Razvoj web aplikacija korištenjem okvira Vue.js

Kekez, Matko

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matko Kekez

RAZVOJ WEB APLIKACIJA KORIŠTENJEM OKVIRA VUE.JS

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matko Kekez

JMBAG: 0016154820

Studij: Informacijske tehnologije i digitalizacija poslovanja

RAZVOJ WEB APLIKACIJA KORIŠTENJEM OKVIRA VUE.JS

ZAVRŠNI RAD

Mentor:

doc. dr. sc. Matija Novak

Varaždin, rujan 2024.

Matko Kekez

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Završni rad proučava razvoj klijentskih web aplikacija korištenjem Vue.js okvira. Radom će se istražiti kako je razvoj web tehnologija rezultirao povećanjem zahtjeva za složenijim i kvalitetnijim web aplikacijama, koji su izazovi u razvoju modernih web aplikacija, zašto se koriste okviri te koje su njihove prednosti i nedostatci. Nakon toga detaljno će se istražiti okvir Vue.js uključujući njegov utjecaj na arhitekturu aplikacije, razlog pojave okvira, sintaksu, karakteristike, najkorištenije biblioteke, komponente i sl. Teorijsko analiziranje okvira popraćeno je kratkim isječcima koda kojima se demonstrira praktično funkcioniranje okvira. U konačnici razvijena je kompletna web aplikacija domene novinskog portala s osnovnim funkcionalnostima pregledavanja, čitanja, komentiranja i stvaranja novinskih članaka i vijesti. Pri razvoju aplikacije naglasak je stavljen na klijentsku aplikaciju kako bi se demonstrirale sve funkcionalnosti Vue.js. Poslužiteljska aplikacija razvijena je korištenjem Node.js platforme i Express.js okvira, a baza podataka je SQLite datoteka. Kao rezultat kompletnog istraživanja okvira i njegove praktične primjene prilikom razvoja moderne web aplikacije na kraju rada iznesen je zaključak o okviru.

Ključne riječi: WEB aplikacija, JavaScript, Okvir za razvoj aplikacija, Vue.js, Reaktivnost, Deklarativno renderiranje, Komponenta, Arhitektura bazirana na komponentama, Vue Router, Vue

Sadržaj

1. Uvod

Završni rad proučava razvoj web aplikacija korištenjem popularnog okvira za razvoj klijentskih web aplikacija Vue.js. Prvo je potrebno objasniti što su web i internet s obzirom da je riječ o važnim tehnologijama u kontekstu rada, a ljudi često nisu svjesni razlike između njih.

Internet i web često se koriste kao sinonimi iako je riječ o različitim tehnologijama. Prema [1] internet je globalna mreža međusobno povezanih računala dok je web kolekcija hipertekstualnih (eng. *Hypertext*) dokumenata čije je stvaranje, spremanje, prikupljanje i pregledavanje omogućeno internetom. Hipertekstualni dokument (poznatiji i kao web stranica) oblik je medija čiji tekst sadrži poveznice na druge hipertekstualne dokumente na koje se trenutni referencira. Internet je dakle infrastruktura sklopovlja, programske podrške i protokola koja omogućava funkcioniranje web tehnologija koje su samo jedan od načina na koji se informacije mogu dijeliti putem interneta.

Očita razlika između ovih tehnologija njihova je povijest korištenja i vrijeme nastanka. Internet se koristi od 1969. godine kada se s računala Sveučilišta u Kaloforniji (eng. *University of California – UCLA*) putem ARPANET-a (tadašnji internet) poslala poruka računalu na Stanford istraživačkom institutu (eng. *Stanford Research Institute – SRI*) čime je realizirana prva uspješna razmjena poruka putem interneta. Ideja o webu je nastala 1989. kada je britanski računalni znanstvenik Tim Berners-Lee u svom prijedlogu „Information Management: A proposal“ predstavio rješenje za efektivno upravljanje informacijama Europske organizacije za nuklearna istraživanja koja je poznatija kao CERN. [1], [2]

Godinu kasnije Tim Berners-Lee razvio je četiri ključne komponente za funkcioniranje web-a [2] :

- Hypertext Markup Language (u nastavku akronim HTML) – jezik oznaka za razvoj hipertekstualnih dokumenata.
- Hypertext Transfer Protocol (u nastavku akronim HTTP) – protokol aplikacijskog sloja koji omogućuje prijenos hipertekstualnih dokumenata putem interneta.
- Uniform Resource Identifier (URI) – adresa koja se koristi kao jedinstveni identifikator resursa (tada samo hipertekstualnih dokumenata) na webu.
- World Wide Web (WWW) – prvi web preglednik koji je interpretirao sadržaj hipertekstualnih dokumenata i prezentirao informacije korisnicima.

Inicijalno su se na webu mogli pregledavati samo statični hipertekstualni dokumenti (u nastavku web stranice) koji su svakom korisniku prezentirali identičan sadržaj koji je stvorio autor stranice. Trend povećanja korištenja weba rezultira pojavom novih tehnologija koje

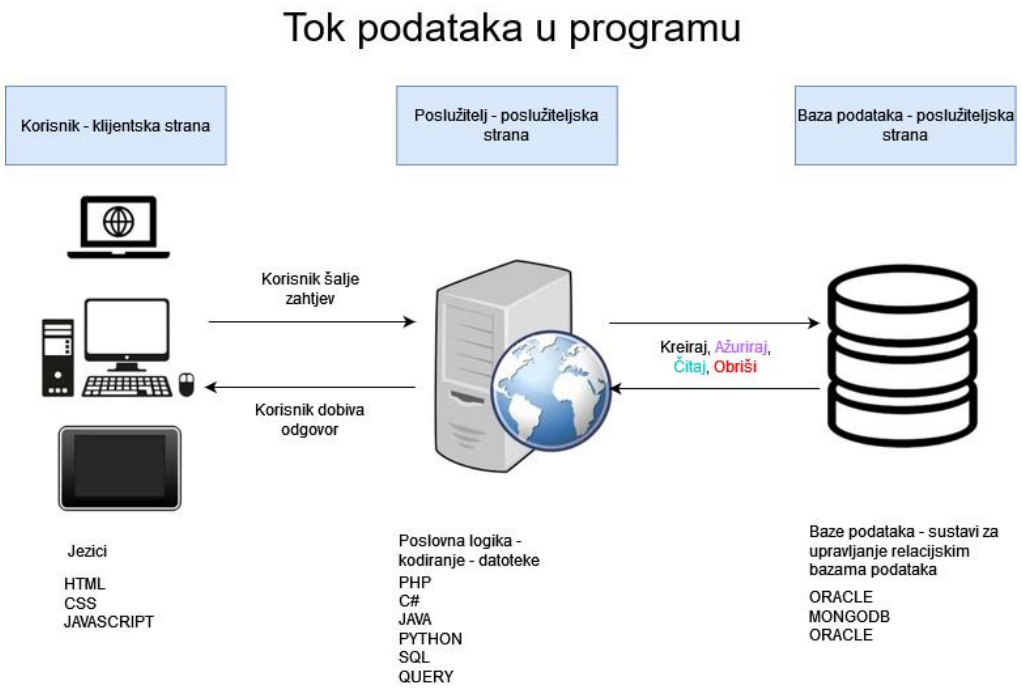
omogućuju razvoj dinamičkih web aplikacija koje implementiraju interaktivnost omogućujući stvaranje sadržaja od strane korisnika i prezentiraju drugačiji sadržaj svakom korisniku. Proporcionalno razvoju web tehnologija raste i njegova popularnost te se može reći da su danas sve društvene i poslovne aktivnosti ovisne o webu. To rezultira rastom zahtjeva za sve kvalitetnijim i funkcionalnijim web aplikacijama zbog čega se pojavljuju okviri poput Vue.js (u nastavku Vue) kako bi se razvojnim inženjerima olakšao razvoj modernih i složenih web aplikacija [1].

Rad u nastavku strukturiran je u 8 poglavlja. U 2. poglavlju opisana je arhitektura modernih web aplikacija te su istraženi izazovi koji se javljaju tijekom razvoja modernih aplikacija. U poglavlju 3 objašnjeni su softverski okviri te su analizirane njihove prednosti, nedostatci, vrste i sl. Također, u sklopu 2. poglavlja objašnjen je Vue okvir uključujući sintaksu okvira i njegove osnovne koncepte – reaktivnost i deklarativno renderiranje. Poglavlje 4 proučava komponente kao osnovne građevinske blokove Vue aplikacije te se razmatra komunikacija između komponenti, arhitektura bazirana na komponentama i koje ugrađene komponente Vue podržava. Poglavlje 5 proučava dvije najkorištenije službene Vue biblioteke – Vuex i Vue Router. Poglavlje 6 razmatra alat komandnog naredbenog sučelja koji se koristi za lakše upravljanje Vue projektom i jednostavnu konfiguraciju biblioteka i modula. U poglavlju 7 predstavljen je praktični dio rada što podrazumijeva web aplikaciju novinskog portala razvijenu u Vue i Node.js. U poglavlju 8 iznesen je konačan zaključak o modernim web tehnologijama i aplikacijama te kako tehnologije kao Vue okvir utječu na proces razvoja aplikacije.

2. Razvoj Web aplikacija

Kako je spomenuto u prethodnom poglavlju, pojava dinamičkih web aplikacija značajno popularizira web jer je korisnicima omogućeno stvaranje sadržaja na webu. Osim stvaranja sadržaja, web aplikacije implementiraju i druge funkcionalnosti kao što su dijeljenje sadržaja između korisnika, ažuriranje sadržaja u stvarnom vremenu, prikupljanje, spremanje i obrada korisničkih podataka, emitiranje videozapisa u stvarnom vremenu i sl. Implementacija tih funkcionalnosti omogućena je pojavom skriptnih programskih jezika kao što su JavaScript i Hypertext Preprocessor (u nastavku akronim PHP). [1]

Prema autoru [1] svaka se web aplikacija može podijeliti na dva dijela: klijentska strana (eng. *front-end* ili *client-side*) i poslužiteljska strana (eng. *back-end* ili *server-side*). Korisnici interakciju imaju samo s klijentskom stranom koja putem web preglednika prezentira informacije korisnicima i prihvaća njihove unose. Poslužiteljska strana poslužuje klijentsku stranu na način da upravlja HTTP/HTTPS zahtjevima i odgovorima te joj pruža sadržaj za prezentaciju, a isto tako prikuplja, obrađuje i pohranjuje sadržaj koji je korisnik unio na klijentskoj strani. Ovakvim razdvajanjem web aplikacije na slojeve postiže se modularnost i neovisnost slojeva što omogućuje jednostavniji razvoj, održavanje, skalabilnost, adaptivnost te osnažuje sigurnost i performanse web aplikacije. Slika 2. vizualizira tok podataka između klijentske i poslužiteljske strane te tehnologije korištene za njihov razvoj.



Slika 1: Vizualizirani prikaz toka podataka između klijentske i poslužiteljske strane [3]

2.1. Programiranje na strani klijenta

Programiranje na strani klijenta podrazumijeva razvoj dijela aplikacije s kojim korisnici imaju direktnu interakciju te se stoga velika pažnja prilikom razvoja posvećuje korisničkom iskustvu (eng. *User experience – UX*) i dizajnu korisničkog sučelja (eng. *User interface design – UI*). Kvalitetnim korisničkim iskustvom i dizajnom sučelja osigurava se razmještaj objekata sučelja koji korisniku omogućuje intuitivno korištenje odnosno jednostavnu navigaciju kroz aplikaciju i izvršavanje akcija. Dizajnom se isto tako mora osigurati prilagodljivost (u struci se koristi termin responzivnost) sučelja različitim dimenzijama ekrana te psihologija boja i uređenje teksta koji odgovaraju namjeni aplikacije. [1]

Tehnologije koje se koriste za razvoj klijentske strane su HTML za strukturiranje sadržaja aplikacije (najnoviji standard HTML5), CSS za dizajn sučelja aplikacije (najnoviji standard CSS3) i JavaScript za dinamičku promjenu strukture i dizajna te implementaciju funkcionalnosti (najnoviji standard ECMAScript 2023 – ES24). [1]

Problem JavaScripta je u tome da nije tipiziran što često uzrokuje pogreške tijekom pisanja koda i tijekom izvršavanja funkcionalnosti aplikacije. Microsoft je stoga 2012. predstavio programski jezik TypeScript koji je sličan JavaScriptu, a podržava tipizaciju olakšavajući proces razvoja. Važno je za napomenuti da TypeScript i JavaScript podržavaju isti standard (ECMAScript). [4]

2.2. Programiranje na strani poslužitelja

Programiranje na strani poslužitelja podrazumijeva razvoj ključnih funkcionalnosti koje implementiraju logiku aplikacije kojom je definirana obrada podataka koji se dohvaćaju iz spremišta podataka te se nakon obrade prezentiraju korisnicima na klijentskoj strani ili se dobivaju od klijentske strane i nakon obrade pohranjuju u spremište. Za spremište podataka uglavnom se koriste relacijske baze podataka ili u novije vrijeme sve popularnije baze podataka koje nisu temeljene na strukturnom upitnom jeziku (eng. *Structured Query Language* – u nastavku akronim SQL) poznatije i kao NoSQL baze (eng. *non-SQL* ili *not only SQL*). [1]

Strana poslužitelja podatke može dohvaćati i sa različitih servisa putem Aplikativnog Programskog Sučelja (eng. *Application Programming Interface* – u nastavku akronim API) nakon čega ih obrađuje i šalje klijentu. Prema [1] API je standardizirana kolekcija naredbi, akcija, pravila i tipova podataka koja omogućuje komunikaciju između različitih aplikacija. Korištenjem API-ja aplikacija komunicira sa vanjskim servisom ili drugom aplikacijom od kojih dobiva podatke. Vanjski servisi mogu sadržavati velike kolekcije podataka koji se ažuriraju u

stvarnom vremenu te njime upravlja treća strana što smanjuje napore ili u potpunosti eliminira potrebu upravljanja pohranom podataka. Podaci se osim poslužitelju mogu direktno slati klijentskoj aplikaciji, no ako zahtijevaju bilo kakvu obradu praksa je da se šalju prvo poslužitelju te nakon obrade klijentu. Servisi osim podataka nude i integraciju različitih funkcionalnosti pa tako Stripe i PayPal nude API čija integracija s aplikacijom web dućana pruža funkcionalnost sigurnog elektroničkog plaćanja. Važno je za napomenuti da API često nije besplatan i iziskuje visoke troškove naknade za njegovo korištenje, no prednost se očituje u tome da nudi provjerene i pouzdane funkcionalnosti čime se eliminira potreba za razvojem i održavanjem vlastitih za koje možda razvojni inženjeri unutar tima nisu specijalizirani (nije njihova domena) pa se proporcionalno tome smanjuju troškovi razvoja i održavanja.

