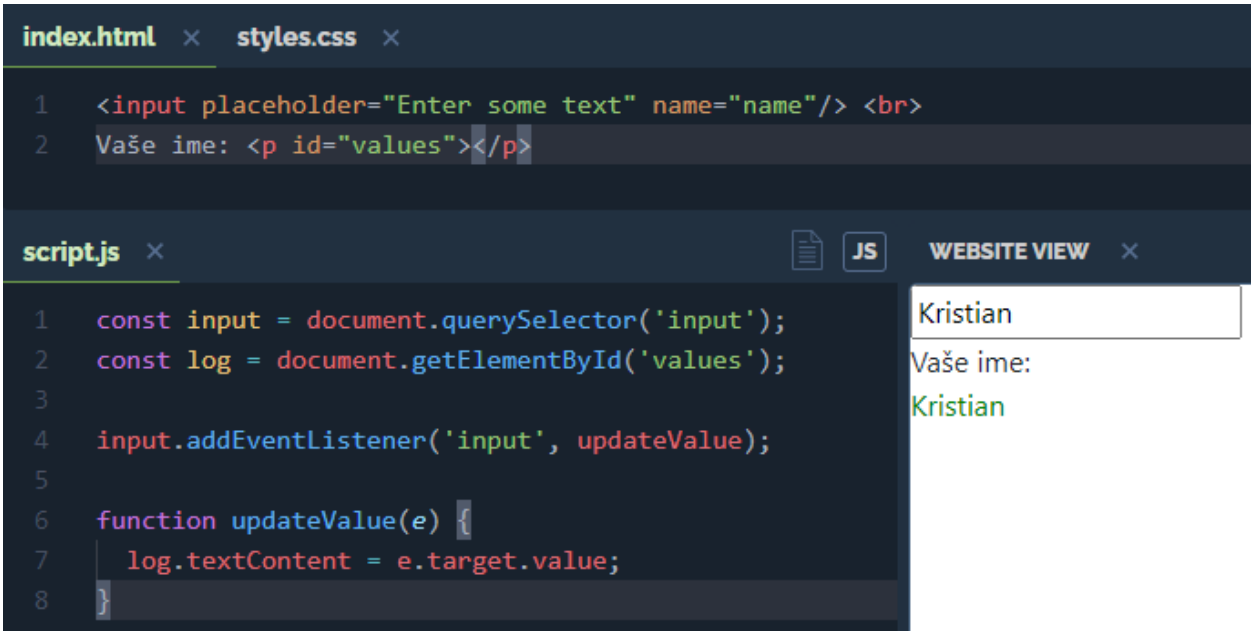
Direktan rad sa DOM-om (eng. Document Object Model) zahtjeva veliko razumijevanje njegovoga funkcioniranja: kako se kreiraju HTML elementi i mijenjaju njihova svojstva, kako se ugrađuju stilovi, kako se grupiraju elementi jedni u druge i postavljaju na web stranicu da čine cjelinu, te u konačnici kako se upravlja interakcijama korisnika preko HTML elemenata. JavaScript okviri su stvoreni upravo iz toga razloga kako bi se sav taj proces učinio jednostavnijim i pogodnijim za programere [1].

Naime, JavaScript programeri primijetili su kako se prilikom razvoja web aplikacije mnogo problema ponavljalo pa se stoga razvio sistem upotrebljivih paketa – knjižnica (eng. libraries), što je znatno ubrzalo razvoj Weba. Knjižnice predstavljaju skup napisanog kôda, koje programeri svojevoljno mogu upotrebljavati i implementirati u svojim projektima kako bi lakše izgradili optimalna korisnička sučelja. Dakle, one predstavljaju već gotove komponente koje se mogu ponovno koristiti, no bitno je naglasiti da one nisu isto što i okviri. Naime, pri korištenju knjižnica prilikom izrade web aplikacije, programer je taj koji ima kontrolu kako i kada će se koristiti i pozivati određene funkcije, dok kod okvira postoje pravila i striktnija struktura kako oblikovati kôd. Programer mora pratiti kontrolu okvira kako se aplikacija razvija i dizajnira, čime se postiže homogenost i lakoća održavanja. Isto tako, time je osigurano i intuitivnije nadograđivanje i skaliranje aplikacije te veći broj ugrađenih značajki i mogućnosti za bolje korisničko iskustvo.

Sam početak Weba započeo je izradom statičkih stranica koje su korisnicima služile samo za čitanje, no pojavom JavaScripta počele su se razvijati dinamične stranice koje su danas ujedno i standard za web aplikacije. Moć atraktivnog i interaktivnog dizajna web aplikacija iskorištavaju i mnoga velika poduzeća koja ih koriste kako bi privukli kupce i pružili informacije o samoj organizaciji i proizvodima koje ona nudi. Jedne od vodećih i vrlo profitabilnih usluga danas na internetu su online trgovine (eng. e-commerce) koje zahtijevaju ugodnu interakciju korisnika jer

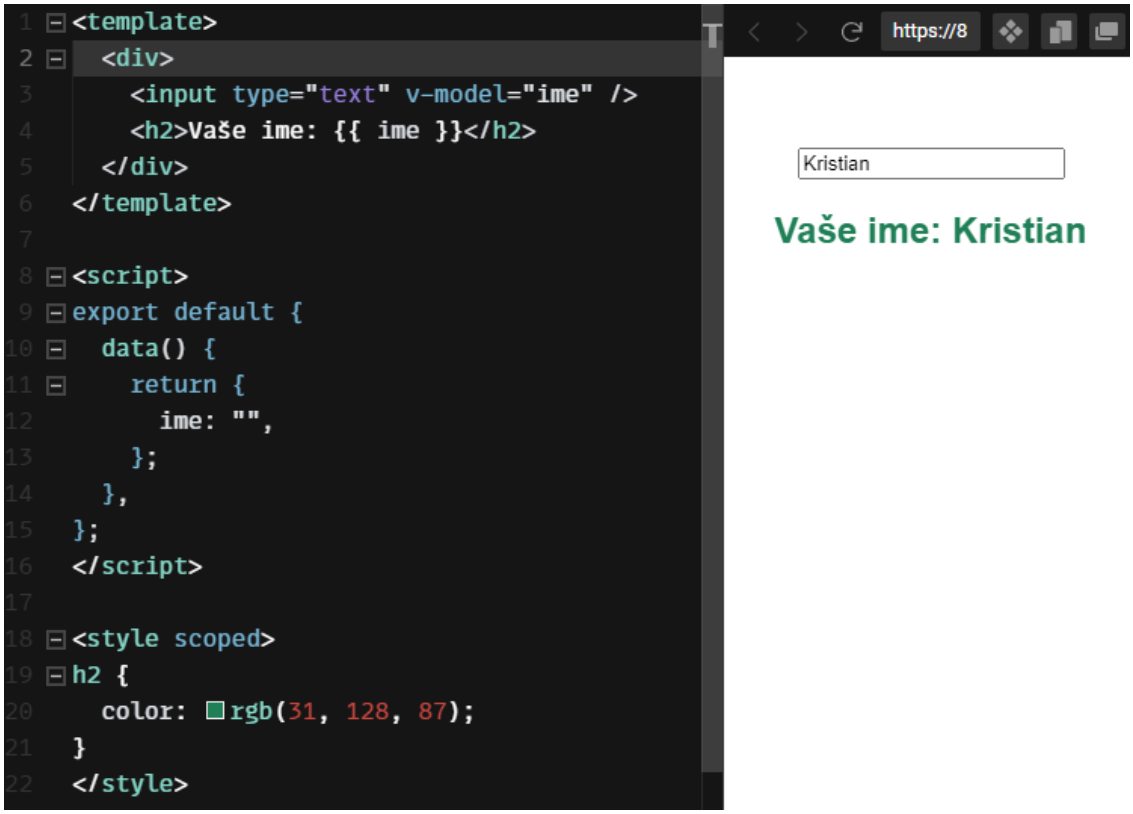
ipak web aplikacija je posrednik između krajnjeg korisnika i proizvoda kojeg on želi kupiti pa je bitno da taj proces bude što prikladniji. Takve aplikacije trebaju omogućiti korisnicima neke akcije kao što su unošenje podataka, sortiranje artikala i spremanje u košaricu, brisanje i mnoge druge stvari, na način da vodi računa o svim tim podacima i ažurira ih. U svijetu programiranja ti temeljni podaci nazivaju se stanjem aplikacije (eng. state).

Kada sagledamo neke od gore navedenih akcija koje korisnik obavlja putem pretraživača (eng. browsera), dakle pretraživanje nad podacima, dodavanje podataka u stanje, te mijenjanje i brisanje, sam razvoj aplikacije se počinje kompliciti. Često nastaju problemi, a jedan od bitnijih je taj da svaki put kada se promijeni stanje aplikacije, mora se ažurirati i korisničko sučelje. Također, kreiranje HTML elemenata i prikazivanje na preglednik zahtjeva ponavljanje istog kôda pa pogledajmo na jednostavnom primjeru (slika br. 2 i 3.) razliku između čistog JavaScripta i Vue okvira:



Slika 2 Običan (Vanilla) JavaScript

Gornji primjer (Slika br. 2) prikazuje kako se najčešće za korištenje običnog JavaScripta u aplikaciji dijeli HTML, CSS i JS kôd u zasebne datoteke te je potrebna dodatna logika za dohvaćanje i kreiranje elemenata te slušanje događaja nad njima.



Slika 3 Primjer Vue komponente

Slika br. 3 prikazuje kako Vue okvir ima raspodjelu HTML, CSS i JavaScripta unutar jedne Vue datoteke te znatno pojednostavljuje slušanje događaja nad elementima te ispis promjena na korisničko sučelje u samo nekoliko jednostavnih linija kôda.