Poslužiteljska strana dakle implementira logiku obrade podataka koja je korisniku skrivena, a rezultate obrade šalje klijentskoj strani koja ih prezentira korisnicima te prihvaća njihove unose koje opet šalje poslužiteljskoj strani na daljnju obradu. Ovisno o funkcionalnim i nefunkcionalnim zahtjevima aplikacije odabire se ona tehnologija koja će najbolje implementirati te funkcionalnosti. Programski jezici koji su se u najvećoj mjeri koristili za razvoj poslužiteljske strane su Python, PHP, Java, Ruby i C#. [1]

JavaScript nije bio pogodan za razvoj poslužiteljske strane sve do 2009. kada je američki razvojni inženjer Ryan Dahl predstavio platformu Node.js koja je prema [5] popularizirala korištenje JavaScripta/TypeScripta i na poslužiteljskoj strani.

2.3. Izazovi u razvoju modernih web aplikacija

Proporcionalno povećanju obujmu korištenja i ovisnosti društvenih i poslovnih aktivnosti za webom, porasli su i zahtjevi za pouzdanim web aplikacijama te proces razvoja tako postaje složen i dinamičan zahtijevajući visoku stručnost, pažljivo planiranje i odabir odgovarajućih tehnologija. Upravljanje cjelokupnim projektom razvoja aplikacije predstavlja veliki izazov jer uključuje popriličan broj aktivnosti – od analize funkcionalnih i nefunkcionalnih zahtjeva aplikacije do distribucije aplikacije korisnicima i njezino održavanje. Aplikacija mora ispuniti promjenjive (poslovne) zahtjeve korisnika te mora biti razvijena unutar definiranog budžeta i vremenskog roka što predstavlja izazov, ne samo razvojnim inženjerima, već kompletnom projektnom timu.

Dodatni izazov predstavlja odabir odgovarajućeg tehnološkog stoga (eng. *tech stack*) koji odgovara funkcionalnim i ne funkcionalnim zahtjevima aplikacije te planiranom budžetu za njen razvoj. Prema [6] tehnološki stog odnosi se na primjenu kompatibilnih tehnologija za razvoj kao što su programski jezici, okviri, biblioteke, API servisi, usluge računalstva na oblaku,

razvojna okruženja i sl. Najveći izazov razviti je web aplikaciju čijom će se arhitekturom osigurati [6] :

- Dobre performanse – brzo učitavanje aplikacije u pregledniku te smanjenje vremena potrebnog za obavljanje funkcionalnosti aplikacije (obrade podataka). Loše optimizirana aplikacija rezultira frustracijom korisnika što dovodi do prekida njenog korištenja. Performanse moraju biti dobre i kada aplikaciju koristi veliki broj korisnika istovremeno.
- Visoka dostupnost (eng. *availability*) – cilj dostupnosti osigurati je dugotrajnu operativnost aplikacije odnosno smanjivanje broja incidenata (pogrešaka) koji uskraćuju dostupnost korištenja aplikacije krajnjim korisnicima.
- Kompatibilnost – aplikacija mora biti kompatibilna različitim platformama odnosno njeno izvođenje ne smije biti ograničeno tehnologijama (sklopovlju i operacijskim sustavom) kako bi ostala dostupna širokom krugu korisnika.
- Jednostavno održavanje – arhitektura aplikacije mora biti slojevita te ovisnost između slojeva bi trebala biti minimalna kako bi se olakšalo njihovo održavanje i testiranje. Razvojni inženjeri moraju se pridržavati najboljih praksi kodiranja prilikom razvoja kako bi kod dugoročno bio jednostavan za čitanje i održavanje.
- Sigurnost – jedan od najvećih izazova u razvoju modernih web aplikacija s obzirom da zahtjevi za sigurnošću naglo rastu zbog velike stope kibernetičkih napada i zakonskih regulativa (NIS direktiva).

3. Softverski okviri

Jedan od najvećih troškova razvoja aplikacija dosta dugo bio je trošak vremena i napora potrebnih za kodiranje aplikacije. Većina aplikacija implementira neke slične ili jednake funkcionalnosti čiji je kod identičan te je ideja biblioteka i softverskih okvira smanjiti vrijeme i napore potrebne za razvoj tih funkcionalnosti. Biblioteke i okviri pružaju unaprijed pripremljeni kod koji pojednostavljuje i ubrzava proces razvoja aplikacije. Iako je ideja ista, biblioteka i okvir nisu sinonimi. Prema [1] biblioteka se definira kao kolekcija pripremljenog koda odnosno funkcija i klasa koje se mogu iskoristiti u razvoju novih aplikacija. Na primjer JavaScript biblioteka „moment“ implementira funkciju „format()“ kojom se formatira datum te se ona poziva gdje god je to potrebno u kodu tako da razvojni inženjer ne mora samostalno razvijati funkciju za formatiranje datuma. Softverski okvir (eng. *software framework*) se prema [7] definira kao skup alata, biblioteka, najboljih praksi i smjernica koji razvojnim inženjerima pruža strukturu i temelje za razvoj aplikacija. Jednostavnije rečeno okvir se može definirati kao ponovo iskoristiva struktura arhitekture aplikacije dok biblioteka podržava ponovno iskoristiv kod. Iz navedenog se može zaključiti da okvir ograničava fleksibilnost prilikom razvoja aplikacije jer diktira dizajn arhitekture dok to nije slučaj kod biblioteke. Okviri ubrzavaju i olakšavaju razvoj robusnih aplikacija i nude mnoge druge benefite, ali isto tako mogu biti ograničavajući i izazvati određene probleme. Ključni benefiti korištenja okvira minimiziranje je redundantnosti koda, smanjenje vremena i napora potrebnih za razvoj, razvojni inženjeri se mogu usredotočiti na specifične funkcionalnosti aplikacije, inženjeri koji nisu pisali kod lakše ga čitaju, testiraju i uklanjaju pogreške i sl. Važno je za napomenuti da okviri omogućuju razvoj aplikacija za određene platforme tehnologijama koje nisu nativno podržane tim platformama. Najbolji primjer je Java okvir Vaadin koji omogućuje razvoj klijentskih web aplikacija iako preglednici ne mogu razumjeti sintaksu Jave te se ona nikada nije koristila za razvoj klijentskih web aplikacija. U pozadini Vaadin dinamički generira Java kod u HTML, CSS i JavaScript koji šalje preglednicima na renderiranje. Okviri tako proširuju specijaliziranost razvojnih inženjera i timova pa onaj inženjer koji je razvijao isključivo stolne aplikacije ili poslužiteljsku stranu web aplikacije korištenjem okvira poput Vaadina može razvijati i klijentsku web aplikaciju bez potrebe za učenjem HTML-a, CSS-a i JavaScripta. [7]

Okvirima upravlja treća strana što se može smatrati prednošću s obzirom da je riječ o stručnjacima koji kontinuirano rade na poboljšavanju okvira čime inženjerima olakšavaju razvoj, no isto tako se može smatrati nedostatkom jer operativnost aplikacija koje su razvijene i održavane okvirom postaje ovisna o poslovnim politika i strategijama vlasnika okvira – okvir se može prestati razvijati ili se može početi naplaćivati (Sencha Ext JS je u početku bio besplatan okvir otvorenog koda čije se korištenje vremenom počelo naplaćivati). Dodatan

nedostatak okvira je što zahtijevaju vrijeme i napore za njihovo učenje, a krivulje učenja su za većinu nerijetko strme.

Ovisno o ciljanoj platformi, namjeni i veličini aplikacije, budžetu, vremenskom roku te zahtjevima za održavanjem koristi se odgovarajući okvir. Za razvoj web aplikacija okviri se mogu podijeliti na okvire za razvoj klijentske aplikacije (eng. *front-end framework*) i okvire za razvoj poslužiteljske aplikacije (eng. *back-end framework*). [7]

Tablica 1 prikazuje programske jezike i pripadajuće okvire za razvoj klijentske aplikacije, a tablica 2 prikazuje programske jezike i pripadajuće okvire za poslužiteljsku aplikaciju.

Tablica 1 Programski jezici i pripadajući okviri za razvoj klijentske web aplikacije

Jezik	**Okvir**
JavaScript/TypeScript	React, Angular, Vue.js
HTML/CSS	Bootstrap, Foundation, Tailwind CSS, Bulma
Dart	Flutter Web

Autorski rad prema podacima [1], [7]

Tablica 2 Programski jezici i pripadajući okviri za razvoj poslužiteljske web aplikacije

Jezik	**Okvir**
JavaScript/TypeScript – Node.js	Express.js, NestJS, Koa.js, Meteor
Python	Django, Flask, FastAPI
Ruby	Ruby on Rails
Java	Spring Boot, Jakarta EE
PHP	Laravel, Symfony, CodeIgniter
C#	ASP.NET Core

Autorski rad prema podacima [5], [7]

3.1. Vue.js

Vue je progresivni JavaScript okvir otvorenog koda razvijen kao supstitut za stroge okvire sa strmom krivuljom učenja kao što su Angular i React. Vue je okvir za razvoj klijentske web aplikacije te pruža deklarativni programski model temeljen na komponentama za razvoj interaktivnih korisničkih sučelja i jednostraničnih web aplikacija (eng. *Single Page Application - SPA*). Razvio ga je kineski programer Evan You koji je kao zaposlenik Googlea koristio Angular te u [8] govori kako je uočio određene prednosti i nedostatke koje Angular nudi te se odlučio razviti novi okvir koji će istovremeno implementirati te prednosti i eliminirati nedostatke. Vue tako podržava koncepte Angulara koje autor okvira smatra korisnima, a to su arhitektura bazirana na komponentama, direktive koje omogućuju manipulaciju objektnim modelom dokumenta (eng. *Document Object Model* – u nastavku akronim DOM) te reaktivnost koja omogućuje vezivanje podataka (eng. *data binding*). Vue ne definira strogu strukturu arhitekture aplikacije kao što je slučaj kod Angulara nudeći time veću fleksibilnost i slobodu programerima. Autor okvira u [8] govori kako je Vue.js inicijalno razvio za vlastite potrebe te kada je uočio potencijal okvira odlučio je svoj rad podijeliti s drugima pa je 2014. objavio prvu verziju izvornog koda na platformi GitHub nakon čega se okvir počeo koristiti.

Vue je progresivan okvir što znači da programerima omogućuje inkrementalnu nadogradnju aplikacija bez potrebe za ponovnim pisanjem većeg dijela koda. Isto tako dobro se izvodi bez obzira na tehnološko okruženje i platformu, pruža dobre performanse te se trudi pružiti alternativna rješenja za one funkcionalnosti aplikacije koje preglednik na kojemu se aplikacija izvodi zbog zastarjelosti ne podržava. Vue se može integrirati u postojeće sustave koji koriste različite biblioteke ili okvire čime se postiže jednostavna modernizacija i nadogradnja aplikacija, a upravo u tu svrhu ga prema [9] koriste ga velike tehnološke organizacije kao što su Facebook, Netflix, Google, Apple, Alibaba i sl.

Krivulja učenja je blaga što ga čini privlačnim za učenjem početnicima koji nemaju puno iskustva u razvoju složenih aplikacija. Službena dokumentacija [10] prilično je opširna i lako razumljiva zbog čega je često primarni izvor prilikom učenja okvira. Okvir je iznimno popularan među programerima u Kini s obzirom da je kreator jedan od prepoznatljivih kineskih poduzetnika jednoroga što rezultira time da je dokumentacija za određene dodatke i biblioteke napisana samo na kineskom jeziku. Isto tako mnoge Vue konferencije i zajednice se služe primarno kineskim jezikom za komunikaciju. [8]

Sve navedene prednosti uočile su i mnoge prepoznatljive tehnološke organizacije koje su Vue okvir iskoristile u razvoju svojih aplikacija [9] :

- Facebook – iako je većina aplikacije razvijena u React-u, dio koji prezentira popis vijesti i objava razvijen je u Vue.
- Xiaomi – poznato da je koristi Vue ali nije poznato za koje proizvode.
- Trivago – aplikacija Trivago Magazine razvijena je korištenjem Nuxt.js i Vue-a.
- GitLab – web Git repozitorij za lakše upravljanje i kolaboraciju tijekom projekta razvoja aplikacija razvijen je u Vue. Izvorni kod dostupan je na https://gitlab.com/gitlab-org/gitlab.
- Nintendo – Nintendove službene stranice za Francusku, Njemačku, Španjolsku i Ujedinjeno kraljevstvo razvijene su u Vue.
- BMW – aplikacija „BMW Car Configurator“ nudi mogućnost prilagodbe opreme te dizajna interijera i eksterijera automobila prema vlastitim preferencijama što je razvijeno Vue-om.
- 9GAG – najpopularnija web platforma za zabavu razvijena je korištenjem različitih okvira uključujući i Vue.

3.1.1. Verzije okvira

Prema [11] broj Vue verzije strukturiran je kao velika.maka.zakrpa. Prvi broj podrazumijeva novu verziju s velikim promjenama. Od 2014. do 2024. promijenjene su 3 velike verzije Vue-a. Kompletni je okvir 2016. napisan ispočetka te je izdana verzija Vue2 kojim su predstavljene bolje performanse, bolja podrška za TypeScript te virtualni objektni model dokumenta (eng. *Virtual Document Object Model – Virtual DOM*). Godine 2020. izdana je verzija Vue3 koja je ujedno i trenutna velika verzija. Vue3 nadogradio je podršku za TypeScript te je uveo novu važnu značajku zvanu kompozicijsko aplikativno programsko sučelje (eng. *Composition API*) kojom je unaprijeđena sintaksa okvira.

Drugi broj odnosi se na promjene uvođenja manje važnih značajki ili poboljšanja koji su kompatibilni s trenutnom velikom verzijom. Treći broj odnosno „zakrpa“ (eng. *patch*) predstavlja male modifikacije koje otklanjaju otkrivene pogreške. Ne postoji fiksno definiran interval izdavanja nove verzije Vue-a (ni za velike ni za male promjene). Izdavanje nove velike verzije se pravovremeno najavljuje i ona uvijek prolazi kroz faze rasprave te alfa i beta faze korištenja i testiranja. Nove male verzije obično se izdaju svakih 3-6 mjeseci te obvezno prolaze kroz beta fazu korištenja i testiranja. Zakrpe se izdaju po potrebi te ne prolaze kroz beta i alfa faze korištenja i testiranja. Trenutna verzija Vue-a je 3.4.31. [11]

3.2. Reaktivnost i deklarativno renderiranje

Prema [10] reaktivnost i deklarativno renderiranje su dva temeljna mehanizma Vue okvira. Deklarativno renderiranje proširuje sintaksu HTML-a omogućujući deklarativno renderiranje objektnog modela dokumenta (eng. *Document Object Model* – u nastavku akronim DOM) temeljem podataka pruženih okviru i stanja definiranog JavaScript izrazima. Deklarativnim renderiranjem definira se struktura DOM-a koja se ažurira sukladno detektiranim promjenama u podacima i stanju aplikacije za čiju je detekciju zadužena reaktivnost. Reaktivnost je mehanizam kojim Vue automatski prati promjene podataka i stanja aplikacije osiguravajući sinkronizaciju podataka dohvaćenih iz spremišta s podacima koji se prikazuju na korisničkom sučelju. Kombinacijom ova dva mehanizma ostvaruje se automatsko praćenje promjene podataka i stanja aplikacije te renderiranje DOM-a sukladno detektiranim promjenama. Reaktivnost i deklarativno renderiranje znatno štede napore i vrijeme razvojnih inženjera jer eliminiraju potrebu za ručnim pisanjem funkcija koje prate promjenu podataka i sukladno tome manipuliraju DOM elementima.

Važno je za napomenuti da iako je zadatak reaktivnosti uvijek jednak, njeno ponašanje ovisi o Vue API-ju koji se koristi u projektu. Vue podržava dva API-ja: opcijski API (eng. *Options API*) i kompozicijski API (eng. *Composition API*). Više o razlikama između navedenih API-ja u nastavku. Ako se koristi opcijski API reaktivna svojstva se definiraju unutar data() funkcije koja vraća objekte koji pohranjuju stanje komponente. Vue poziva data() funkciju onda kada se kreira nova instanca komponente nakon čega sustav reaktivnosti automatski počinje pratiti objekte koje je funkcija vratila. Sukladno detektiranim promjenama stanja objekata ažurirati će se DOM. [12] Svojstva definirana u data() funkciji inicijaliziraju se u trenutku kreiranja komponente zbog čega je važno da im se dodjeli vrijednost. Ako vrijednost još nije poznata [10] preporučuje korištenje null ili undefined stanja ili praznog tekstualnog niza.

```
<body> <div id="app"> <p>{{ poruka }}</p>
<p>{{korisnik?`Korisnik:${korisnik.ime}`:'Anonimni
korisnik'}}</p> <button @click="azurirajKor">Azuriraj</button>
</div>
<script> new Vue({
      el: '#app', data(){
      return { poruka: 'Pozdrav svijetu', korisnik: null, };},
      methods:{
            azurirajKor(){
```

```
            this.korisnik = {
            ime: 'Matko'};}, }});
</script> </body>
```

Prethodni primjer demonstrira reaktivnost korištenjem opcijskog API-ja – funkcija data() vraća objekte poruka i korisnik. Pretpostavlja se da za korisnika ne postoje podatci prilikom kreiranja Vue instance te se stoga za vrijednost dodjeljuje null stanje. Unutar methods svojstva definirana je funkcija kojom se objektu korisnik na svojstvu ime dodjeljuje vrijednost „Matko“ onda kada se stisne gumb „Azuriraj“. Reaktivnost će tu promjenu detektirati i sukladno tome ažurirati DOM.

Kod kompozicijskog API-ja se koristi ref() metoda koja prima određenu vrijednost kao argument te ju pridružuje objektu. Za pristup ref() funkciji unutar komponente koristi se setup() funkcija koja vraća deklarirani ref() unutar svog bloka. [12]

```
<body><div id="app"><p>{{ poruka }}</p> <p>{{ korisnik ?
`Korisnik: ${korisnik.ime}` : 'Anonimni korisnik'}}
</p><button @click="azurirajKor" >Azuriraj</button> </div>
<script>const { createApp, ref } = Vue;createApp({
setup(){
const poruka = ref('Pozdrav svijetu');const korisnik = ref(null);
const azurirajKor = () => { korisnik.value = { ime: 'Matko' };};
return{poruka,korisnik,azurirajKor};
}}).mount('#app');</script>
</body>
```

Prethodni primjer demonstrira reaktivnost korištenjem kompozicijskog API-ja. Unutar setup() funkcije definirani su objekti tako što su ref() funkcijama proslijeđene njihove vrijednosti. Funkcionalno je isječak koda identičan isječku u kojem se koristio opcijski API. Umjesto setup() funkcije reaktivni objekti mogu se definirati unutar <script setup> oznake što demonstrira primjer u nastavku. Funkcionalno su isječci isti, no korištenjem <script setup> oznake kod djeluje čišćim.

```
<template><div><p>{{ poruka }}</p>
<p>{{ korisnik ? `Korisnik: ${korisnik.ime}` : 'Anonimni
korisnik' }}</p>
<button @click="azurirajKor">Azuriraj</button></div>
</template><script setup>import { ref } from 'vue';
```

```
const poruka = ref('Pozdrav svijetu');const korisnik = ref(null);
const azurirajKor = () => { korisnik.value = { ime: 'Matko'
};};</script>
```

Bez obzira koristi li se opcijski ili kompozicijski API sustav reaktivnosti može pratiti sve promjene nad svim objektima i strukturama podataka pa čak i onima koji su duboko ugniježđeni u ref() i data() funkcije. To se svojstvo prema [10] imenuje terminom dubinska reaktivnost (eng. *Deep reactivity*).

3.3. Sintaksa okvira

Vue sintaksa je bazirana na HTML-u što znači da okvir može interpretirati bilo koji standardni preglednik ili HTML raščlanjivač (eng. *parser*) čime se osigurava kompatibilnost okvira s različitim platformama i alatima. Isto tako programerima olakšava pisanje i čitanje okvira s obzirom da je sintaksa HTML-a prilično jednostavna. U pozadini Vue HTML oznake konvertira u optimizirani JavaScript kod. [13]

3.3.1. Interpolacija

Interpolacija se koristi za prikazivanje podataka unutar HTML elemenata omogućujući automatsko ažuriranje korisničkog sučelja onda kada reaktivnost detektira promjenu stanja komponente. Interpolacija se označava dvostrukim otvorenim i zatvorenim vitičastim zagradama unutar HTML elementa kao što je prikazano u sljedećem primjeru. [13]

```
<div id="app"> <p>{{ poruka }}</p></div>
<script>new Vue({el: '#app',data: {poruka: 'Interpolacija!'}})
</script>
```

Na prvoj liniji koda prikazana je sintaksa interpolacije korištenjem dvostrukih vitičastih zagrada unutar <p> elementa. Unutar vitičastih zagrada smješten je objekt poruka s vrijednošću „Interpolacija!“ te će se u pregledniku stoga ispisati „Interpolacija!“. Ako se tijekom izvođenja aplikacije vrijednost objekta poruka promjeni iz „Interpolacija!“ u „Matko!“ reaktivnost će tu promjenu detektirati te će se ispisati nova vrijednost.

3.3.2. Direktive

Direktive su specijalni atributi koji počinju s „v-“ prefiksom te se koriste za ažuriranje elemenata DOM-a onda kada se vrijednost izraza direktive promjeni. Sintaksa direktive piše se u obliku: „v-nazivDirektive: izraz“. Izraz može biti varijabla, operacija, poziv funkcije ili bilo

koji drugi JavaScript izraz koji se evaluira i izvršava unutar komponente. Elementi DOM-a koji sadrže direktive dinamički se mijenjaju ovisno o rezultatu izvršavanja JavaScript izraza. [13]

Programer po potrebi može razviti vlastitu direktivu iako Vue već podržava velik broj ugrađenih direktiva od kojih se najčešće koriste [12] :

- v-bind – jednosmjerno veže JavaScript izraze ili objekte kao vrijednosti HTML atributa unutar elementa.
- v-model – dvosmjerno veže podatke između HTML elemenata i objekata. Zbog dvosmjernog povezivanja podatci iz komponente su sinkronizirani s onima na korisničkom sučelju pa se uvijek koristi u obrascima za unos podataka.
- v-if – renderira HTML element ovisno o vrijednosti JavaScript izraza ili objekta. Ako je vrijednost izraza ili objekta istinita, element se renderira i prikazuje na ekranu, a ako nije element se ne prikazuje na ekranu. Koncept je identičan if(izraz){…} grani.
- v-else – dodatak v-if direktivi koji funkcionira kao else{…} grana – ako izraz v-if nije istinit renderira se element s v-else direktivom.
- v-else-if – dodatak v-if direktivi koji funkcionira kao else if(izraz){…} grana – ako vrijednost izraza v-if nije istinit provjerava se izraz v-else-if direktive te se renderira ovisno o istinitosti izraza. Ako ni v-if ni v-else-if nisu istiniti renderira se element s v-else direktivom.
- v-for – direktiva koja iterira kroz elemente polja ili objekta te proporcionalno tome renderira listu HTML elemenata. Najčešće se koristi unutar <li> elementa te za svaki element objekta renderira novu stavku liste.
- v-on – na element pričvršćuje slušača događaja za izvršavanje JavaScript funkcije kada se događaj dogodi. Umjesto v-on direktive može se koristi znak „@“ koji se jednako interpretira pa se u kodu često može vidjeti direktiva @click koja sluša događaj klika miša na element.

Direktiva v-bind najčešće je korištena direktiva te je na njenom primjeru u nastavku objašnjen način funkcioniranja svake Vue direktive. Direktiva v-bind veže HTML atribut s određenim izrazom što omogućuje dinamičku promjenu vrijednosti atributa ovisno o izrazu u Vue komponenti. Zbog čestog korištenja direktive praksa je da se umjesto njenog punog naziva koristi samo znak dvotočka koji se jednako interpretira. Direktiva se najčešće se koristi za dinamičko ažuriranje src, href, class i style HTML atributa. [13]

```
<div id ="app">
<img :src="slikaSrc"> <!-- ILI <img v-bind:src="slikaSrc"> -->
</div><script>new Vue({el: '#app',data: {
```

```
slikaSrc: 'https://vuejs.org/images/logo.png'}});</script>
```

Slika u aplikaciji će se dinamički promijeniti ovisno o promjeni vrijednosti putanje na svojstvu slikaSrc. Svaka Vue direktiva ima specifični slučaj korištenja ali sve funkcioniraju na isti način – reagiraju na promjene stanja i sukladno tim promjenama ažuriraju DOM.