3 ZNAČAJKE I FUNKCIONALNOSTI OKVIRA

Svaki frontend okvir ima svoj pristup kako će se web aplikacija razvijati, upravljati događajima pretraživača i ažurirati DOM te ne postoji jedinstveni način kako će se ona izraditi. Različiti projekti zahtijevaju različita rješenja i svaki okvir ima svoje prednosti i mane pa je stoga ključan odabir pravoga i primjerenoga rješenja. Neka pitanja koja valja uzeti u obzir su koje web pretraživače okvir podržava, kakva mu je zajednica i koliko je opširna dokumentacija, može li se uklopiti u aplikacije koje koriste druge okvire, koliko teško ga je naučiti i koje programske jezike na domeni (domain specific languages - DSL) upotrebljava? U kontekstu okvira, DSL objedinjuje varijacije JavaScript i HTML kôda koja omogućava veću razinu apstrakcije, deklarativniji pristup i generalno manje kôda. Dakle to je posebna sintaksa koju pretraživači ne mogu direktno čitati već se prvo mora transformirati u JavaScript ili HTML te na taj način okviri u pozadini upravljaju DOM interakcijama umjesto programera [1].

3.1 Krivulja učenja i stabilnost

Frontend područje jedno je od brzo rastućih grana IT svijeta, a kao i ostale zahtjeva konstantno učenje. Okviri evoluiraju i napreduju na način da ustanovljene mane otklanjaju te zamjenjuju novim značajkama jer ipak je to način da ostanu relevantni i pouzdani. Ne bi bilo poželjno da se programeri moraju brinuti hoće li stići naučiti koristiti okvir prije nego li dođe nova verzija koja će dodatno zakomplicirati sam proces učenja. Stoga je prije donošenja odluke o tome koji okvir se želi koristiti u projektu, vrlo važno uzeti u obzir i vrijeme koje je potrebno da se njegove značajke nauče koristiti.

Svaki okvir koristi svoju sintaksu (DSL) kako bi pružili deklarativno pisanje kôda te ujedno smanjili njegovu količinu, što ujedno označava dodatno učenje novih stvari koje za neke okvire povećavaju krivulju učenja. Primjerice, React se zasniva na JSX sintaksi koja je kombinacija HTML i JavaScripta. React programer nije nužan koristiti JSX no potencijalno se smanjuju mogućnosti iskorištavanja određenih značajki okvira tako da se uglavnom preporučuje njezino korištenje. Angular je poznat po svojoj kompleksnosti, a jedan od razloga je što se temelji na TypeScript programskom jeziku koji je superset JavaScripta. Generalno gledajući on ima najstrmiju krivulju učenja i uglavnom se koristi za veće aplikacije. S druge strane, Vue koji je relativno noviji od prethodno spomenutih okvira, pojednostavio je mnoge stvari i zapravo osim

nekih ključnih riječi nema poseban DSL već je fokus na običan HTML i JavaScript, što je jedan od razloga zašto ga većina smatra vrlo jednostavnim izborom.

Promatrajući trenutno najpopularnije okvire, React, Angular i Vue, sva tri su stabilna i pod stalnom kontrolom svojih timova, evoluiraju korak po korak te se nove značajke ne dodavaju tako često, tako da se programeri mogu prilagoditi sitnim inkrementalnim promjenama. Nova verzija Angulara se uglavnom objavljuje svakih šest mjeseci, no postoji razdoblje od još pola godine prije nego li se uvedu promjene, što ukupno znači da programeri koji se odluče koristiti Angular imaju jednu godinu napraviti potrebne promjene i prilagoditi se novoj verziji. React tim nema točno određeni raspored već kao i ostali nastoje minimizirati broj značajnijih promjena, no tipično se one događaju svake godine. Nova verzija Vue 3 objavljena je krajem 2020. godine u kojoj je objavljena nova značajka Composition API, inspirirana Reactovom Hook značajkom.

3.2 Upotreba i performansa

Još jedan bitan aspekt kod razvoja web aplikacija pomoću okvira je njezina veličina koja je vezana uz brzinu učitavanja na različite uređaje. Performansa okvira obuhvaća koliko brzo je potrebno da se sadržaj aplikacije prikaže na korisnikovom zaslonu i postane upotrebljiva i spremna za interakciju. Kako bi aplikacija postala funkcionalna potrebno je preuzeti kôd samog okvira te kôd aplikacije, stoga neki okviri pružaju bolje performanse od drugih ukoliko je riječ o manjim projektima. Manja veličina paketa označava brže preuzimanje na korisnikov pretraživač pa usporedivši Vue koji trenutno teži oko 33kB, Angular oko 65kB te React oko 43kB, možemo uočiti kako je Vue vrlo optimalna solucija za manje projekte, no ipak to ne znači da se njegova performansa smanjuje povećanjem zahtjeva aplikacije [12].

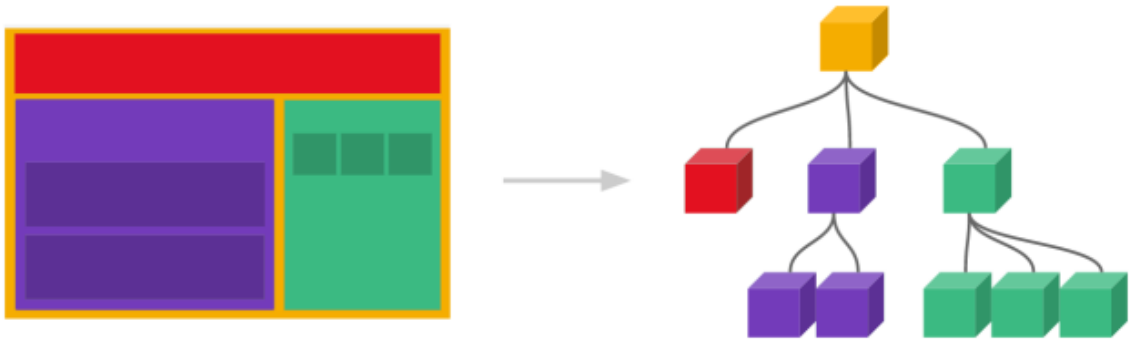
3.3 Preusmjeravanje (eng. Routing)

Jedna od glavnih značajki koju okviri koriste za izgradnju SPA je mehanizam preusmjeravanja koji korisniku omogućuje navigaciju kroz web aplikaciju. Postoje dvije vrste preusmjeravanja, preusmjeravanje na klijentskoj strani i na strani servera. U samome početku razvoja web aplikacija, za preusmjeravanje sadržaja bi web stranice ponovno slale zahtjev na server za dohvaćanje novih podataka te je to bio jedini način na koji su se podaci naizmjenično prikazivali u pregledniku. Tada se to činilo kao brzo i efikasno rješenje, no problem kod ovog načina je što se za svaki novi zahtjev, tj. svakom novom navigacijom rute taj proces mora ponoviti, što rezultira kašnjenje

učitavanja stranice nekoliko stotinki pa čak i sekundi. Stoga, novije web aplikacije baziraju se na paradigmi preusmjeravanja na strani klijenta za što su zaduženi frontend okviri koji na svoje načine ukomponiraju HTML poslan sa servera i pretvaraju ga u dinamičnu DOM strukturu. Kada se aplikacija prvo učita u preglednik, ruta i sadržaj koji će se prikazati ovise o URL-u. Navigacijom kroz aplikaciju tj. pritiskanje određenih linkova promijeniti će URL adresu te će se učitati ruta sa novim sadržajem, no neće se dogoditi ponovno učitavanje stranice. Isto tako, ukoliko se pritisne botun za povratak na prethodnu stranicu, učitati će prethodno posjećena ruta i njen prikaz.

3.4 Komponente - kompozicija

Ono što većina frontend okvira dijeli je ideja izgradnje web aplikacije korištenjem komponenti gdje se individualni dijelovi poput zaglavlja, podnožja, bočne navigacijske trake i sl. hijerarhijski slažu i spajaju te tako skupa predstavljaju jednu cjelinu (slika br. 4). Dakle, komponente su manji blokovi korisničkog sučelja koji enkapsuliraju HTML (sadržaj/struktura), CSS (stil) te JavaScript (i/ili DSL) logiku potrebnu za renderiranje. Svaki okvir ima svoj način na koji se one implementiraju i slažu, bilo višerazinski gdje imamo komponente unutar komponenti ili pak jedna uz drugu. Ovakav modularan dizajn s razlogom je korišten u većini okvira jer programeri mogu jasnije i brže kreirati korisnička sučelja, integrirati ih, izmjenjivati te otklanjati greške ukoliko je to potrebno. Također, unutar jedne aplikacije one se mogu koristiti i referirati više puta, a sav kôd koji se odnosi na jednu komponentu se uglavnom nalazi u jednoj datoteci pa tako programer točno zna gdje treba napraviti izmjene. No problem do kojeg dolazimo je prosljeđivanje podataka (eng. props) ili slušanje događaja iz drugih komponenti koje uzrokuju određene akcije unutar aplikacije. Ukoliko se radi o većim aplikacijama slanje podataka kroz slojeve komponenti se može zakomplicirati pa kako bi se zaobišao taj problem, okviri pružaju funkcionalnost koja se zove ubrizgavanje ovisnosti (eng. dependency injection). Ideja je da se određeni podaci ne prosljeđuju nepotrebno kroz sve komponente sve dok se ne dođe do željene komponente, već se odmah direktno pošalju odgovarajućoj komponenti. Svaki okvir tu implementaciju koristi drukčijim imenom i na drukčiji način, no svrha je uglavnom ista. Vue u tom slučaju koristi provide() i inject() metode, Angular naziva to ubrizgavanjem ovisnosti, a React koristi Context API [1].



Slika 4 Ideja kompozicije [4]

3.5 Spajanje podataka (eng. data binding) i događaja (eng. event binding)

Još jedna bitna stavka koju okviri na svoje posebne načine primjenjuju je spajanje podataka (eng. data binding) između komponenti i DOM-a, što označava sinkronizaciju između prikaza i modela aplikacije. Ova ideja se još naziva sistem reaktivnosti jer se zasniva na spajanju stanja aplikacije s aktivnim elementima sučelja koje korisnik mijenja. Postoje dva tipa spajanja podataka: jednosmjerno i dvosmjerno spajanje. Jednosmjerno kao što samo ime govori, podrazumijeva da se bilo koja vrijednost varijable iz komponente preslikava na UI ili obrnuto, dakle u samo jednome smjeru. S druge strane, dvosmjerno označava da se sinkronizacija događa u oba smjera istovremeno. To je primjerice unos teksta u element forme (osluškuje se događaj iz DOM-a) i ispisivanje tog unosa u isto vrijeme na neki element u DOM-u.

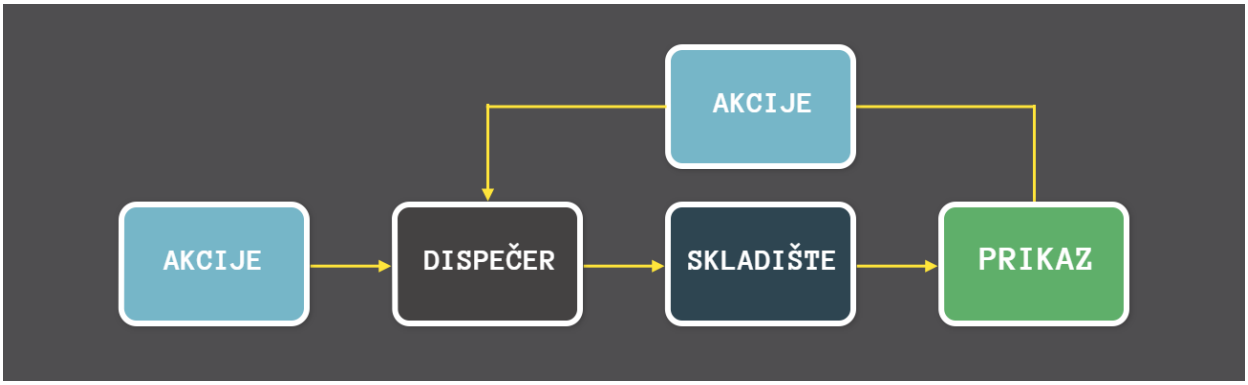
Vue i Angular omogućuju oba načina spajanja na vrlo sličan način (v-model i ng-model), dok React primarno nije dizajniran za dvosmjeran rad te uglavnom teži jednosmjernom (iz komponenti ka sučelju), iako se može dodatno implementirati logika za dvosmjerni tok.

3.6 Upravljanje stanjem (eng. state management)

Kao što se već moglo utvrditi, moderne web aplikacije često upravljaju s velikim brojem podataka te je zbog toga potrebno imati robustan mehanizam upravljanja stanjem. Ako uzmemo primjer aplikacije online trgovine gdje se odvija mnogo interakcija, primjerice ukoliko korisnik odabere neki proizvod i stavi ga u košaricu, tada je potrebno te podatke zapisati i pohraniti negdje u našoj

aplikaciji te ih prikazati na sučelje. Isto tako ukoliko korisnik zatraži detaljan ispis nekog proizvoda, vrlo vjerojatno ćemo morati dohvatiti te podatke negdje iz baze kako bi ih aplikacija dalje reprezentirala. Dakle, svaki put kada korisnik obavlja neku interakciju sa aplikacijom mijenja se njezino stanje, a svaki okvir naravno ima svoj način upravljanja stanjem, točnije koriste se različite knjižnice. Vue tim razvio je svoju knjižnicu Vuex, a uz React okvir često se koristi Redux, no iako se one u principu razlikuju, obje su inspirirane i temeljene na aplikacijskoj arhitekturi Flux razvijene od Facebook tima. Za razliku od standardne MVC (Model-View-Controller) arhitekture, koja je razvojem kompleksnijih aplikacija počela predstavljati otežano rješavanje problema zbog dvosmjernog putovanja podataka, Facebook je razvio ideju da podaci kroz aplikaciju teku u jednome smjeru. Pojava Flux arhitekture znatno je pridonijela općem razvoju okvira i knjižnica za upravljanje stanjem web aplikacija [9].

Glavna ideja je podijeliti aplikaciju na četiri dijela: **action, dispatch, store i view**. Akcije (eng. actions) su zapravo funkcije koje korisnik pri interakciji sa aplikacijom poziva kroz sloj prikaza da se ažuriraju i promjene podaci u DOM-u. Te akcije se šalju na tzv. dispečere (eng. dispatchers) koji su središnji čvorovi zaduženi za primanje akcija (i podataka) te prosljeđivanje istih određenim redoslijedom na prikladno skladište (eng. store) da se ažurira stanje aplikacije. Skladište predstavlja objekt koji je odgovoran za upravljanje stanjem aplikacije putem metoda, dakle njegova uloga je vrlo slična modelu u MVC arhitekturi. Iz njega se dohvaćaju ti podaci koji se zahtijevaju preko akcija kako bi se u konačnici oni prikazali na sloj prikaza. Također, specifično je da se podaci unutar stanja ne mogu direktno mijenjati već je moguće jedino zatražiti promjenu stanja prosljeđivanjem akcija na dispečere. Prikaz (eng. view) nije ništa drugo nego komponenta korisničkog sučelja koja je zadužena za prikazivanje podataka te upravljanje događajima korisnika. Sloj prikaza se u nekim okvirima može dodatno podijeliti na sloj prikaz-model (npr. script dio u Vue) koji je povezan na skladište i dispečere, te prezentacijski sloj koji nije povezan (npr. predlošci u Vue).



Slika 5 Pregled Flux arhitekture [9]

Dakle, možemo primijetiti kako se Flux arhitektura (slika br. 5) temelji na jednosmjernom toku podataka - preko sloja prikaza korisnik poziva akcije, a akcije se prosljeđuju dispečerima koji dalje prosljeđuju zahtjeve ili podatke na određeno skladište koje ažurira sloj prikaza (eng. view).

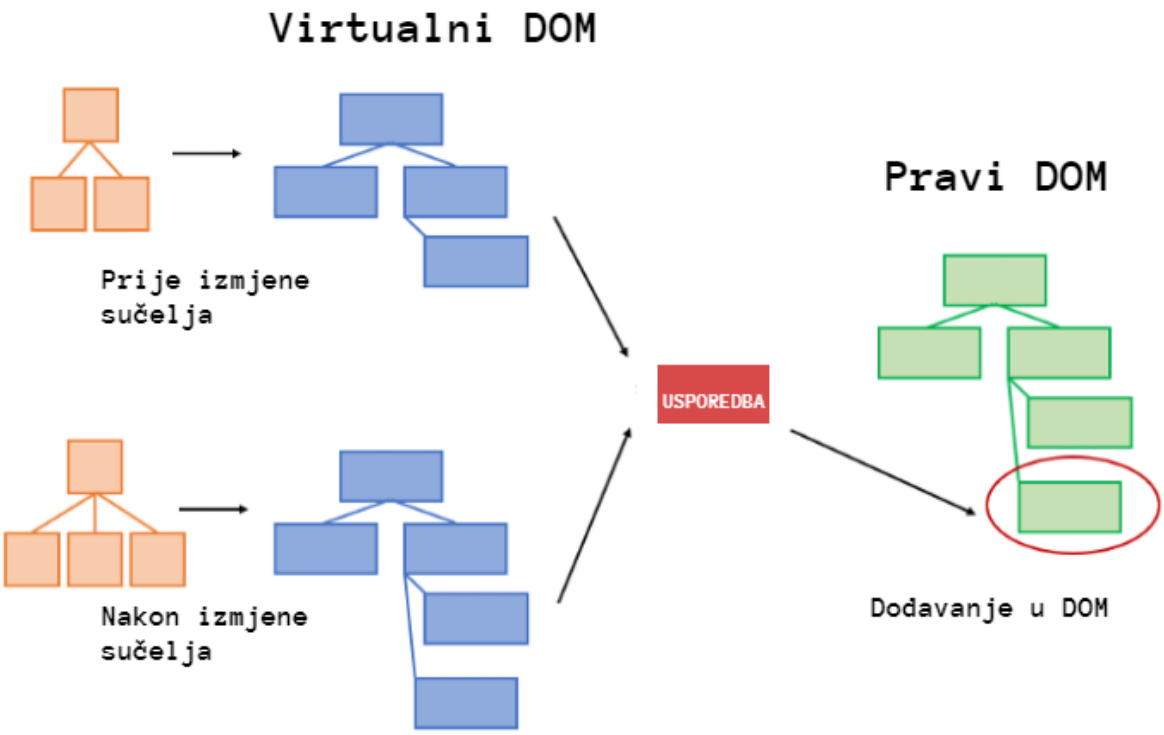
3.7 Virtualni i inkrementalni DOM

Prije nego što se opiše koncept Virtualnog DOM-a, treba se znati da je DOM stablasta struktura objekata koja predstavlja web stranicu. Web preglednici su odgovorni za obradu DOM implementacijskih detalja, kako bi mi kao programeri koristeći JavaScript i CSS mogli raditi promjene nad čvorovima tog modela, brisati ih te ubacivati nove. Običan DOM nikada nije bio optimiziran za kreiranje dinamičnog korisničkog sučelja jer su u prošlosti web stranice bile statičke. Naime, prikazivanje informacija i manipulacija nad DOM-om jako je skupo iz razloga što se prilikom mijenjanja jednog čvora cijela struktura mora promijeniti i sve ponovno prikazati na sučelje. Pojavom modernih web aplikacija koje se baziraju na mnogo interaktivnih aktivnosti javila se i potreba za optimalnijim rješenjem – Virtualnim DOM-om [1].