3.3.3. JavaScript izrazi

Vue podržava korištenje JavaScript izraza unutar interpolacije i unutar vrijednosti atributa na koji je pričvršćena v-bind direktiva. JavaScript izrazi koji se koriste imaju pristup svim objektima i funkcijama unutar komponente pa se tako mogu provoditi složene manipulacije nad DOM-om ovisno o podacima unutar komponente. [13]

Kod u nastavku prikazuje primjer korištenja JavaScript izraza unutar interpolacije. Unutar interpolacije <p> elementa pristupa se poruka objektu te se njegova vrijednost transformira u velika slova korištenjem ugrađene JavaScript funkcije toUpperCase(). Tako će se umjesto stvarnog teksta ispisati tekst „JAVASCRIPT IZRAZ PRIMJER“. Unutar drugog <p> elementa provjerava se istinitost objekta jeVidljiv te se proporcionalno tome ispisuje odgovarajuća poruka. Na isti način se može pristupiti bilo kojoj funkciji ili objektu.

```
<div id="app"> <p>{{ poruka }}</p><p>{{ poruka.toUpperCase()
}}</p> <p>{{ jeVidljiv ? 'Vidljiv' : 'Skriven' }}</p> </div>
<script> new Vue({ el: '#app', data: { poruka: 'JavaScript izraz
primjer',jeVidljiv: true}});</script>
```

Kod u nastavku prikazuje primjer korištenja JavaScript izraza kao vrijednosti atributa korištenjem v-bind direktive. Izraz u „alt“ atributu određuje da će se alternativni tekst slike ispisivati velikim slovima. Ternarnim operatorom u bloku unutar „style“ atributa provjerava istinitost izraza jeVidljiv te će se sukladno tome na tekst primijeniti odgovarajuća boja. Tekst će zbog izraza nad stilskim uputama elementa uvijek biti 10 piksela veći od veličine definirane unutar Vue komponente što znači da će u primjeru tekst biti veličine 24 piksela.

```
<div id="app"> <img :src="slikaSrc"
:alt="poruka.toUpperCase()">
<div :style="{ color: jeVidljiv ? 'green' : 'red', fontSize:
(velicinaFonta + 10) + 'px' }">{{ poruka }}</div> </div>
<script> new Vue({ el: '#app',
 data: { slikaSrc: 'https://vuejs.org/images/logo.png',
poruka: 'Vue.js', jeVidljiv: true, velicinaFonta: 14} });
</script>
```

4. Komponente

Vue je komponentno bazirani okvir što znači da su Vue aplikacije realizirane kombiniranjem komponenata koje se mogu promatrati kao samostalni i višestruko iskoristivi konstrukcijski blokovi aplikacije. Razumijevanje komponente iznimno je važno jer su postale standard prisutan i u ostalim okvirima za razvoj klijentskih web aplikacija kao što su Angular, React, Spring Boot i sl. Prema [13] komponente se definiraju kao neovisni moduli koji apstrahiraju HTML kojim je definirana struktura DOM-a, CSS stilske upute kojima je definirana prezentacija DOM-a i JavaScript ili TypeScript funkcije kojima je definirano ponašanje komponente. Jednom razvijena komponenta može se višestruko koristiti u različitim dijelovima aplikacije što rezultira jednostavnijim razvojem i održavanjem.

Svaka komponenta sastoji se od tri osnovna elementa – predložak (oznaka <template>) koji se sastoji od HTML-a i JavaSript izraza koji definiraju strukturu komponente koju će preglednik renderirati i prikazati u DOM-u, skripte (oznaka <script>) koja podržava opcijske objekte komponente (eng. *component option object*) koji su zapravo JavaScript objekti koji se koriste za upravljanje podacima, logikom i životnim ciklusom komponente te stila (oznaka <style>) koji podržava stilske upute za prilagodbu prezentacije predloška komponente. Struktura komponente može se promatrati kao arhitektura tradicionalnih klijentskih aplikacija – predložak (<template>) se može promatrati kao HTML datoteka kojom je definirana struktura DOM-a, skripta (<script>) može se promatrati vanjska JavaScript datoteka kojom je definirano ponašanje aplikacije te stil (<style>) može se promatrati kao vanjska CSS datoteka kojoj je definiran izgled DOM-a. Komponenta se ne mora nužno sastojati od sva tri elementa ali prema [12] minimalna komponenta mora podržavati barem predložak kako bi se mogla primijeniti i polučiti korisnošću jer će se definirana struktura primijeniti na DOM i nadopuniti korisničko sučelje.

4.1. Vrste Vue aplikativnih programskih sučelja

Već je ranije spomenuto da Vue podržava dvije vrste API-ja kojima je uvjetovana sintaksa i ponašanje komponente: stariji opcijski i noviji kompozicijski. Ako se koristi opcijski API tada su logika i ponašanje komponente organizirani u opcijske objekte kao što su data() , methods, computed i slični unutar <script> elementa komponente. Ako se koristi kompozicijski API koji je predstavljen u Vue3 verziji tada se logika komponente ne razdvaja u različite opcijske objekte već je podržana unutar setup() funkcije koja podržava reaktivni API za definiranje logike i upravljanje ponašanjem komponente. [12]

U nastavku slijede dva primjera od kojih prvi demonstrira definiranje komponente korištenjem opcijskog API-ja, a drugi kompozicijskog API-ja. Oba su primjera funkcionalno identični te je razlika samo u sintaksi koja je uvjetovana korištenim API-jem.

```
<template> <div> <h1>{{naslov}}</h1> <p>{{poruka}}</p>
<button @click="pozdrav"> {{ gumbPoruka[brojac % 2] }}
</button> <p>{{ spavanje }}</p> </div> </template>
<script> export default { data() {
return { naslov: "Komponenta primjer",
poruka: "Definiranje komponente opcijski API", brojac: 0,
gumbPoruka: ["Dobro Jutro!", "Laku Noć!"], }; },
methods: { pozdrav() { this.brojac++; } },
computed: {spavanje(){ return this.brojac%2
? "Stisni za spavanje" : "Stisni za buđenje" } },
created() { console.log("Komponenta kreirana"); } };
</script> <style scoped> h1 { color: #42b883; }
p { color: #35495e; font-size: 20px; } </style>
```

Sadržaj unutar <template> elementa definira strukturu DOM-a koji će se u ovom slučaju sastojati od jednog bloka, paragrafa <p> koji ispisuje statičnu poruku, gumba <button> koji sluša događaj klika mišem i paragrafa <p> na koji se veže jedno računalno svojstvo (eng. *computed property* – u nastavku computed). Unutar <script> elementa koriste se opcijski objekti od kojih svaki definira specifičnu logiku upravljanja podacima. Unutar data() objekta definirani su objekti koji sadrže podatke te se vežu na HTML elemente u predlošku. Unutar methods objekta definirane su funkcije koje se pozivaju na određeni događaj kao što su klik miša, unos teksta i sl. Unutar computed objekta definiraju se funkcije čije stanje ovisi o objektima definiranim unutar data() objekta. Razlika između methods i computed je što se unutar methods objekta definiraju funkcije koje mijenjaju stanje svojstava komponente ovisno o događajima, dok se unutar computed bloka definiraju funkcije čije se stanje mijenja promjenom ostalih svojstva komponente. Tako se u prethodnom primjeru unutar methods objekta definira funkcija koja se poziva na događaj klika miša na gumb, dok se unutar computed objekta definira funkcija čija se vrijednost mijenja ovisno o promjeni vrijednosti na objektu „brojac“. Objekt created() je kuka životnog ciklusa komponente (eng. *lifecycle hook*) koja se poziva nakon što je komponenta kreirana ali prije nego što je renderirana i prikazana u DOM-u. [12]

Ovisno o literaturi za opcijske objekte se mogu koristiti različiti termini pa se tako u [4] imenuju kao svojstva komponente (eng. *Property*). Unutar <style> elementa definirane su stilske upute za <h1> naslove i <p> paragrafe.

```
<template><div><h1>{{ naslov }}</h1>
<p>{{ poruka }}</p><button @click="pozdrav">{{
 gumbPoruka[brojac % 2] }}</button>
<p>{{ spavanje }}</p></div></template><script setup>
import { ref, computed, onMounted } from 'vue';
const naslov = ref("Komponenta primjer");
const poruka = ref("Definiranje komponente kondicijski API");
const brojac = ref(0);
const gumbPoruka = ref(["Dobro jutro", "Laku Noć!"]);
const pozdrav = () => {brojac.value++;}; const spavanje =
 computed(() => {return brojac.value % 2 ? "Stisni za spavanje"
: "Stisni za buđenje";});onMounted(() =>
{console.log("Komponenta kreirana");});</script>
<style scoped>h1 {color: #42b883;}
p {color: #35495e;font-size: 20px;}</style>
```

Prethodni primjer demonstrira kreiranje komponente korištenjem kompozicijskog API-ja. Blok koda unutar <template> elementa identičan je kao i kod opcijskog API-ja. Glavna razlika je unutar <script setup> elementa unutar kojeg se primjenom kompozicijskog API-ja ne koriste jedinstveni opcijski objekti kao što je slučaj kod opcijskog API-ja. Jedini opcijski objekt koji se može koristiti u oba API-ja je component koji registrira komponente djecu. [12]

Utjecaj API-ja najviše se odražava na proces razvoja aplikacije, a semantika će biti jednaka neovisno o izabranom API-ju. Prema [4] kompozicijski API pruža veću fleksibilnost u razvoju te je pogodan za veće projekte koji će zahtijevati stalno održavanje i nadogradnju. Isto tako kompozicijski API osigurava veću modularnost od opcijskog API-ja što znatno ubrzava razvoj aplikacije. Prednost opcijskog API-ja je da je lakše razumljiv te je krivulja učenja blaža. Oba API-ja podržavaju TypeScript, no opcijski API zahtjeva dodatne anotacije čime se povećava obujam koda, a sintaksa postaje teže čitljiva. Kompozicijski API objavljen je u Vue 3 verziji kada je postalo jasno da TypeScript postaje standard te je stoga razvijena bolja podrška za njega. Vue zajednica posvećena je održavanju oba API-ja bez obzira na široku prihvaćenost novijeg kondicionalnog API-ja. Ovisno o zahtjevima aplikacije odabire se odgovarajući API.

Bez obzira na to koristi li se opcijski ili kompozicijski API, Vue komponenta se može implementirati kao JavaScript objekt unutar JavaScript datoteke ili u posebnoj datoteci s ekstenzijom .vue što je učestaliji pristup poznatiji pod nazivom jedno datotečna komponenta (*eng. Single-File Component* – SFC).

Vue podržava širok izbor opcijskih objekata od kojih svaki ima specifičnu namjenu upravljanja ponašanja komponentom. Važno je za napomenuti da se opcijski objekti koriste isključivo u opcijskom API-ju, a kompozicijski API za neke od njih podržava alternativne funkcije iste namjene. U prethodnom su poglavlju kroz primjere već objašnjeni najkorišteniji opcijski objekti i njihove alternative u kompozicijskom API-ju. U nastavku slijedi popis ostalih opcijskih objekata u opcijskom API-ju i njegovih alternativnih funkcija u kompozicijskom API-ju zajedno s objašnjenjima: [4]

- props u opcijskom, defineProps() u kompozicijskom – koristi se za definiranje objekta koje komponenta djeca nasljeđuju od roditelja. Tako komponenta roditelj može svakoj komponenti djetetu proslijediti podatke.
- watch u opcijskom, watch() ili watchEffect() u kompozicijskom – koristi se za definiranje objekta koji pozivaju funkciju povratnog poziva (eng. *Callback function*) kada se detektira promjena stanja komponente.
- filter – filteri su se u Vue2 koristili za formatiranje teksta te nisu podržani u Vue3 verziji bez obzira na to koristi li se opcijski ili kompozicijski API. Preporuka je tekst formatirati kroz vlastite ili bibliotečne funkcije.
- mixin u opcijskom, kompozicijski API ne podržava ugrađene alternative – koristi se u opcijskom API-ju za uvoz i spajanje dijelova koda izvan komponente u komponentu, omogućujući različitim komponentama dijeljenje jednakih funkcionalnosti. Iako su podržane u Vue3, [10] ne preporučuje njihovo korištenje zbog Vuex biblioteke koja je bolja i sigurnija alternativa.
- extends u opcijskom, kompozicijski API ne podržava ugrađene alternative – koristi se u opcijskom API-ju za nasljeđivanje kompletne roditeljske komponente. Razlika između props i extends je što se props objektom nasljeđuju samo podaci, a extends kompletno ponašanje komponente. Prema [10] nasljeđivanje se treba izbjegavati u kompozicijskom API-ju.
- render() u opcijskom, kompozicijski API ne podržava ugrađenu alternativu već se preporučuje korištenje vraćanje anonimne funkcije unutar setup() funkcije. Oba načina koriste se za programsko definiranje strukture komponente.

4.2. Životni ciklus komponente

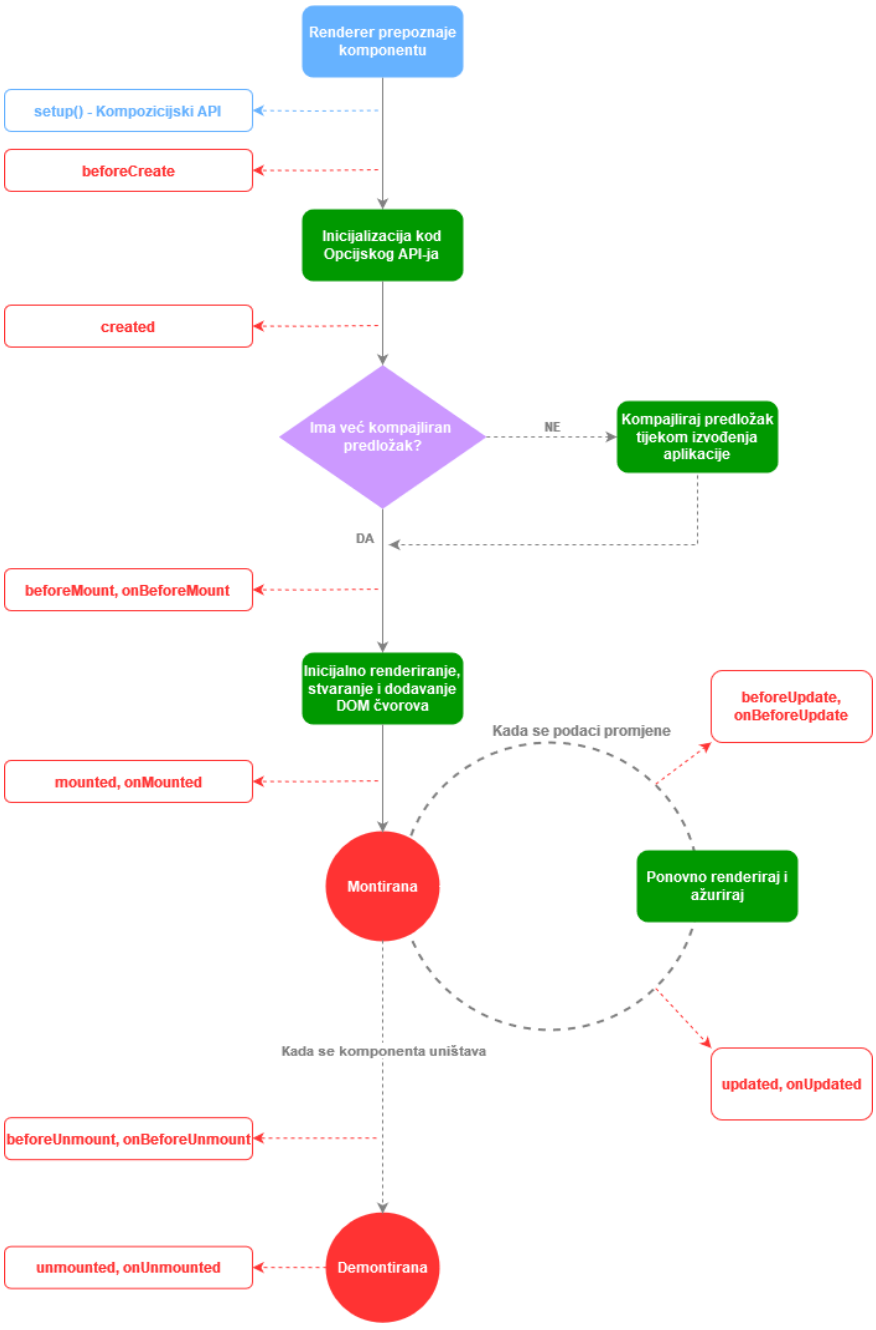
Prema [12] životni ciklus komponente se odnosi na stadije kroz koje komponenta prolazi tijekom svog životnog vijeka, od trenutka inicijalizacije do njenog uništavanja. Životni ciklus komponente karakteriziraju različiti stadiji od kojih svaki ima specifičnu ugrađenu metodu kojom se definiraju akcije koje će se poduzeti kada komponenta doživi promatrani stadij. Te metode poznatije su i kao kuke životnog ciklusa komponente. Kuke životnog ciklusa komponente razlikuju se ovisno o API-ju koji se koristi. Korištenjem opcijskog API-ja kuke se definiraju kao opcijski objekti dok se u kompozicijskom API-ju moraju eksplicitno uvesti te se nakon toga definiraju unutar setup() funkcije. U nastavku slijedi popis svih kuka životnog ciklusa i opis pripadajućeg stadija životnog ciklusa komponente za oba API-ja [4] :