Virtualni DOM je koncept u programiranju koji označava da se virtualna reprezentacija informacija o preglednikovom DOM-u nalazi u memoriji te se sinkronizira sa pravim DOM-om (slika br. 6). Dakle, uvodi se novi sloj apstraktne verzije koja je zapravo kopija u obliku JavaScript objekata koji predstavljaju stablastu strukturu pravog modela. Ti virtualni čvorovi su vrlo lagani jer sadrže samo nužne podatke za razliku od onih u pravom modelu koji su mnogo detaljniji, stoga je ovakav proces mnogo brži. Kada god se promijeni stanje aplikacije dodavanjem, brisanjem, ili ažuriranjem elemenata, posao okvira je napraviti promjene nad virtualnim modelom, bez da se odmah direktno mijenja pravi DOM. Svaki okvir ima drukčiji pristup kako pratiti gdje su se

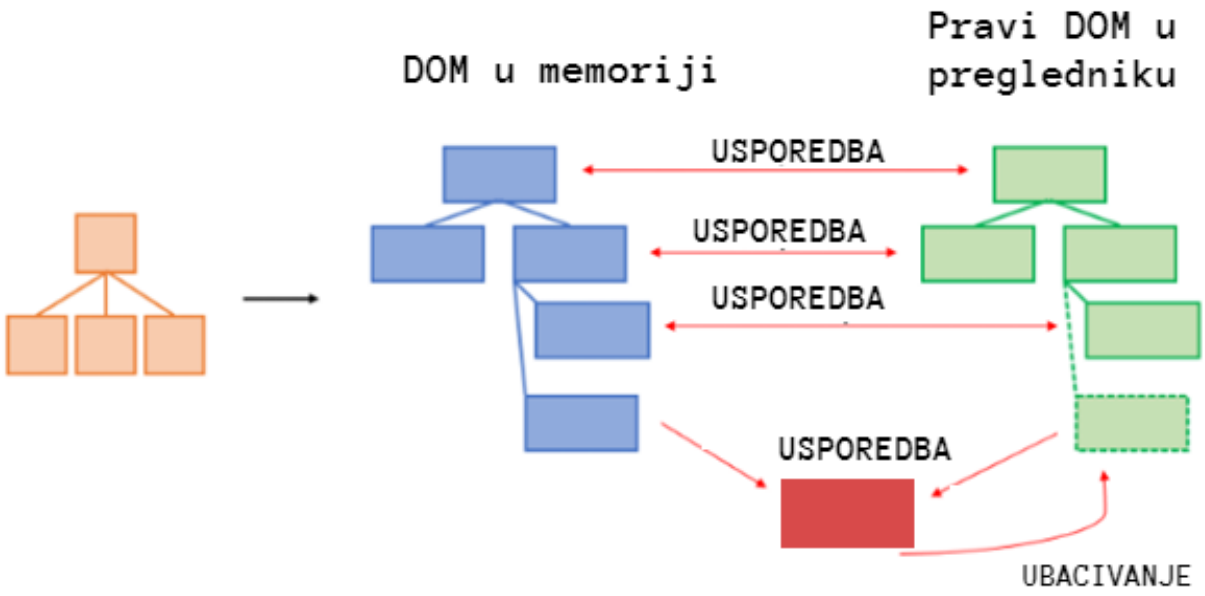
dogodile promjene, na koji način se odvija usporedba virtualnog i pravog modela (eng. diffing) te kako bi se komponente unutar aplikacije trebale izmijeniti.

Zbog toga što sve te odluke manipuliranja atributima, upravljanje događajima i sl. donose okviri umjesto nas, ne moramo se uopće brinuti o tome kako upravljati DOM-om, što dodatno dokazuje njihovu vrijednost.



Slika 6 Virtualni DOM

Za razliku od virtualnog, za implementaciju inkrementalnog DOM-a se ne gradi virtualno stablo već se za lociranje promijenjenih čvorova odmah koristi pravi model (slika br. 7). Ovaj pristup inzistira na smanjenju memorijskog alociranja i straničenja (eng. paging), inkrementalno ažurirajući čvor po čvor što ga čini pogodnim za uređaje manjeg memorijskog kapaciteta kao što su mobiteli [1].



Slika 7 Inkrementalni DOM

Naravno, svaki ima svoje prednosti i nedostatke sam za sebe te je teško odrediti koji pristup je bolji, stoga se pri odabiru mora napraviti kompromis između vremena i upotrebe memorije. U svakom slučaju, virtualni i inkrementalni DOM odlične su opcije koje okviri koriste za razvoj dinamičnih web aplikacija. Angular je za svoj izbor odabrao inkrementalni, dok Vue i React koriste virtualni DOM, iako ne primjenjuju potpuno istu logiku za prikazivanje i usporedbu promjena.

4 VUE I OSTALI JAVASCRIPT OKVIRI

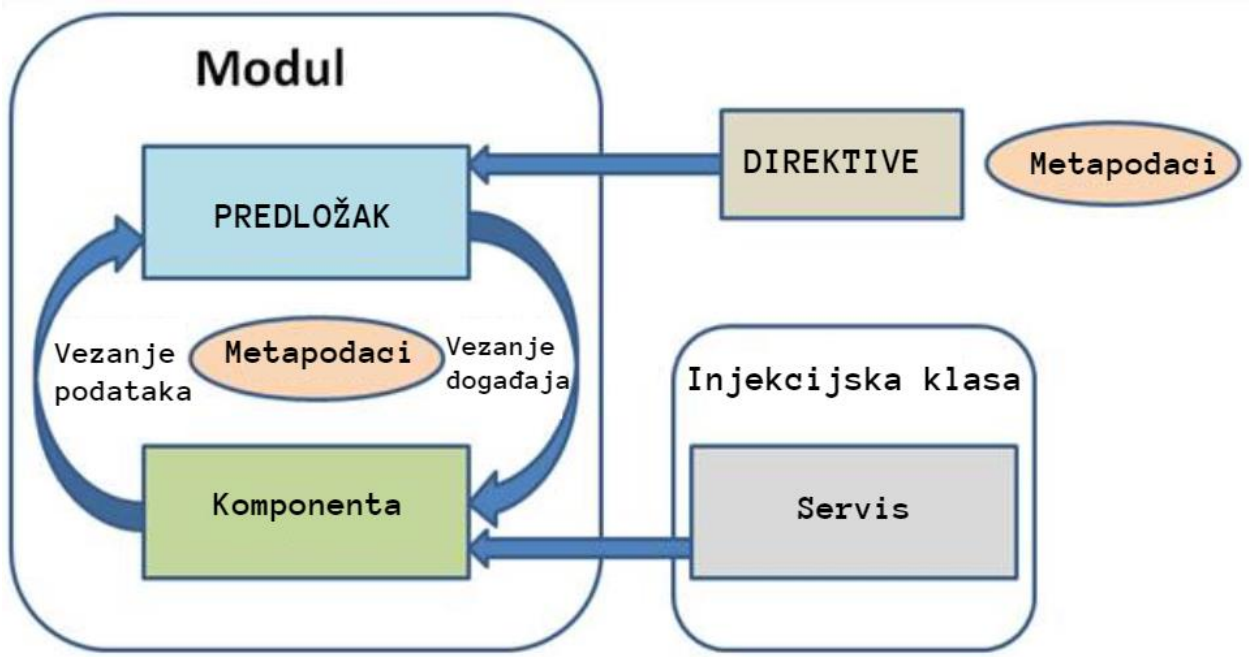
U ovom poglavlju će se ukratko napraviti usporedba trenutno tri najpopularnija frontend okvira Angular, React i Vue, te će se opisati na koji način funkcioniraju. Zatim će se sastaviti popis prednosti i nedostataka svakoga kako bi se bolje istaknulo za koje svrhe je određeni okvir prikladniji.

4.1 Angular

Kao najstariji član ove grupe, Googleov AngularJS (Angular 1) je od svog osnutka 2010. godine prošao kroz različite transformacije, a najznačajnija promjena dogodila se 2016. godine kada objavljenja potpuno nova verzija okvira Angular (Angular 2). To je uzrokovalo veliku frustraciju među programerima jer nije bilo nikakve migracije između nove verzije zbog čega su neki odlučili kompletno napustiti okvir. Jedna od značajki novije verzije Angulara je TypeScript sintaksa koja ima značajne prednosti, kao što su navigacija, lakše otklanjanje grešaka unutar aplikacije, statička provjera tipa podatka i sl.

Kada je riječ o razvijanju skalabilnih aplikacija, Angular se čini kao efikasan i pouzdan izbor zbog svoje modularne strukture te bogatog opusa funkcionalnosti. U sebi ima puno ugrađenih UI komponenti te značajki poput validacije formi, preusmjeravanja, upravljanja stanjem i HTTP zahtjevima i sl. pa ga mnogi stoga smatraju i platformom. Vrlo je rijedak slučaj da Angular nema već ugrađeno rješenje, no to ujedno označava i više dokumentacije kojom se programeri moraju koristiti. Njegov CLI (eng. command-line interface) alat jako je moćan i mnogo olakšava programerima postavljanje projekta navodeći ih kako strukturirati kôd, kako postaviti ruter za usmjeravanje, kako dodavati nove komponente i sl.

Zbog svojih određenih restrikcija i strožih pravila kako bi se tok aplikacije trebao razvijati, Angular do nekih mjera uskraćuje slobodu programera da postavi projekt na njegov način što se nekima možda i neće svidjeti.