- beforeCreate (opcijski API) – poziva se prilikom inicijalizacije komponente, kompozicijski API ne podržava kuku za ovaj stadij jer se setup() funkcija automatski izvršava prije bilo koje kuke životnog ciklusa.
- created (opcijski API) – poziva se nakon što su komponenta, reaktivni podaci komponente, metode, computed svojstva i promatrači (eng. *watchers*) inicijalizirani. U kompozicijskom API-ju ne postoji slična kuka jer istu funkcionalnost implementira setup() funkcija.
- beforeMount(opcijski API), onBeforeMount (kompozicijski API) – sva reaktivna svojstva komponente su inicijalizirana i počinje proces montiranja komponente za DOM. Vue kreira DOM stablo komponente i konvertira virtualni DOM u pravi DOM unutar preglednika.
- mounted (opcijski API), onMounted (kompozicijski API) – poziva se nakon što je komponenta montirana odnosno nakon što je DOM stablo komponente prikazano unutar DOM-a.
- beforeUpdate (opcijski API), onBeforeUpdate (kompozicijski API) – poziva se prije nego što reaktivnost ažurira podatke u DOM-u zbog detektiranih promjena stanja komponente.
- updated (opcijski API), onUpdated (kompozicijski API) – poziva se nakon što je DOM stablo komponente ažurirano.
- beforeUnmount (opcijski API), onBeforeUnmount (kompozicijski API) – poziva se prije nego što je instanca komponente demontirana. U ovom stadiju instanca komponente je i dalje funkcionalna te se može manipulirati njenim ponašanjem.
- unmounted (opcijski API), onUnmounted (kompozicijski API) – poziva se nakon što je komponenta demontirana odnosno nakon što su sva djeca koja nasljeđuju

komponentu demontirana i nakon što su očišćena sva reaktivna svojstva komponente.

- errorCaptured – poziva se za hvatanje i upravljanje greškama koje su propagirane kroz komponente djecu.

Osim sintaktičkih razlika u nazivima kuka životnog ciklusa, vidljivo je da postoje i određene razlike u njihovom ponašanju s obzirom da kompozicijski API inicijalizaciju obavlja kroz setup() funkciju. Slika 2 vizualizira životni ciklus komponente i pripadajuće kuka životnog ciklusa.



Slika 2. Životni ciklus komponente [14]

4.3. Ugrađene komponente

Vue podržava predefinirane ugrađene komponente (eng. *built-in components*) koje pojednostavljuju i ubrzavaju razvoj određenih značajki. Ugrađene komponente mogu se direktno koristiti u projektu bez potrebe za eksplicitnim uvozom ili registracijom. Ukoliko se koriste unutar render() objekta tada je potrebno eksplicitno uvesti ugrađenu komponentu. Ugrađene komponente dizajnirane su tako da im se ne moraju definirati dodatna svojstva, no po potrebi se mogu modificirati i prilagoditi vlastitim zahtjevima. Vue trenutno podržava četiri u potpunosti funkcionalne ugrađene komponente: <Transition>, <TransitionGroup>, <KeepAlive> i <Teleport> te komponentu <Suspense> koja je trenutno u eksperimentalnoj fazi. [10]

Ugrađene komponente <Transition> i <TransitionGroup> koriste se za implementaciju animacije i tranzicije DOM elemenata kod promjene stanja komponente. Razlika je da se <Transition> komponenta koristi za tranziciju jednog DOM elementa prilikom njegova pojavljivanja i nestajanja dok se <TransitionGroup> koristi za tranziciju grupe elemenata prilikom njihova umetanja, izbacivanja ili pomicanja kroz listu. U nastavku slijedi primjer koji demonstrira korištenje <Transition> ugrađene komponente. [4]

```
<template><div><button @click="promjeni">Testiraj</button>
<transition name="fade"><p v-if="prikazi">Transition ugrađena
komponenta</p></transition></div></template><script>export
default {data() {return { prikazi: true};},methods: {promjeni()
{this.prikazi = !this.prikazi; }}};</script><style>
.fade-enter-active, .fade-leave-active {transition: opacity 1s;}
.fade-enter-from, .fade-leave-to {opacity: 0.09;}</style>
```

Unutar <template> elementa koristi se <Transition> ugrađena komponenta zbog čega obični tekst paragrafa ugniježđenog unutar komponente poprima laganu tranziciju prilikom pojavljivanja i nestajanja na sučelju. Pojavljivanje i nestajanje može biti uvjetovano kondicionalnim renderiranjem korištenjem v-if direktive, kondicionalnim prikazom korištenjem v-show direktive, dinamičkom promjenom komponente korištenjem <component> elementa ili promjenom stanja atributa ključa „:key“ komponente. Na kraju se unutar <style> elementa korištenjem pred definiranih klasnih selektora definiraju stilske upute tranzicije. Sve ugrađene komponente smještaju se unutar <template> elementa kao u prethodnom primjeru.

Ugrađena komponenta <TransitionGroup> ponaša se jednako kao i <Transition> samo što djeluje nad listom elemenata te je zbog toga obvezno korištenje v-for direktive za iteraciju kroz listu i atribut ključa „:key“ koji identificira svaki element liste. [4]

Vue podržava mogućnost da se na istom čvoru DOM-a dinamički izmjenjuju različite komponente (eng. *Dynamic Components*). To je omogućeno korištenjem <component> elementa i njenog specijalnog atributa „:is“ kojemu se kao vrijednost može proslijediti registrirana komponenta, uvezeni objekt komponente ili obični HTML element. Ovisno o proslijeđenoj vrijednosti DOM će se dinamički mijenjati na istom čvoru. Problem je što se promijenjeno stanje komponente ne sprema kod dinamičke promjene komponente jer se kod svake promjene stvara nova instanca komponente koja je proslijeđena kao vrijednost „is:“ atributu, a trenutna komponenta se uništava. Ako je korisnik promijenio stanje inicijalno učitane komponente i zatim dinamički zamijenio komponentu, promijenjeno stanje inicijalne komponente neće se spremiti, a kada se komponenta ponovno prikaže na sučelju poprimit će inicijalno definirano stanje. Ugrađena komponenta <KeepAlive> omogućuje spremanje promjene stanja komponente pa se kod dinamičke promjene pamti stanje komponenata koje se izmjenjuju. Ako korisnik svojom akcijom uzrokuje promjenu podataka oni će ostati spremljeni prilikom dinamičkih izmjena između komponenti. Ugrađena komponenta <KeepAlive> koristi se tako da se kao njen sadržaj ugnijezdi <component :is=“vrijednost“> element. [10]

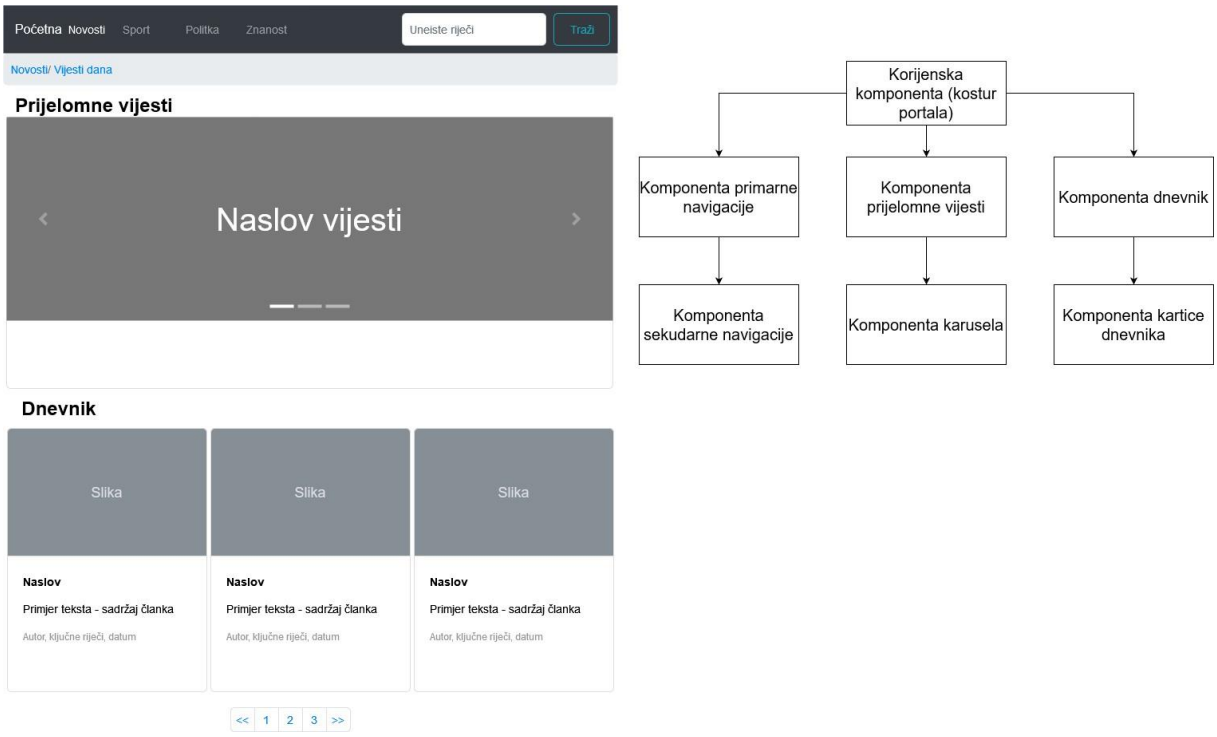
Željeni dizajn korisničkog sučelja može zahtijevati odvajanje dijela predloška komponente u posebni čvor DOM-a što može predstavljati problem jer je poželjno da se struktura komponente ne razdvaja u posebne komponente, a istovremeno je potrebno dijelove predloška renderirati na različitim mjestima unutar DOM-a. Ugrađena komponenta <Teleport> omogućuje izdvajanje dijela predloška u posebni čvor DOM-a koji nije dio DOM stabla komponente. Specijalni atribut „:to“ specificira ciljani DOM element na koji će se elementi ugniježđeni unutar <Teleport> komponente prikvačiti. Ciljani element mora biti renderiran prije <Teleport> komponente. [4] Važno je za napomenuti da [10] preporučuje da ciljani HTML element bude idealno izvan konteksta Vue aplikacije.

S obzirom da je ugrađena komponenta <Suspense> trenutno u fazi testiranja neće se detaljno objašnjavati. Ono što je važno spomenuti za <Suspense> je da samostalno upravlja asinkronim operacijama unutar komponente i sukladno njihovom izvršavanju ažurira DOM. Ako na primjer komponenta dohvaća veliku količinu podataka s vanjskog servisa ta operacija može dugo trajati te će se stoga izvršavati asinkrono kako se glavni program ne bi blokirao. Za to vrijeme komponenta će prikazati određenu poruku ili animaciju rotirajućeg kruga sve dok podaci ne budu dohvaćeni i spremni za prikaz. Kada su podaci dohvaćeni komponenta će

ukloniti poruku ili rotirajući krug te će vezati podatke na svoj predložak nakon čega će podaci biti prikazani unutar DOM-a. Iznimno je korisna kada duboko ugniježđeni elementi unutar predloška ovise o asinkronim funkcijama. [4]

4.4. Komponentno bazirana arhitektura

Za kraj poglavlja važno je objasniti komponento baziranu arhitekturu kojom je definirana arhitektura svake Vue aplikacije. Kako je već prethodno objašnjeno Vue aplikacija razvijena je korištenjem komponenata koje su građevinski blokovi aplikacije. To znači da je cijela aplikacija razvijena kao stablo međusobno povezanih komponenata čije se veze opisuju analogijom roditelj-dijete. Svaka komponenta može se promatrati kao komponenta roditelj i kao komponenta dijete. To znači da će svaka komponenta imati nadređenu komponentu roditelja te može, ali i ne mora, imati komponente djecu kojima je ona nadređena. U svakoj Vue aplikaciji postoji jedna korijenska komponenta (eng. *root component*) koja je roditelj svim ostalim komponentama i koja nema svog roditelja. Slika 3 vizualizira arhitekturu bazirane na komponentama.