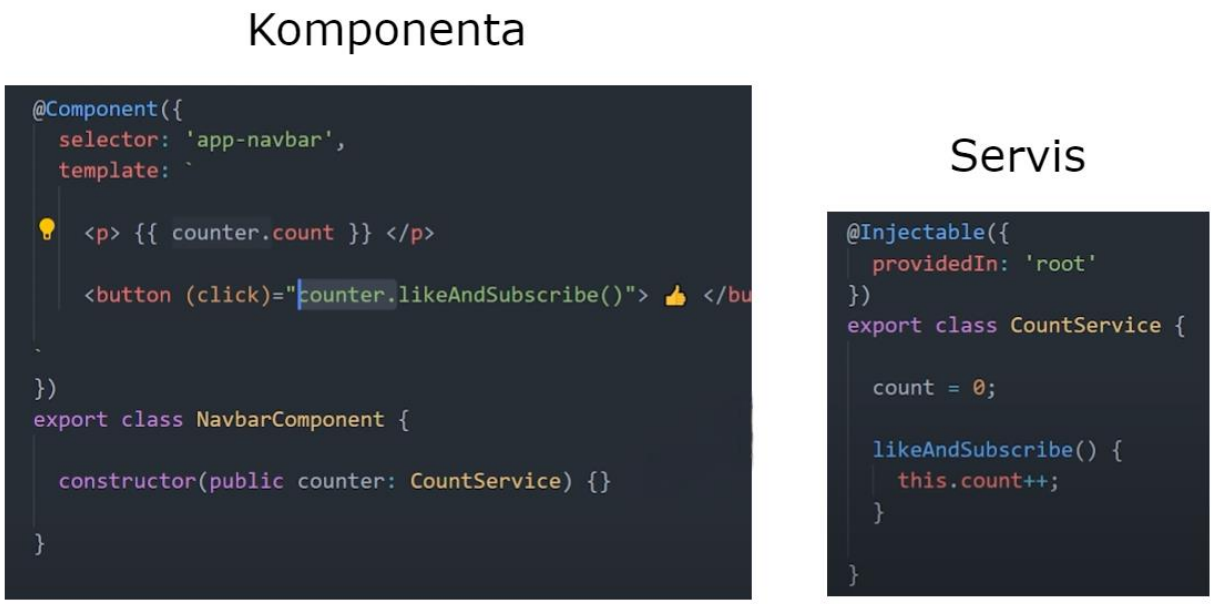


Slika 8 Angular arhitektura

Zbog svojih ranih početaka razvoja, prva verzija Angulara inspirirana je standardnom MVC (Model-View-Controller) arhitekturom koja se temelji na separaciji modela i prikaza, što programerima omogućuje paralelan rad na različitim dijelovima aplikacije (frontend i backend). Model, u kojemu su pohranjeni čisti podaci aplikacije, je u potpunosti odvojen od prikaza (eng. view) koji prezentira te podatke korisniku i unutar kojega se implementira kako će se odvijati interakcija s korisnikom. Ključan dio ove arhitekture je kontroler (eng. controller), on donosi odluke kao što je generiranje ili brisanje prikaza na sučelje te spaja prikaz i model na način da ažurira promjene iz modela na sučelje. No razvojem kompleksnijih web aplikacija, sve više okvira kao i novija verzija Angulara teži MVVM arhitekturi gdje umjesto kontrolera imamo prikaz-model (eng. view-model). To je zapravo apstrakcija prikaza (view) koja pojednostavljuje komunikaciju i povezivanje podataka s modelom (eng. data binding), te sadrži funkcije vezane za događaje i logiku za procesiranje podataka kao npr. validaciju formi [11].

Angular UI komponenta razdvojena je na dva dijela gdje se u jednom piše TypeScript logika, a za drugi dio se definiraju predlošci (eng. templates) unutar kojih se piše HTML skupa s okvirovim ključnim riječima. Tako imamo odvojenu TypeScript logiku koja komunicira s tim predloškom, dok u pozadini okvir manipulira i ažurira DOM (slika br. 8).

Kada se web aplikacija razvije do te mjere da sadrži ogromnu količinu komponenti, potrebno je ostvariti što jednostavniju komunikaciju i prosljeđivanje podataka pa zbog toga Angular koristi servise. Svaki servis zamišljen je kao injekcijska klasa (eng. injectable service class) koja enkapsulira bilo koju funkciju, vrijednost varijable ili značajku koja je potrebna aplikaciji [3]. Drugim riječima, glavna svrha servisa je postići organiziranu biznis logiku i efikasno omogućiti korištenje podataka, funkcija i sl. unutar različitih komponenti. Ideja je učiniti komponente što modularnijima i jednostavnijima te da im jedini posao bude prezentiranje podataka. Za svaku komponentu koja treba koristiti stanje ili logiku određenog servisa (npr. validacija formi ili ispis na konzolu) mora se proslijediti ime klase tog servisa u konstruktor komponente (slika br. 9).



Slika 9 Primjer injekcijske klase u Angularu

Prednosti:

- Velika zajednica i dobra reputacija – kvalitetna podrška od Googlea
- Brže i lakše otklanjanje grešaka – prednosti TypeScripta
- Potpuno opremljen velikim brojem značajki i bogatom dokumentacijom
- MVVM arhitektura koja omogućuje lakši i intuitivniji rad nad kompleksnim projektima
- Modularnost i princip injekcijskih klasa
- Dvosmjerno povezivanje podataka (eng. two-way data binding)
- Vrlo razvijeni alati za testiranje i poboljšane performanse serverskog renderiranja

- Prijevremeno (eng. ahead-of-time) kompiliranje HTML-a i TypeScripta što pruža brže renderiranje i poboljšava sigurnost jer se JavaScript kôd kreira pri procesu izgradnje, prije nego što ga preglednik preuzme

Nedostaci:

- Strma krivulja učenja zbog svoje kompleksnosti i dodatnog učenja TypeScripta
- Nije idealan izbor za vrlo jednostavne projekte zbog svoje veličine
- Nagla migracija i velika promjena sa stare verzije na novu
- Proces ažuriranja DOM-a je za nijansu sporiji od Vue i Reacta koji koriste virtualni DOM

4.2 React

React se smatra UI knjižnicom otvorenog koda (eng. open-source library) što podrazumijeva da je mnogo manje ugrađenih značajki koje okvir sadržava za razliku od Vue i Angulara. Isključivo se temelji na izgradnji komponenti koje kreiraju UI pa ukoliko se radi o većoj aplikaciji za koju je potrebno upravljanje stanjem, preusmjeravanje i sl., tada se moraju koristiti paketi njegove zajednice (eng. community packages). Facebook tim razvio je React 2013. godine, a od tada je znatno dobio na popularnosti te iza sebe prikupio veliki broj programera koji kontinuirano pridonose razvoju okvira i njegovog ekosustava [12].

Ideja se zasniva na kreiranju i svojevoljnom slaganju komponenti koje se mogu ponovno koristiti i međusobno ugnježđivati, no u konačnici imamo jednu glavnu (eng. root) komponentu koja „omata“ ostale i tako predstavlja cijelu aplikaciju. Svaka komponenta ima svoju unutarnju logiku i može, ali i ne mora imati svoje stanje, no ne postoji stroga podjela HTML-a i JavaScripta kao kod Angulara i Vue (template i script), već je sve kombinirano JSX sintaksom. Pomoću te sintakse se opisuje Reactu koje HTML elemente bi trebao prikazati u pravom DOM-u, implementira se upravljanje događajima i sl. Komponente su obične JavaScript funkcije (ili klase) koje kao argumente prihvaćaju tzv. svojstva (eng. props - properties) koja se mogu referirati u tijelu te funkcije ili u dijelu koji ta funkcija vraća. U ovome kontekstu, ono što su atributi za HTML elemente, to su svojstva za React komponente. Funkcije vraćaju React elemente napisane u JSX sintaksi i svaki put kada se promijeni stanje u komponenti, dobijemo novi React element nad kojim se poziva ReactDOM.render() metoda koja ažurira UI. Bitno je naglasiti da React koristi virtualan DOM pomoću kojega se ažurira samo tamo gdje je to potrebno. Ako želimo svakoj komponenti