Slika 3: Skica korisničkog sučelja i vizualizacija pripadajuće veze između komponenata [autorski rad]

Prema [13] komponentno bazirana arhitektura definira tok podataka i ograničava komunikaciju između komponenata. Komponenta je samo sadržavajući objekt i izolirana je od ostatka aplikacije što znači da njenim podacima i metodama ne mogu direktno pristupiti ostale komponente. Komponenta roditelj može proslijediti podatke komponentama djeci korištenjem props opcijskog objekta u opcijskom API-ju ili defineProps() funkcije u kompozicijskom API-ju. Na taj se način ostvaruje jednosmjerni tok podataka od roditelja prema djeci i osigurava se da komponente ne mogu direktno pristupati drugim komponentama i mijenjati njihovo stanje ali da mogu primati podatke od nadređenih komponenata. Komunikacija od komponente djeteta prema roditelju moguća je isključivo definiranjem prilagođenih događaja (eng. *custom events*) koje emitiraju komponente djeca, a komponente roditelji slušaju i poduzimaju odgovarajuće akcije je događaj emitiran.

5. Vue biblioteke

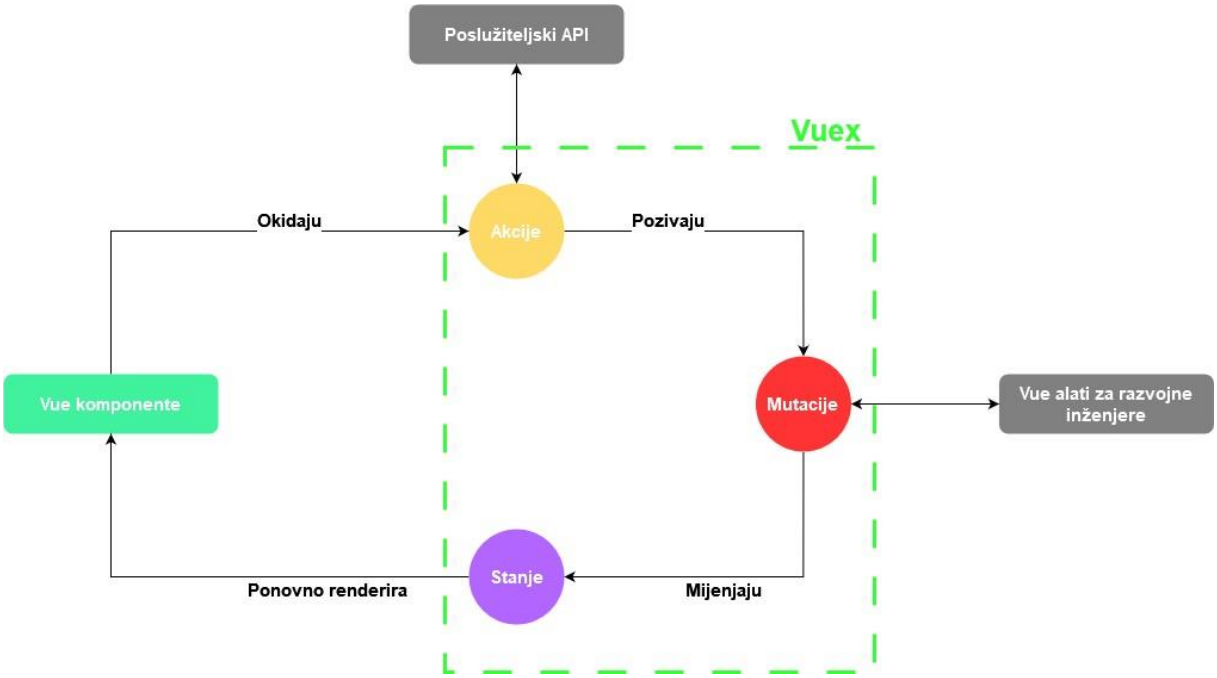
U narednim poglavljima predstavit će dvije najkorištenije Vue biblioteke – Vuex i Vue Router. Njihovo korištenje nije nužno za razvoj Vue aplikacije ali je česta praksa, naročito kod razvoja složenijih aplikacija za koje se predviđa dugoročno korištenje čime se povećavaju zahtjevi za održavanjem.

5.1. Vuex i Pinia

Pinia i Vuex biblioteke su za upravljanje stanjem komponenata u Vue.js aplikaciji. Pinia i Vuex podržavaju identičnu funkcionalnost samo što se nakon Vuex 5 verzije ime biblioteke promijenilo iz Vuex u Pinia. Razlike između Vuexa i Pinie očituju se dakle u tome da je Pinia novija verzija Vuexa te stoga podržava šire mogućnosti kao što su bolja podrška za TypeScript i kompozicijski API.

Prema [13] Vuex je biblioteka koja omogućuje pohranu i dohvat svih podataka Vue aplikacije korištenjem jedinstvene globalne strukture podataka koja se naziva stanje (eng. *state*), a koja je spremljena u spremište (eng. *store*). Stanje se nikada ne može mijenjati direktno izvan spremišta već pogledi i komponente odašilju akcije (eng. *actions*) odnosno funkcije koje opisuju što se dogodilo i pozivaju mutacije (eng. *mutations*) odnosno sinkrone funkcije koje su zadužene za promjenu stanja podataka unutar spremišta. Jednom kada je stanje promijenjeno, sve komponente i pogledi ovisne o tom stanju se ponovno renderiraju. Problem koji Vuex rješava jesu situacije u kojima različite komponente u aplikaciji ovise o istom stanju podataka, a akcije koje odašilju različite komponente mijenjaju isto stanje. Takve situacije često iziskuju repetitivno pisanje robusnog koda koji je teško čitljiv i održiv te usporava performanse aplikacije. Vuex nudi mogućnosti definiranja i upravljanja stanjem podataka aplikacije u jedinstvenom globalnom spremištu. Na taj se način stanje podataka odvaja od komponente za što se često koristi termin razdvajanje interesa (eng. *separation of concerns*) jer je logika upravljanja stanjem podataka odvojena od logike prezentacije tih podataka korisniku. To rezultira jasnijom arhitekturom i jednostavnijim održavanjem jer je definirana granica između upravljanja stanjem podataka i njihove prezentacije. Najbolji primjer korištenja Vuex-a spremanje je korisnika koji se prijavljuje u aplikaciju. Korisnički podaci često su potrebni kroz cijelu aplikaciju što uključuje veliki broj komponenata. Nakon prijave korisnika Vuex ga sprema u globalno spremište čime korisnički podaci postaju dostupni svakoj komponenti. Na taj način izbjegava se višestruko dohvaćanje korisničkih podataka sa poslužitelja čime se smanjuje količina koda i optimiziraju se performanse aplikacije. Svaka komponenta bez obzira na njezinu ugniježđenost u aplikaciji može odašiljati akcije koje će promijeniti stanje u

spremištu. Vuex i Pinia kompatibilni sa Vue alatima za razvojne inženjere (eng. *Vue DevTools*) koji nude napredne mogućnosti pregledavanja stanja aplikacije tijekom njezina izvođenja. Slika 4 vizualizira tok akcija korištenjem Vuex-a ili Pinie.



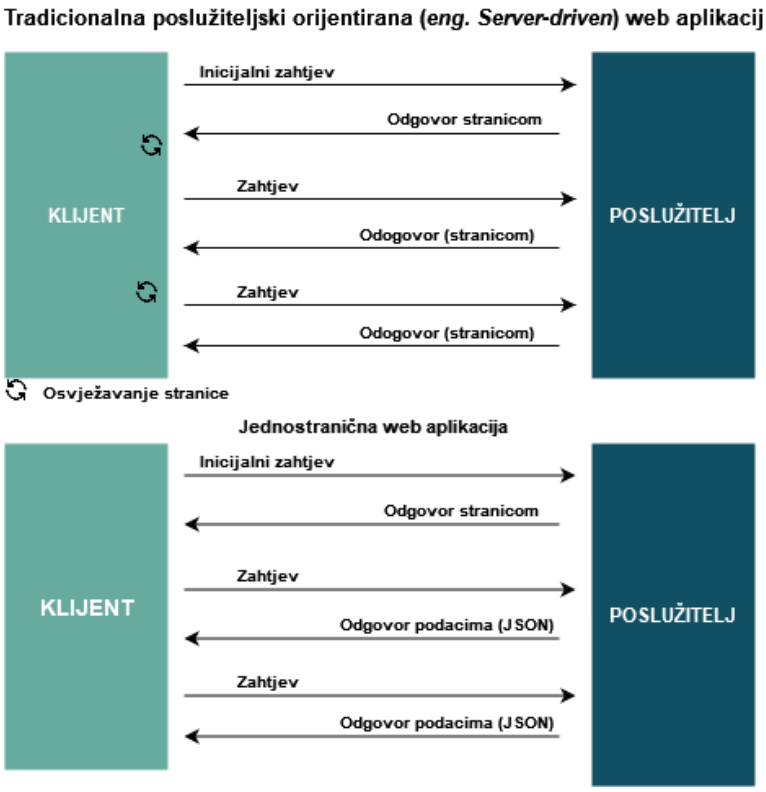
Slika 4: Tok akcija korištenjem Vuex-a ili Pinie [4, str. 334]

5.2. Vue router

Vou Router službena je Vue biblioteka namijenjena pojednostavljenju razvoja jednostraničnih web aplikacija (eng. *Single-page application* – SPA). Prema [5] jednostranična web aplikacija predstavlja arhitekturu kojom poslužitelj na inicijalni zahtjev klijenta za određenim resursom odgovara jednim dokumentom. Klijent za svaki novi resurs šalje novi zahtjev, a poslužitelj ne odgovara kompletnom web stranicom kao što je slučaj kod tradicionalnih web aplikacija, već samo vraća podatke. Klijent je zadužen za prezentaciju zaprimljenih podataka unutar stranice. Tako se smanjuje promet između klijenta i poslužitelja jer klijent od poslužitelja ne prima kompletne stranice s funkcionalnostima već samo podatke što rezultira optimiziranijim performansama aplikacije i poboljšanim korisničkim iskustvom. Jednostranične web aplikacije realiziraju se korištenjem ugrađene JavaScript funkcije fetch() ili biblioteka poput Axiosa koji omogućuju slanje zahtjeva poslužitelju te primanje odgovora od poslužitelja i rukovanje zaprimljenim podacima bez potrebe da preglednik osvježi trenutnu stranicu. Tako se korisniku prikazuje samo jedna web stranica čiji se sadržaj mijenja ovisno o akcijama korisnika i dohvaćenim podacima.

Razvoj jednostraničnih aplikacija olakšava se korištenjem okvira poput Angulara, Reacta i Vue-a zajedno s pripadajućim bibliotekama tako da se na jednoj stranici izmjenjuju različiti pogledi i komponente. Za olakšanu implementaciju promjene pogleda i komponenti na jednoj stranici koristi se Vue Router biblioteka.

Slika 5 vizualizira razliku u komunikaciji između klijenta i poslužitelja kod tradicionalnih poslužiteljski orijentiranih (*eng. Server-driven*) web aplikacija i modernih jednostraničnih aplikacija.



Slika 5: Razlika između tradicionalnih i jednostraničnih web aplikacija [13, str. 263]

Vue Router podržava opcijske objekte, funkcije i ugrađene komponente kojima pojednostavljuje definiranje jedinstvenih pronalazača resursa (eng. *Uniform Resource Locator* – u nastavku akronim URL) i pripadajućih resursa. Sukladno konfiguraciji URL-a korisniku će se za putanju unesenu u pretraživač preglednika prikazati odgovarajući resurs odnosno komponenta ili pogled. U nastavku slijede dva isječka koda koja demonstriraju inicijalizaciju Vue Router instance i konfiguriranje putanja za poglede koji će se prikazati unutar iste stranice.

```
<template><p>Trenutna putanja: {{ $route.fullPath }}</p>

<nav><RouterLink to="/">Početna</RouterLink><RouterLink

to="/profil">Profil</RouterLink></nav><main><RouterView
/></main></template>
```

Dvije najčešće korištene komponente Vue Routera su RouterLink i RouterView. Komponenta RouterLink slična je <a> HTML elementu hiperveze. Umjesto „href“ atributa RouterLink sadrži atribut „to“ kojemu se kao vrijednost prosljeđuje relativna putanja do resursa koji će se prikazati unutar RouterView elementa. Dakle onda kada korisnik klikne na element RouterLink na mjestu RouterViewa će se prikazati odgovarajući pogled ili komponenta.

```
import { createRouter, createWebHistory } from 'vue-router'
import IndexPogled from './IndexPogled.vue'
import ProfilPogled from './ProfilPogled.vue'
const routes = [{ path: '/', component: IndexPogled },{ path:
 '/profil', component: ProfilPogled },] const router =
createRouter({history:createWebHistory(),routes,})
export default router
```

Instanca Vue Routera kreira se pozivanjem metode createRouter() kojoj se kao prvi argument obvezno prosljeđuje postavka povijesti, a kao drugi obvezni argument objekt routes koji je zapravo polje koje sadrži objekte kojima su konfigurirane putanje te pripadajuće komponente ili pogledi za prikaz.