dati svoje lokalno stanje koristi se tzv. React *hook* funkcija *useState*() čije pozivanje vraća trenutno stanje i funkciju pomoću koje se može ažurirati to stanje (slika br. 10).

```
import React, { useState } from 'react';

function Example() {

  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Slika 10 React komponenta s Hook implementacijom

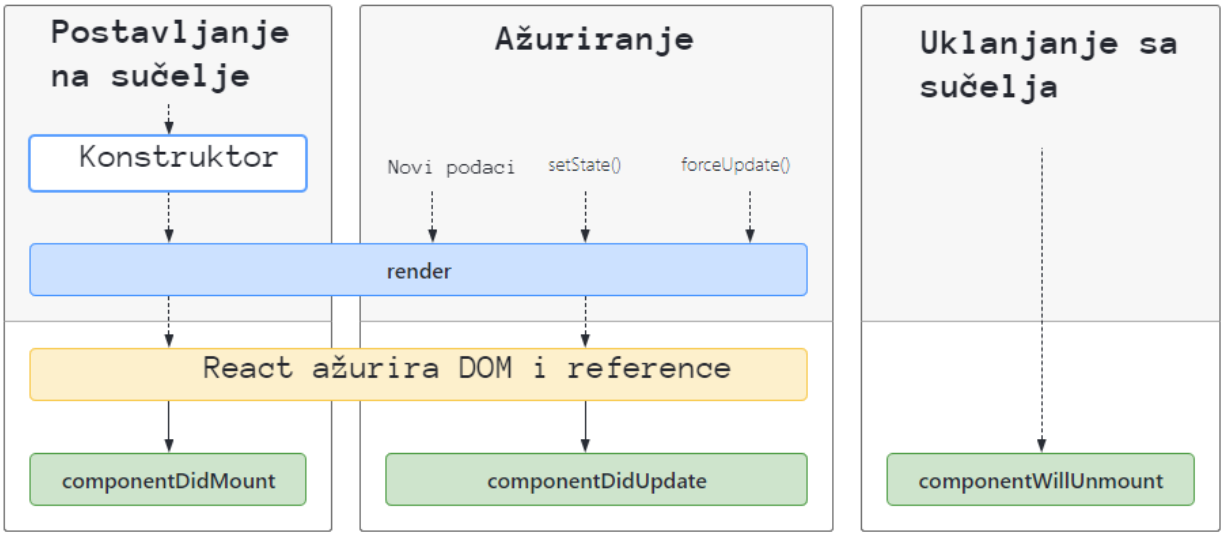
Na gornjem primjeru varijabla *count* predstavlja reaktivno stanje, a jedini način da ga se promijeni je pomoću funkcije *setCount,* dakle ne može ga se direktno modificirati. I tako svaki put kada se promjeni stanje, React zna da je došlo do promjene koje je potrebno renderirati na sučelje.

Ono što svaka komponenta ima je svoj životni ciklus (eng. lifecycle) tijekom kojega se mogu pozivati ugrađene metode u različitim fazama života komponenti, koje programerima pružaju veću kontrolu pri upravljanju ponašanja komponenti. Razumijevanje životnog ciklusa komponenti bitno je za efikasno iskorištavanje značajki koje React nudi. Primjerice, može se iskoristiti znanje ako određena komponenta nije spremna tj. čeka učitavanje podataka, tada se za to vrijeme može korisniku prikazati animacija učitavanja, sve dok se određeni podatak ne dohvati.

React ima tri faze koje treba uzeti u obzir (slika br. 11) [10]:

- Mounting - renderiranje na DOM po prvi put, pozivaju se metode za prikazivanje - *render*() i kada se komponenta postavila na sučelje - *componentDidMount*()

- Updating - ažuriranje stanja i podataka unutar već postojeće komponente u DOM-u (ponovno renderiranje komponente), osim *render*() se najčešće koristi i metoda kada se komponenta ažurirala - *componentDidUpdate*()
- Unmounting – posljednja faza komponente, brisanje elementa iz DOM-a, a prikladna metoda je *componentWillUnmount*()



Slika 11 React životni ciklus [2]

Prednosti:

- Jako velika zajednica koja kontinuirano pridonosi razvoju okvira
- Održavan od Facebooka, a korišten i u popularnim aplikacijama kao što su Netflix, Airbnb, WhatsApp, Instagram itd.
- Brzo renderiranje i virtualni DOM
- Manja kompleksnost učenja za razliku od Angulara i laka migracija na nove verzije
- Jednosmjerno povezivanje podataka – lakše otklanjanje grešaka i stabilan kôd
- React Native – znanje o Reactu može se primijeniti na okvir koji omogućuje web programerima kreiranje mobilnih aplikacija na sličan način kao React web aplikacije
- Kvalitetan CLI (eng. command-line interface) alat - *create-react-app* koji pomaže programerima pri kreiranju i postavljanju projekata

Nedostaci:

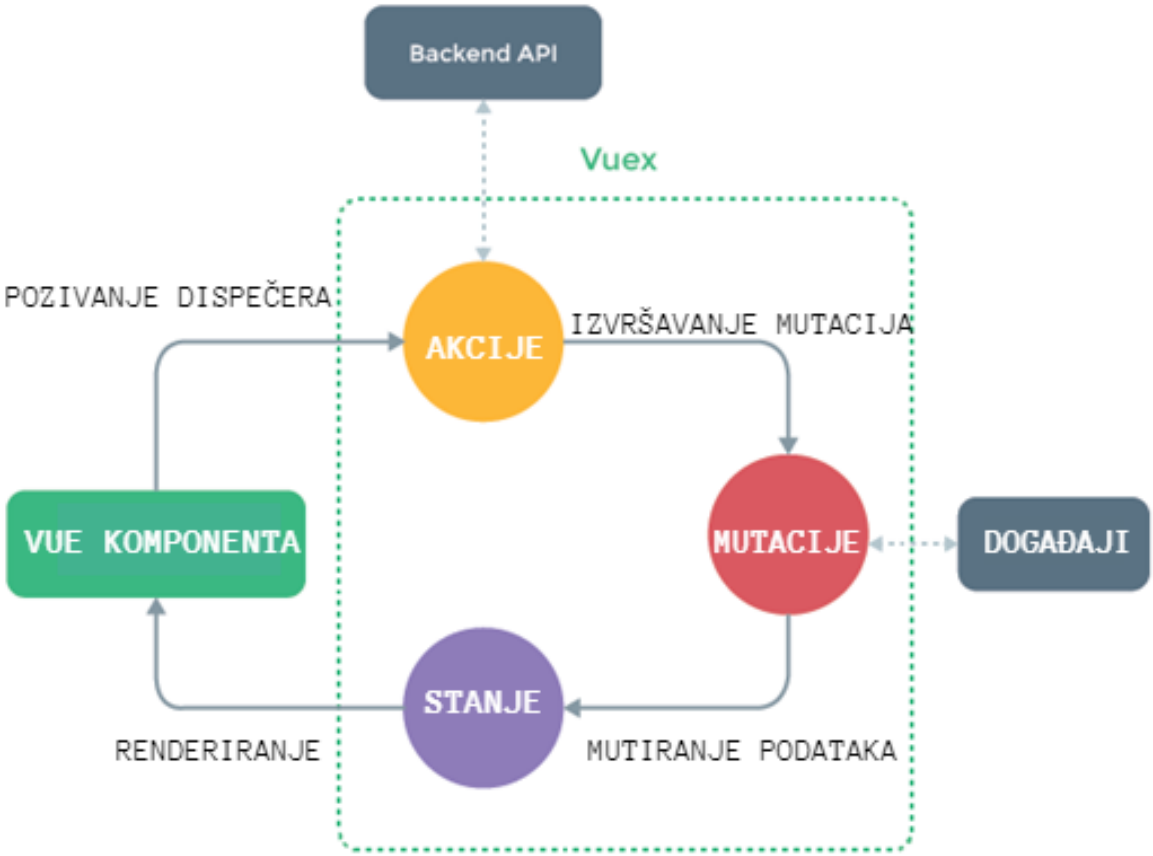
- React nema stroge konvencije i pravila (eng. un-opinionated), već njih uglavnom donose programeri iz React zajednice što ne znači da će principi dizajna aplikacije biti uvijek konzistentni

4.3 Vue

Vue je relativno mladi okvir, no svoje kasnije pojavljivanje u frontend svijet iskoristio je na efikasan način tako što je uzeo sve najbolje od već postojećih i najpopularnijih okvira te nadodao ono što im fali. Njegova fleksibilnost, lakoća učenja te jednostavnost samo su neke od ključnih karakteristika zašto se unazad nekoliko godina popeo na ljestvicu među najpopularnije okvire. Vue je modularan i progresivan, što označava da se razvoj web aplikacije započne sa nečim manjim pa postepeno nadograđuje u veći projekt ukoliko je to potrebno. Primarni fokus je na izgradnji sloja prikaza, što ga čini vrlo adaptivnim za integriranje u već postojeće projekte ili knjižnice i ne moramo ga koristiti da kontroliramo sve već samo neke dijelove stranice. Isto tako valja napomenuti da Vue također podržava TypeScript okruženje, a kako i Angular podržava arhitekturu izgradnje sučelja koja se temelji na komponentama, postoji mogućnost uzajamnog korištenja oba okvira.

Za razliku od Reacta ili Angulara, iza kojih od samog početka stoji velika organizacija, Vue je svoj uspon započeo od pojedinca, točnije bivšeg zaposlenika Googlea pod imenom Evan You. Prije nego što je Vue ugledao svoje svjetlo dana 2014. godine, Evan je bio član tima koji je radio projekte u okruženju AngularJS (Angular 1.0), od kojega je Vue i naslijedio dosta principa što je vidljivo primjerice u sintaksi (npr. v-if vs ng-if). Kao i Angular, Vue ima malo strožu odvojenost HTML kôda i JavaScripta, definiraju se tzv. predlošci unutar kojih se u HTML elemente ubacuju značajke okvira npr. da se ponavlja element (v-for) ili slušanje na klik događaj (v-on:click). Tako isto kao i kod Angulara imamo odvojeni JavaScript kôd koji je povezan s tim predloškom preko kojega se u konačnici podaci prikazuju na sučelje korisnika. Dakle postoji jasna separacija na 3 dijela: dio logike (script), predloška (template) i stila.

Za razliku od Angulara koji ima monolitsku strukturu, Vue je stavio naglasak na trivijalnosti programiranja i smanjenju ograničenja kako bi se aplikacija trebala strukturirati.



Slika 12 Vue arhitektura [4]

Na gornjoj slici (slika br. 12) predstavljena je MVVM (Model-View-View Model) struktura na kojoj se Vue temelji, a vidljive su sličnosti Flux arhitekture kao i kod Reacta. Dok je MVC (Model-View-Controller) dizajniran tako da su model i prikaz u potpunosti odvojeni, ideja MVVM-a je dopustiti sloju prikaza i modela direktnu komunikaciju, a to je upravo ono što Vue kroz prikaz-model (eng. View-Model) sloj postiže dvosmjernim povezivanjem podataka. Vue koristi posebne oznake – direktive, inspirirane od Angulara, koje okviru govore što da učini nad DOM elementima na način da web aplikacija bude interaktivna i sinkronizirana s podacima aplikacije [4].

Vue tim razvio je svoju knjižnicu Vuex za centralizirano upravljanje stanjem te se najčešće koristi pri porastu kompleksnosti aplikacije s čime raste i broj komponenti koje koriste to zajedničko stanje. Naime, ova knjižnica sastoji se od jednostavnih pravila koje pridonose održavanju i organizaciji strukture stanja, a stvorena je prema uzoru na Flux, Redux i Elm arhitekturu [8]. Ideja centraliziranog upravljanja kod Vuexa je da svaka komponenta aplikacije može pristupiti stanju, čime se rješava problem prosljeđivanja podataka između susjednih komponenti ili duboko

ugniježđenih komponenti (eng. prop drilling). Vuex stanje je reaktivno pa ukoliko neka komponenta prima neke podatke iz stanja, svaka promjena tog podatka će se odraziti i u komponenti. Također, slično kao kod Reacta (reduceri), stanje se ne može direktno mijenjati već se mora izvršiti mutacija (eng. commit mutation). Ova konvencija osigurava lakše otkrivanje gdje se dogodila promjena te korištenje alata za lakše razumijevanje razvoja aplikacije (npr. ispis svake mutacije).

Prednosti:

- Jednostavna sintaksa temeljena na čistom HTML-u i JavaScript-u te je zbog toga relativno lagan za naučiti
- Fleksibilan - podržava JSX sintaksu i TypeScript
- Prilagodljiv - sličnosti sa Angularom i Reactom u dizajnu i arhitekturi
- Mala veličina okvira te brzo renderiranje zbog virtualnog DOM-a
- Skalabilan– može se koristiti i za jednostavne aplikacije na jednoj stranici (eng. SPA), ali i za kompleksnije projekte
- Laka migracija na nove verzije
- Dvosmjerno povezivanje podataka (eng. two-way data binding)
- Kvalitetan CLI alat koji pomaže programerima pri kreiranju i postavljanju projekata

Nedostaci:

- Malo manja zajednica u odnosu na React i Angular (iako sve više raste) pa razvoj alata trećih strana nije toliko brz kao kod konkurenata
- Manji broj stručnih Vue programera i generalno manje otvorenih pozicija za zapošljavanje

5 WEB APLIKACIJA - PROJEKT

Kao praktični dio završnog rada izrađena je web aplikacija za kreiranje citata kroz koju su se demonstrirale glavne funkcije Vue okvira. Korisnik ove aplikacije može upisivati i pregledavati citate, izmjenjivati ih te brisati, a za svaku akciju je kreirana ruta tako da se može navigirati kroz različite dijelove aplikacije. Time je postignuto dinamičko mijenjanje sadržaja što je jedna od značajki SPA, a za upravljanje stanjem aplikacije je korištena knjižnica Vuex. Za lakše postavljanje projekta i instaliranje drugih paketa korišten je alat Vue CLI kojim se na vrlo jednostavan način ostvarila konfiguracija projekta i stvorilo okruženje spremno za programiranje.

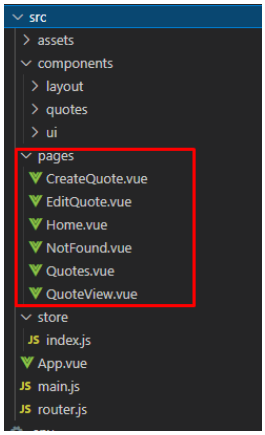
5.1 Struktura i komponente aplikacije

Stablasta struktura aplikacije započinje glavnim (root) čvorom App.vue (slika br. 13) koja čini glavnu komponentu unutar koje se ugnježđuju tri komponente: zaglavlje (TheHeader.vue), funkcionalna komponenta za navigaciju (eng. router-view) te podnožje (TheFooter.vue).

Slika 13 Glavna Root komponenta

Sučelje aplikacije podijeljeno je na šest sekcija koje komponenta za navigaciju izmjenjuje u ovisnosti o ruti u URL-u (slika br 14.):

- Home
- Quotes
- Create Quote
- View Quote
- Edit Quote
- Not Found



Slika 14 Glavne komponente za prikaz

5.2 Preusmjeravanje

Za navigaciju aplikacije korištena je metoda za kreiranje rutera (eng. createRouter) iz knjižnice vue-router unutar koje se definiraju rute i prikladne komponente za svaku od navedenih. Svaka ruta ima dio koji predstavlja putanju (eng. path) u URL adresi te dio za komponentu unutar kojega se definira koja će se komponenta prikazati za tu putanju (slika br. 15). Osim toga, Vue ruter nudi različite funkcije kao što je pamćenje prethodne rute (eng. createWebHistory) tako da se korisnik može vraćati unutar preglednika. Objekt ruter u konačnici se mora uključiti u Vue aplikaciju kako bi okvir u pozadini pratio na kojoj URL adresi se korisnik nalazi i reagirao na te promjene.

router.js

```
import { createRouter, createWebHistory } from "vue-router";
import Home from "./pages/Home.vue"
import Quotes from "./pages/Quotes.vue"
import CreateQuote from "./pages/CreateQuote.vue"
import QuoteView from "./pages/QuoteView.vue"
import EditQuote from "./pages/EditQuote.vue"
import NotFound from "./pages/NotFound.vue"

const router = createRouter({
    history: createWebHistory(),
    routes: [
        { path: "/", redirect: "/home" },
        { path: "/home", component: Home, name: "Home", meta: { title: "Home" } },
        { path: "/quotes", component: Quotes, name: "Quotes", meta: { title: "Quotes" } },
        { path: "/quotes/:quoteId", component: QuoteView, name: "QuoteView", meta: { title: "Quote View" }, props: true },
        { path: "/edit-quote/:quoteId", component: EditQuote, name: "EditQuote", meta: { title: "Quote Edit" }, props: true },
        { path: "/create-quote", component: CreateQuote, name: "CreateQuote", meta: { title: "Create Quote" } },
        { path: "/:notFound(.*)", component: NotFound, meta: { title: "Not Found" } },
    ]
});

router.beforeEach((to, _, next) => {
    document.title = `${to.meta.title} | Quotex`;
    next();
})

export default router
```

Slika 15 Primjer definiranih ruta

5.3 Povezivanje podataka unutar komponente

Unutar Vue komponente vidljiva je podjela HTML, JavaScript i CSS dijela, a ono što Vue okvir čini vrlo jednostavnim i intuitivnim su direktive s prefiksom "v-" kojima se vežu podaci iz stanja na elemente sučelja (slika br. 16). Kao što se može vidjeti, moguće je vezati podatke na atribute i klase elemenata, događaje i sl., a za ispis podataka u obliku teksta se koristi interpolacija sa duplim vitičastim zagradama. Kako je ideja razdvojiti JavaScript logiku te pojednostaviti template dio i učiniti ga što jednostavnijim za održavanje, koriste se svojstva za računanje (eng. computed). Ta svojstva pogodna su za dohvaćanje dinamičnih podataka te se pri svakoj promjeni tog podatka uz koju se to svojstvo veže, automatski sinkronizira u template dijelu. Vrlo efikasna stvar je što Vue koristi virtualni DOM pa se određena svojstva ne pozivaju više puta ukoliko se vrijednosti tog jednog podatka nisu promijenile (princip cachinga).

```
<template>
  <div v-if="quoteData">
    <div class="quote-wrapper" v-bind:style="{  backgroundImage: 'url(' + photoUrl + ')', 'background-size': 'cover'}" >
      <blockquote class="quote-blockquote">
        {{ quoteText }}
        <span>~ {{ quoteAuthor }} </span>
      </blockquote>
    </div>
    <div>
      <p class="attribution">Photo by {{ photoAuthor }} on Unsplash</p>
    </div>
  </div>
</template>

<script>
export default {
  name: "QuoteView",
  data() {
    return {…
    };
  },
  computed: {
    photoUrl() {…
    },
    photoAuthor() {
      return this.quoteData.photoAuthor;
    },
    quoteText() {…
    },
    quoteAuthor() {
      return this.quoteData.quoteAuthor;
    },
  },
  mounted() {…
  },
};
</script>