6. Vue CLI

Vue sučelje naredbenog retka (eng. *Command Line Interface* – u nastavku akronim CLI) alat je koji olakšava kreiranje i upravljanje projektom razvoja Vue aplikacije. Prema [4] osnovna ideja Vue CLI-a je pojednostavljenje konfiguracije postavki projekta omogućujući razvojnim inženjerima da više vremena i napora posvete razvoju funkcionalnosti aplikacije, a manje upravljanju konfiguracijama projekta ili dodavanja ekstenzija u projekt. Vue CLI instalira se kao globalni NPM paket nakon čega se u naredbenom retku mogu izvršavati Vue naredbe tako da svaka počinje izrazom „vue“. Tako se za kreiranje i inicijalno konfiguriranje projekta koristi naredba „vue create“ unutar direktorija u kojem se želi kreirati projekt. Naredbom „vue add router“ jednostavno i brzo se uključuje spomenuta Vue Router biblioteka u projekt, a naredbom „vue add vuex“ spomenuti Vuex.

Vue CLI omogućuje jednostavnu konfiguraciju projekta specifičnim potrebama aplikacije, preferencijama razvojnog inženjera ili razvojnog tima unutar „vue.config.js“ datoteke čime se eliminira potreba za ručnim konfiguriranjem postavki (eng. *Ejecting*) u konfiguracijskim datotekama svakog alata ili biblioteke korištene u projektu. Tako se smanjuju pogreške koje nastaju prilikom konfiguracije projekta i dodavanja novih biblioteka te se isto tako olakšava održavanje projekta s obzirom da su sve konfiguracijske postavke spremljene u jednoj datoteci koja apstrahira postavke ostalih alata i ekstenzija. Od Vue3 verzije Vue CLI proširen je grafičkim korisničkim sučeljem pa su kompletni CLI alat i „vue“ naredbe mogu zamijeniti vizualnim okruženjem koje razvojnim inženjerima dodatno olakšava kontrolu i upravljanje projektom. [4]

7. Razvoj novinskog portala

Praktični dio rada obuhvaća razvoj kompletne web aplikacije uključujući klijentsku aplikaciju razvijenu korištenjem Vue okvira i poslužiteljsku aplikaciju razvijenu u Node.js (u nastavku Node) platformi i Express.js (u nastavku Express) okvirom. Aplikacija implementira funkcionalnosti popularnih novinskih portala kao što su LiderMedia.hr, TheGuardian.com, DeutscheWelle.com i sl. Aplikacija tako omogućuje kreiranje i čitanje novinskih članaka, registraciju i prijavu u aplikaciju te sudjelovanje u raspravama za specifični članak. Naredna poglavlja detaljno proučavaju tehnologije i alate korištene u razvoju, funkcionalne zahtjeve aplikacije, logički model podataka, korištene biblioteke i okvire, izgled korisničkog sučelja, dijelove izvornog koda osnovnih funkcionalnosti te scenarije korištenja aplikacije.

7.1. Korištene tehnologije i alati

Web aplikacija koja podrazumijeva praktični dio rada razvijena je u Linux Mint 21.1 operacijskom sustavu korištenjem uređivača teksta otvorenog koda VS Codium. Poslužiteljska strana aplikacije razvijena je korištenjem Node platforme, a dodatni moduli potrebni za razvoj koji nisu direktno instalirani s Node platformom preuzeti su alatom za upravljanje Node paketima (eng. *Node Package Manager,* u nastavku akronim NPM). Za konfiguraciju projekta i dodavanje modula na klijentskoj aplikaciji korišten je Vue CLI. Za upravljanje izvornim kodom (eng. *Version control*) koristi se alat Git, a za sigurnosno kopiranje koda i olakšanu suradnju između mentora i studenta koristi se platforma GitHub te je izvorni kod kompletnog projekta dostupan na repozitoriju kojemu se može pristupiti putem poveznice https://github.com/MatkoKekez/zavrsni_rad_prakticni_dio.git. Baza podataka implementirana je korištenjem MySQL Workbench alata koji pruža grafičko sučelje za dizajn baze, a podaci aplikacije pohranjeni su u SQLite datoteku. Za savladavanje osnovnih koncepata Vue okvira korištena je službena dokumentacija okvira [10], besplatna knjiga „Fullstack Vue“ autora Hassana Djirdeha [13], kupljena knjige „Vue.js 3 By Example“ autora Johna Au-Yeaunga [12], kupljena knjiga „Vue.js 3 Cookbook“ autora Hietora Ramona Ribeira [4] te online portal „Vue Mastery“ na kojemu se objavljuju lekcije za učenje Vue.js okvira na poveznici https://www.vuemastery.com/. Dodatno su za savladavanje okvira proučene dvije Vue aplikacije otvorenog koda – službena stranica Lutris alata čiji je izvorni kod dostupan na poveznici https://github.com/lutris/website te web alat za upravljanje izvornim kodom i Git repozitorijima GitLab čiji je izvorni kod dostupan na poveznici https://gitlab.com/gitlab-org/gitlab.git. Proučavanje strukture navedenih projekata otvorenog koda pruža uvid u dobre prakse kodiranja i olakšava razumijevanje arhitekture Vue aplikacija.

7.1.1. Biblioteke i okviri korišteni na poslužiteljskoj aplikaciji

Poslužiteljska aplikacija razvijena je u Node.js platformi koju [5] opisuje kao program koji omogućuje izvršavanje JavaScript koda izvan preglednika odnosno na poslužitelju, omogućujući tako razvoj jednodretvenih poslužiteljskih aplikacija. Za razvoj poslužiteljske aplikacije korišteno je ukupno 7 modula od kojih je najvažniji okvir Express koji prema autoru [5] znatno pojednostavljuje rukovanje HTTP zahtjevima i odgovorima.

Autor u [5] preporučuje veliki broj modula i paketa koji su prikladni za implementaciju poslužitelja u Node.js, a u radu su korišteni:

- bcrypt – biblioteka za rukovanje osjetljivim podacima koja se u razvijenoj aplikaciji koristi za sigurno pohranjivanje korisničkih lozinki. Biblioteka lozinke sažima (*eng. Hash*) algoritmom koji je baziran na Blowfish algoritmu sažimanja te automatski dodaje sol (*eng. Salt*) kako bi se osigurao integritet podataka. Biblioteka se koristi jer pojednostavljuje sažimanje lozinki prilikom njihova spremanja u bazu i zato što apstrahira proces validacije lozinke prilikom prijave korisnika. Funkcionalnosti registracije i prijave se tako razvijaju znatno brže uz visoku razinu sigurnosti.
- cors – paket kojim se omogućuje dijeljenje resursa između različitih izvora (*eng. Cross-Origin Resource Sharing* – u nastavku akronim CORS). Zbog sigurnosnih zahtjeva moderni preglednici implementiraju politiku istog izvora (*eng. Same-Origin Policy*) kojom se klijentskoj aplikaciji onemogućuje slanje zahtjeva prema poslužitelju ako su im izvori različiti (različita domena i otvor). Ovaj paket koristi se za jednostavnu konfiguraciju CORS-a omogućujući tako komunikaciju između klijenta i poslužitelja različitih izvora.
- express-session – paket za konfiguraciju i upravljanje sesijama za aplikacije implementirane Express okvirom. U aplikaciji se koristi za kreiranje sesije za prijavljenog korisnika. Sesija se automatski uništava ako klijent za kojeg je kreirana sesija ne pošalje novi zahtjev poslužitelju 30 minuta od posljednjeg zahtjeva.
- morgan – paket koji u konzolu ispisuje detalje o zahtjevima s klijenta i pripadajućim odgovorima poslužitelja.

7.1.2.Korištene biblioteke i okviri za klijentsku aplikaciju

Klijentska aplikacija razvijena je kao jednostranična aplikacija što je realizirano korištenjem već spomenute biblioteke Vue Router. U aplikaciji se također koristi i Vuex u čije se globalno spremište spremaju podaci o trenutno prijavljenom korisniku.

Sukladno preporukama autora [4] i [12] u projektu se koristi i Vuetify okvir koji sadrži pred definirane komponente kao HTML oznake koje se mogu koristiti u predlošku vlastitih

komponenata. Vuetify komponente koristi se za prikaz popisa novinskih članaka kao kartica na početnoj stranici aplikacije i specifičnoj stranici kategorije članka. To je realizirano ugnježđivanjem komponente <v-card> u komponentu <v-row>. Novinski članci se tako prikazuju kao kartice unutar kontejnera <v-row> koji sve kartice poravnava horizontalno i vertikalno, ostavljajući jednake razmake između svake kartice. Važno je za napomenuti da su obje komponente responzivne pa se automatski prilagođavaju različitim dimenzijama ekrana.

Isti autori preporučuju korištenje biblioteke Vuelidate koja apstrahira funkcije za validaciju obrazaca. U projektu se biblioteka koristi za provjeru popunjenosti obveznih polja, provjeru ispravnosti adrese e-pošte te provjeru broja znakova za unesena polja.

Za slanje zahtjeva prema poslužitelji i rukovanje odgovorom poslužitelja koristi se klijent Axios koji prema [4] pojednostavljuje sintaksu dohvaćanja podataka s poslužitelja i eliminira potrebu eksplicitnog rukovanja s JSON formatom što rezultira čišćim kodom.

7.2. Popis funkcionalnosti po ulogama

Poglavlja u nastavku opisuju funkcionalnosti aplikacije temeljem autorizacijskih uloga. U aplikaciji postoje četiri uloge – gost, korisnik, novinar i administrator. Funkcionalnosti svake uloge popisane su u istoimenom poglavlju.

7.2.1. Gost

Gosti su svi korisnici koji pristupaju aplikaciji putem weba i nemaju korisnički račun u aplikaciji. To je uloga s najmanjim pravima te ih može koristiti svaki korisnik weba. Gosti imaju pravo samo čitati podatke aplikacije.

Funkcionalnosti uloge gost:

- Pregledavanje popisa svih članaka na početnoj stranici.
- Pregledavanje popisa svih članaka na stranici specifične kategorije članka – financije, politika, sport, kultura, životni stil, tehnologija.
- Pregledavanje kompletnog sadržaja članka uključujući pripadajuće komentare i odgovore.
- Registracija u aplikaciju definiranjem opcionalnog imena i prezimena te obveznih vjerodajnica – jedinstvena adresa e-računa i lozinka duljine između 8 i 16 znakova.

7.2.2. Registrirani korisnik

Ako se korisnik uspješno registrira i prijavi u aplikaciju povećavaju mu se prava pa može stvarati dio sadržaja na portalu.

Funkcionalnosti registriranog korisnika:

- Pozitivno ili negativno reagiranje na sadržaj članka – registriranim korisnicima nudi se mogućnost ostavljanja povratne informacije na članak. Tako mogu ostaviti pozitivnu reakciju klikom na ikonu palca prema gore ili negativnu reakciju klikom na ikonu palca prema dolje. Temeljem svih reakcija izračunava se ocjena članka kao aritmetička sredina svih pozitivnih reakcija kojima je dodijeljena težinska vrijednost 10 i negativnih reakcija kojima je dodijeljena težinska vrijednost 1.
- Ostavljanje pregleda na članak – za svaki se članak broje pregledi koji odražavaju njegovu čitanost i popularnost. Broj pregleda se povećava ako registrirani korisnik provede minimalno 30 sekundi na stranici specifičnog članka. Svaki registrirani korisnik može imati samo jedan pregled za specifični članak.
- Komentiranje članaka – svaka stranica članka ima sekciju s komentarima gdje registrirani korisnik može ostavljati svoje dojmove i stajališta o pročitanom članku.
- Brisanje i ažuriranje komentara – korisnik može brisati i ažurirati svoje komentare.
- Pozitivno ili negativno reagiranje na vlastite i tuđe komentare – slično kao i za sadržaj članka, registrirani korisnici mogu ostavljati povratne informacije na vlastite i tuđe komentare klikom na gumb „Sviđa mi se“ ili „Ne sviđa mi se“.
- Odgovaranje na vlastite i tuđe komentare – s ciljem olakšane diskusije, registrirani korisnici mogu ostavljati odgovore na specifični komentar. Tako svaki komentar sadrži vlastitu sekciju odgovora.
- Brisanje i ažuriranje odgovora – svaki korisnik može brisati i ažurirati svoje odgovore na specifični komentar.
- Pozitivno ili negativno reagiranje na vlastite i tuđe odgovore – registrirani korisnici mogu ostavljati povratne informacije na vlastite i tuđe odgovore klikom na gumb „Sviđa mi se“ ili „Ne sviđa mi se“.
- Promjena korisničkih podataka – registrirani korisnici mogu promijeniti korisničke podatke koje su definirali prilikom registracije. Mogu se promijeniti svi podaci osim lozinke. Prilikom promjene podataka definirana su jednaka ograničenja kao i prilikom registracije – ime i prezime su opcionalni, a jedinstvena adresa e-računa je obvezna.

7.2.3. Novinari

Korisnici s ulogom novinara mogu izvršavati sve funkcionalnosti kao i registrirani korisnici te uz to mogu dodavati nove članke.