<style scoped>
.quote-wrapper {
  display: flex;
  flex-direction: column;
  justify-content: center;
  min-height: 455px;
  align-items: center;
  box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1),
    0 2px 4px -1px rgba(0, 0, 0, 0.06);
}
```

Slika 16 Primjer tempalte sintakse unutar Vue komponente

5.4 Upravljanje stanjem – Vuex

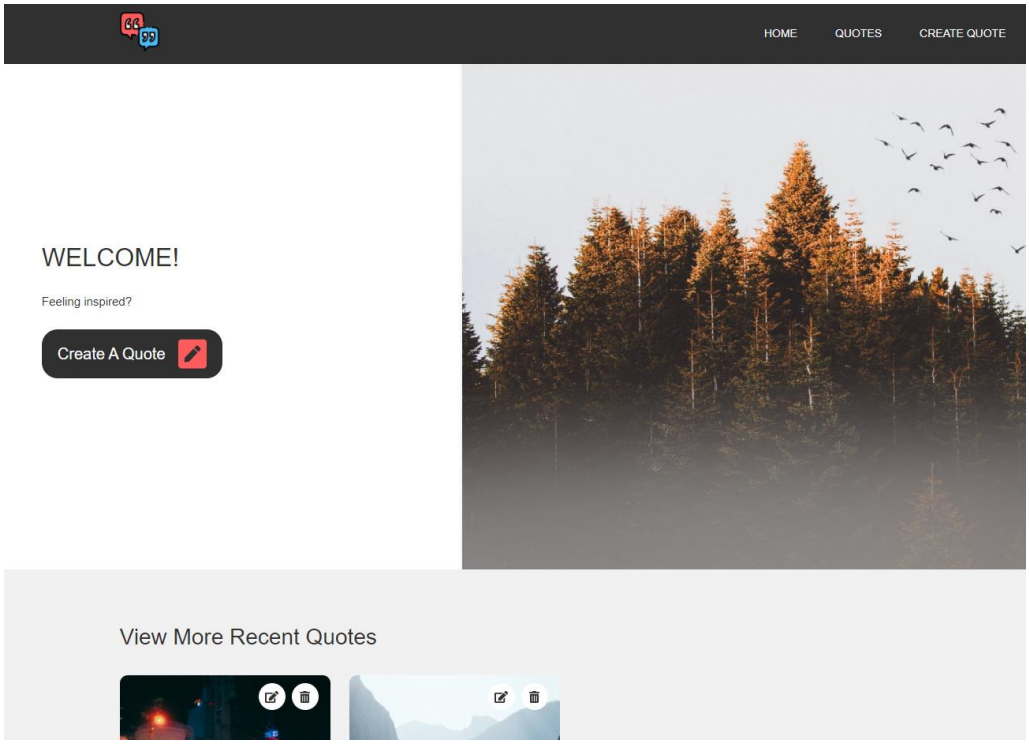
Za upravljanje stanjem korištena je Vuex knjižnica koja se sastoji od četiri dijela:

- State
- Actions
- Mutations
- Getters

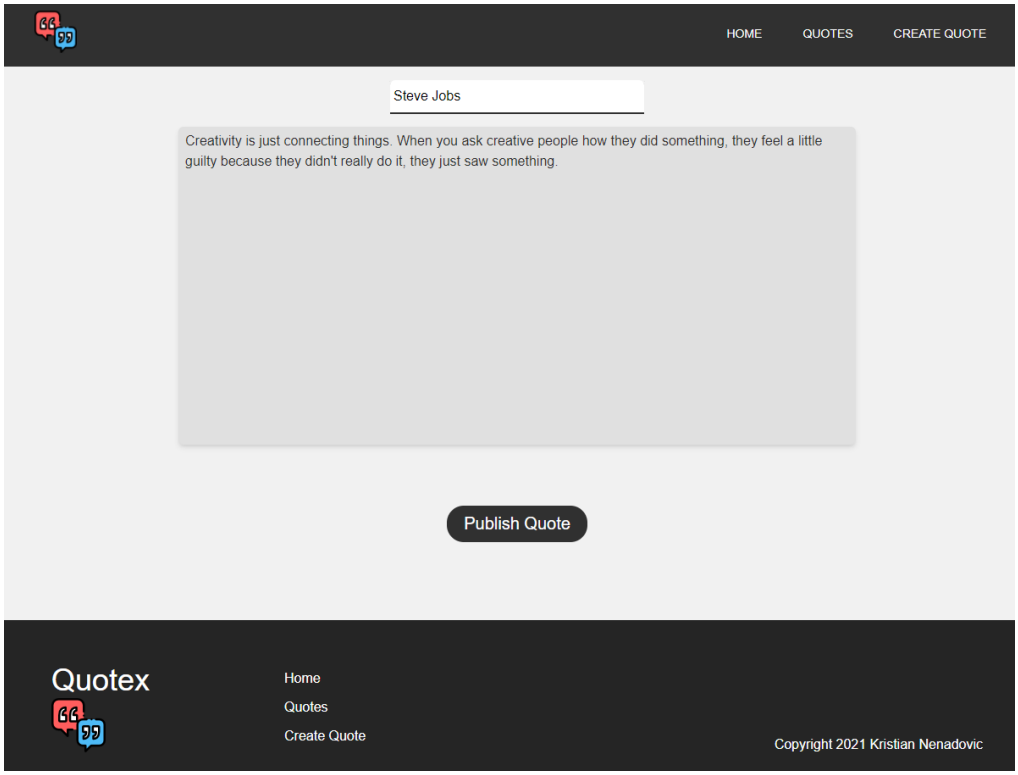
State dio predstavlja globalno stanje aplikacije kojemu mogu pristupiti sve komponente te ono što čini ovu knjižnicu vrlo moćnom je reaktivnost stanja koje efikasno sinkronizira sve komponente koje koriste te podatke. Princip jednosmjernog toka podataka osiguran je akcijama i mutacijama, čime je postignuto predvidivo mutiranje i lakše otklanjanje grešaka, a osim toga za dohvaćanje podataka iz stanja koriste se *getters* funkcije (slika br. 17). Unutar akcija, osim sinkronih operacija moguće je i obavljanje asinkronih kao što su API zahtjevi. Svaka akcija mora izvršiti (eng. commit) mutaciju te su one jedine funkcije koje mijenjaju podatke unutar stanja.

Slika 17 Primjer korištenja Vuex knjižnice

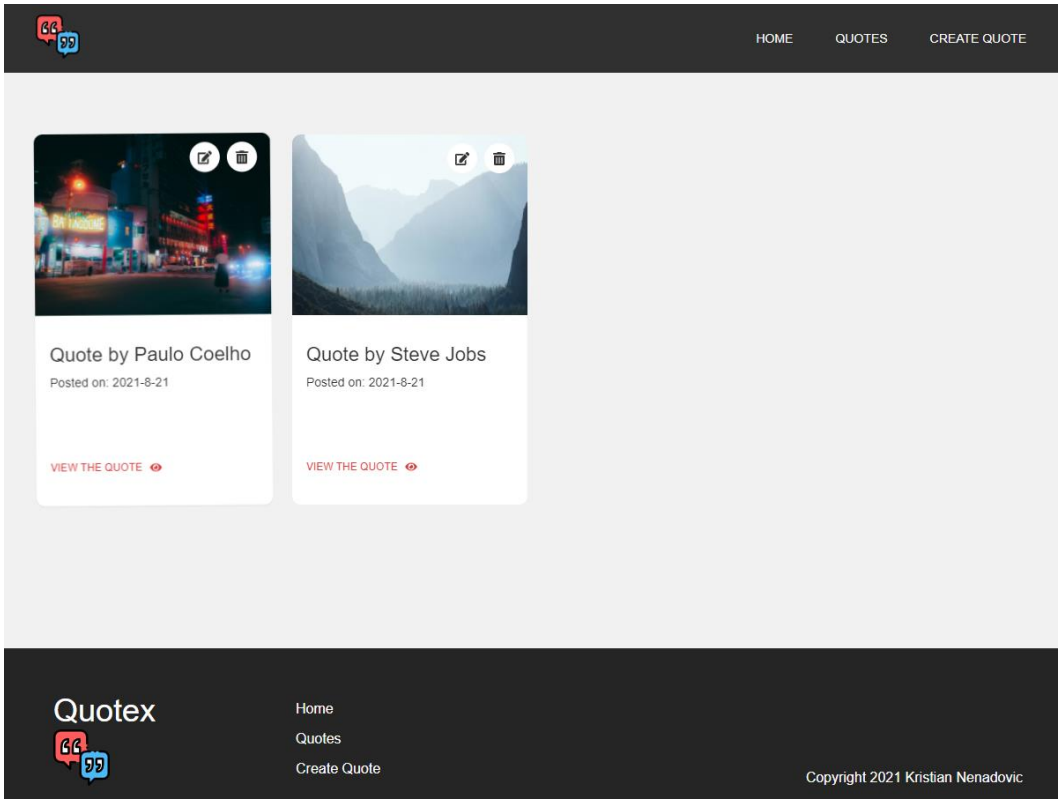
5.5 Izgled aplikacije



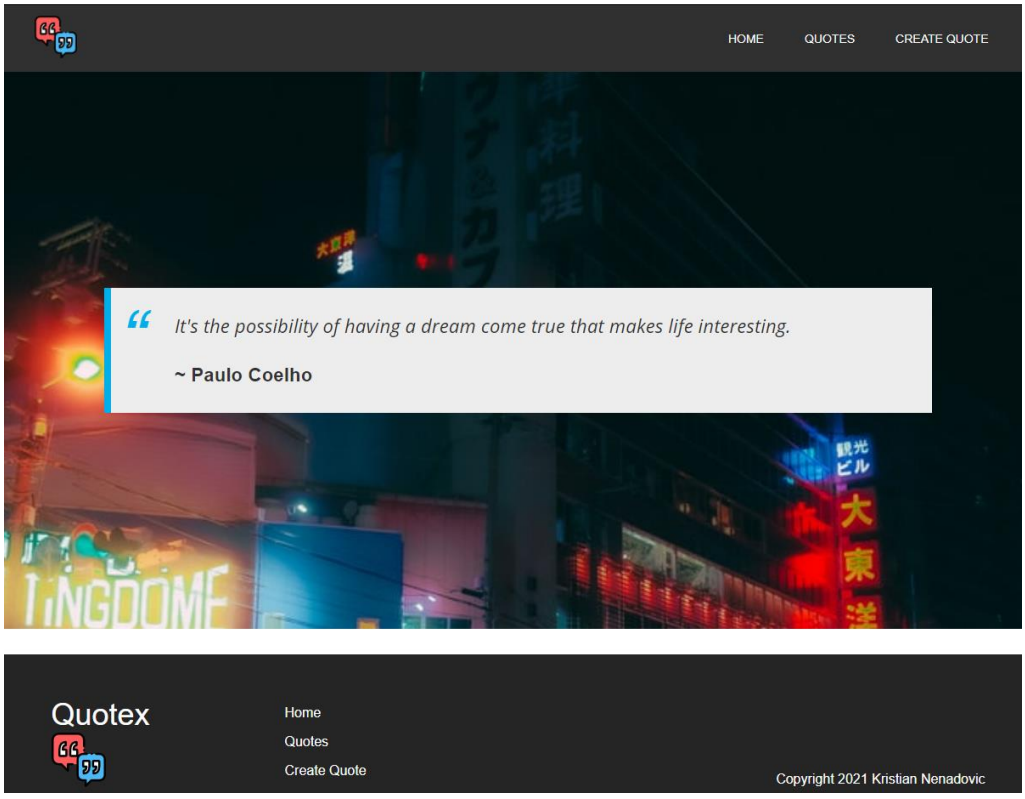
Slika 18 Početna stranica "Home"



Slika 19 Stranica za kreiranje citata



Slika 20 Stranica "Quotes" - pregled svih citata



Slika 21 Stranica za pregled pojedinačnog citata

6 ZAKLJUČAK

Pojava JavaScript programskog jezika znatno je utjecala na evoluciju industrije za razvoj web aplikacija što je dovelo do velike potražnje za novim programerima. Unazad nekoliko godina web aplikacije su postale mnogo kompleksne te su sukladno time stvorena nova rješenja i ideje kako bi se olakšao i pojednostavio razvoj te njihovo održavanje. Korištenje „čistog“ JavaScripta u većim i zahtjevnijim aplikacijama se ispostavilo mukotrpnijim te su se zbog toga počeli razvijati okviri. U posebnim poglavljima ovoga rada opisuju se važne definicije frontend okvira i prezentiraju osnovne funkcije modernih web aplikacija.

Različite aplikacije ne zahtijevaju uvijek iste potrebe i funkcionalnosti tako da je odabir primjerenog okvira od velike važnosti. Iako okviri brzo evoluiraju, učenje jednoga podrazumijeva i učenje zajedničkih koncepata pa je također napravljen i kratak pregled trenutno tri najpopularnija okvira: Angular, React i Vue.

Kako bi se dobilo bolje razumijevanje Vue okvira i dokazale njegove osobine, u posljednjem poglavlju je prezentirana aplikacija za kreiranje citata kojom su opisane njegove funkcije: upravljanje stanjem, preusmjeravanje, rad s komponentama te spajanje podataka.

7 LITERATURA

[1] Introduction to client-side frameworks, https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction

[2] Tutorial: Intro to React, https://reactjs.org/tutorial/tutorial.html

[3] Introduction to the Angular Docs, https://angular.io/docs

[4] Introduction to Vue, https://v3.vuejs.org/guide/introduction.html

[5] Model-view-viewmodel, https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel

[6] Model-view-controller, https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[7] Vue CLI, https://cli.vuejs.org/guide/#cli

[8] Vuex, https://vuex.vuejs.org/

[9] Flux in depth overview, https://facebook.github.io/flux/docs/in-depth-overview/

[10] Medium, The Missing Introduction to React by Eric Elliott URL: https://medium.com/javascript-scene/the-missing-introduction-to-react-62837cb2fd76

[11] Angular Minds, Comparison Between MVC Vs MVP Vs MVVM by Amit Khirale https://www.angularminds.com/blog/article/mvc-vs-mvp-mvvm.html

[12] Relevant, Angular vs. React vs. Vue.js – Choosing a JavaScript Framework for Your Project by Ihor Feoktistov, https://relevant.software/blog/angular-vs-react-vs-vue-js-choosing-a-javascript-framework-for-your-project/

8 POPIS KRATICA

SSR	Server Side Routing
DSL	Domain Specific Language
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
URL	Uniform Resource Locator
UI	User interface
HTTP	Hypertext Transfer Protocol
CLI	Command line interface
MVVM	Model-View-View Model
MVC	Model-View-Controller
API	Application Programming Interface
JSX	JavaScript XML
DOM	Document Object Model
SPA	Single Page Application
URL	Uniform Resource Locator
SSR	Server side rendering
AOT	Ahead Of Time

9 POPIS SLIKA