Funkcionalnosti novinara:

- Dodavanje novih članaka – novinar može napisati i objaviti novi članak na portalu. Prilikom pisanja novog članka novinar mora poštivati ograničenja kojima je definirano da svaki članak mora sadržavati naslov duljine između 10 i 90 znakova, sažetak duljine između 60 i 200 znakova i sadržaj duljine između 1500 i 5000 znakova. Novinar mora odrediti kategoriju članka sukladno njegovoj temi te može opcionalno dodati naslovnu sliku koja može biti ili resurs na internetu pri čemu se prosljeđuje pripadajući URL ili datoteka slike koja se pohranjuje u bazu i čija veličina ne smije prelaziti 200 kilobajta.
- Pregledavanje članaka s API servisa – novinar može pregledavati članke s vanjskog servisa News API o kojemu se više informacija može pronaći na stranici službene dokumentacije: https://newsapi.org/docs. Novinar u tražilicu unosi ključne riječi pretraživanja nakon čega se sa servisa dohvaćaju svi relevantni podaci uključujući naslov članka, sažetak, naslovnu fotografiju, dio sadržaja te poveznica na puni članak.
- Dodavanje članka s API servisa – uz svaki dohvaćeni članak s API servisa nudi se mogućnost dodavanja istog u bazu klikom na gumb „Dodaj članak“. Novinaru se tako otvara uobičajeni obrazac za pisanje članaka samo što je već popunjen podacima odabranog članka. Novinar te podatke može prilagoditi i promijeniti prije nego što ih objavi kao novi članak. Ograničenje servisa je da ne nudi puni sadržaj članka već samo prve dvije rečenice pa novinar mora ručno kopirati izvorni sadržaj članka ili napisati vlastiti.

7.2.4. Administrator

Korisnik s ulogom administratora može izvršavati sve funkcije kao i registrirani korisnik ali ne može izvršavati funkcionalnosti novinara pa tako ne može stvarati nove članke na portalu.

Funkcionalnosti administratora:

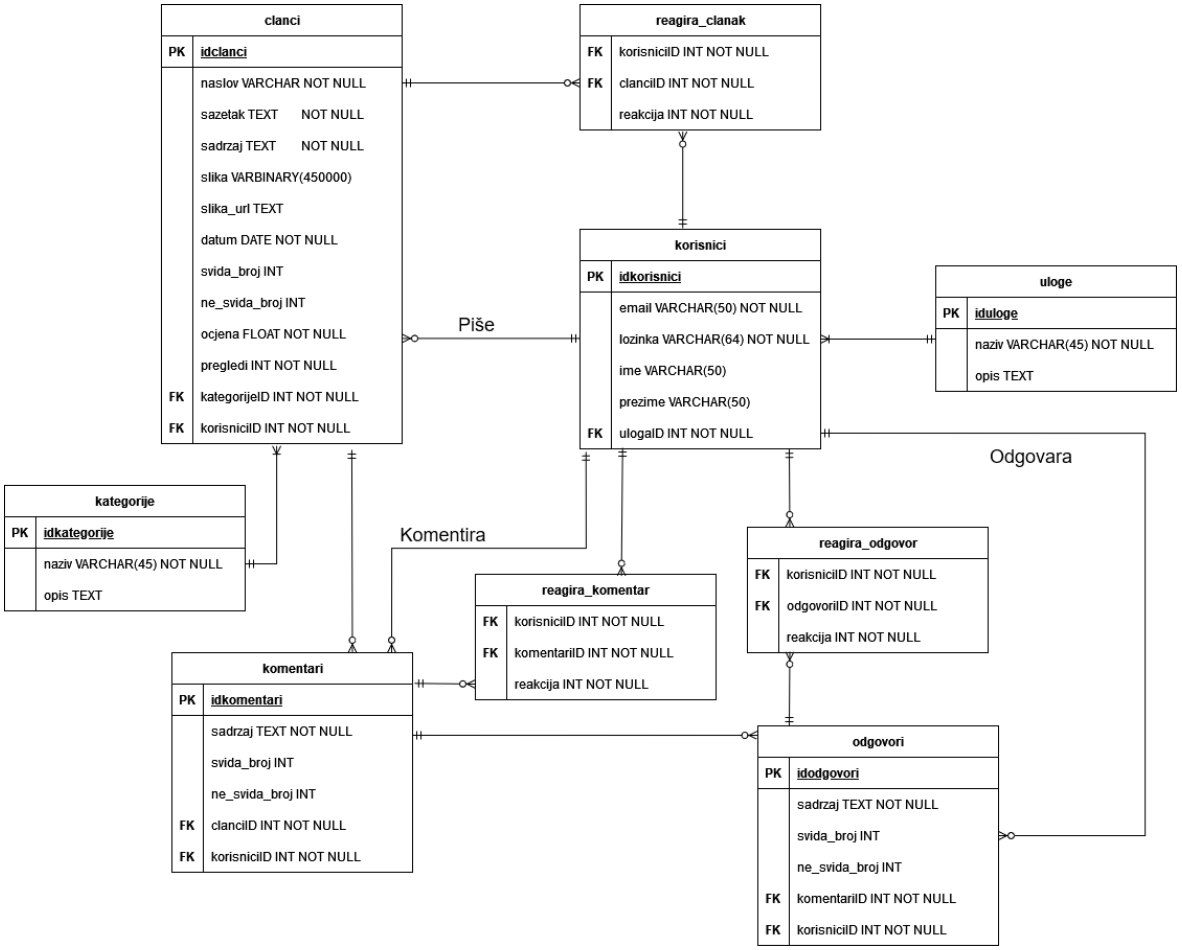
- Dodavanje novih novinara na portal – administrator na stranici novinari može postojećem korisniku dodijeliti prava novinara tako da unese pripadajuću adresu e-

računa. Korisnik pri tome mora imati definirano ime i prezime inače se administratoru odbija dodjela novinarskih prava korisniku s unesenom adresom e-računa.

- Pregled postojećih novinara – administrator na stranici novinara može pregledavati sve novinare portala.
- Brisanje novinara – u tablici s popisom novinara se uz svakog nudi mogućnost njegova brisanja. Pri tome se ne briše korisnički račun nego se samo prava smanjuju s novinarskih na običnog korisnika.
- Pregled statističkih podataka – na stranici statistika administrator može pregledavati statističke podatke o čitanosti članaka i aktivnosti novinara na portalu. Administratoru se tako prikazuju 2 stupčasta grafa od kojih jedan prikazuje 10 najčitanijih članaka na portalu temeljem broja pregleda, a drugi 10 najbolje ocjenjenih članaka temeljem reakcija korisnika. Uz to postoje i 4 grafa pite od kojih jedan prikazuje čitanost svake kategorije temeljem broja pregleda, drugi prikazuje broj članaka objavljenih za svaku kategoriju, treći prikazuje 10 najaktivnijih novinara temeljem broja objavljenih članaka i četvrti prikazuje 10 najčitanijih novinara temeljem broja pregleda njegovih članaka. Grafovi se ažuriraju na svaki novi zahtjev klijenta te se mogu izvesti kao slika.
- Brisanje tuđih komentara i odgovora – administrator može obrisati komentare i odgovore drugih korisnika ako ih smatra neprimjerenima.

7.3. Logički model podataka

Dijagram logičkog modela podataka u nastavku vizualizira entitete aplikacije i njihove odnose (eng. *Entity Relationship Model*). Za implementaciju baze podataka aplikacije koristi se standardni relacijski model podataka.



Slika 6 ER dijagram aplikacije [autorski rad]

Baza podataka aplikacije implementira ukupno 9 tablica od kojih su 6 entitetske (korisnici, članci, odgovori, komentari, uloge, kategorije) i 3 asocijativne (reagira_odgovor, reagira_komentar, reagira_clanak). U nastavku slijedi popis objašnjenja odnosa između entiteta:

uloge (1) : (M) korisnici – svaki korisnik mora imati jednu ulogu dok ista uloga može biti dodijeljena različitim korisnicima.

korisnici (1) : (M) clanci – svaki korisnik može napisati više različitih članaka dok je specifični članak je napisao samo jedan korisnik.

kategorije (1) : (M) clanci – svaki članak mora imati određenu kategoriju dok ista kategorija može biti dodijeljena različitim člancima.

clanci (1) : (M) komentari – svaki članak može imati više komentara dok specifični komentar pripada samo jednom članku.

komentari (1) : (M) odgovori – svaki komentar može imati više odgovora dok specifični odgovor pripada samo jednom komentaru.

korisnici (1) : (M) komentari, korisnici (1) : (M) odgovori – svaki korisnik može objaviti više različitih komentara ili odgovora na komentare dok specifični komentar i odgovor pripadaju samo jednom korisniku.

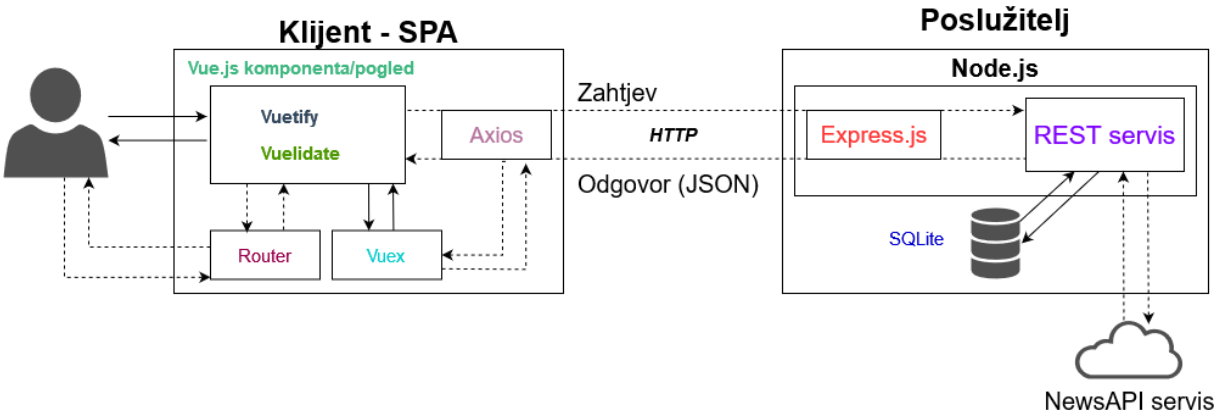
reagira_clanak, reagira_komentar, reagira_odgovor –. asocijativne (vezivne) tablice koriste se za implementaciju odnosa više naprama više (M:M) budući da relacijski model podataka ne dozvoljava direktnu implementaciju takvog odnosa. Sve tri tablice koriste se za implementaciju jednake funkcionalnosti – reakcije korisnika (na članak, komentar ili odgovor) te se objašnjava tako da korisnik može reagirati na više različitih članaka, komentara ili odgovora, a svaki članak, komentar ili odgovor može sadržavati reakcije od više različitih korisnika.

7.4. Arhitektura aplikacije

Arhitektura aplikacije dijeli se na dva temeljna sloja – klijentska aplikacija i poslužiteljska aplikacija. Poslužiteljska aplikacija implementira servis za prijenos reprezentativnog stanja (*eng. Representational State Transfer* – u nastavku akronim REST) koji komunicira s bazom podataka i poslužuje podatke iz baze u formatu JavaScript objekta (*eng. JavaScript Object Notation* – u nastavku akronim JSON). REST servis isto tako komunicira s vanjskim API servisom s kojeg na zahtjev klijenta dohvaća podatke o člancima te ih nakon obrade vraća klijentu. Podaci sa poslužitelja dohvaćaju se korištenjem Axios klijenta te se prosljeđuju zadanoj komponenti ili Vuexu. Okvir Express apstrahira rukovanje HTTP zahtjevima i odgovorima između klijenta i poslužitelja.

Klijentska aplikacija razvijena je kao jednostranična aplikacija pa se tako korisniku prikazuje samo jedna stranica, a Vue router mijenja komponente i poglede unutar iste stranice sukladno korisnikovoj navigaciji. Klijent tako poslužitelju šalje samo zahtjeve za podacima te poslužitelj proporcionalno tome odgovara samo JSON podacima. Podatci koji su potrebni većem broju komponenata spremaju se u globalno spremište te postaju dostupni kompletnoj

klijentskoj aplikaciji. Slika 7 vizualizira opisanu arhitekturu te tok podataka, zahtjeva i odgovora između klijenta i poslužitelja.



Slika 7 Dijagram arhitekture aplikacije [autorski rad]

7.5. Demonstracija funkcionalnosti aplikacije

Naredna poglavlja proučavaju izvorni kod razvijene aplikacije te pripadajuće funkcionalnosti i izgled korisničkog sučelja. Većina funkcionalnosti dijeli slične korake koji se slijede pri implementaciji. Na primjer, za svaki se obrazac validacija provodi prvo na klijentu, a zatim na poslužitelju nakon čega se podaci u slučaju zadovoljenja svih uvjeta validacije spremaju u bazu. Korisnik se obavještava odgovarajućom porukom ako validacija nije uspješno provedena. Zbog tih sličnosti u implementaciji različitih funkcionalnosti, kompletni kod koji uključuje i klijenta i poslužitelja će se objasniti samo na primjeru registracije. Za ostale funkcionalnosti detaljnije će se proučavati kod na klijentu. Kako bi se olakšalo čitanje i minimizirala količina koda neće se proučavati kompletne komponente pa se će se rupe u kodu označavati s tri točke.

7.5.1. Konfiguracija Vue Routera

Ako se u projektu koriste Vuex i Router tada će okruženje pri inicijalizaciji projekta kreirati pripadajuće direktorije i index.js datoteke za obje biblioteke. Direktorij pod nazivom store kreira se za Vuex, a direktorij router za Router. Inicijalne index.js datoteke koriste se za konfiguraciju biblioteka. Isječak koda u nastavku prikazuje konfiguraciju Router biblioteke definiranjem putanja za različite poglede i komponente aplikacije.

```
const routes = [
  {
    path: '/',
```

```
    name: 'home',
    component: IndexView
  },{
    path: '/registracija',
    name: 'registracija',
    component: RegistracijaView
  }, ...{
    path:`/:catchAll(.*)*`,
    name:'notfound',
    component: NotFoundView
  },]
const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes
})
```

Navigacija kroz aplikaciju definira se u App.vue komponenti korištenjem elementa <router-link> čijem se atributu „to“ prosljeđuje putanja definirana u index.js datoteci.

```
<nav> <router-link to="/">Početna</router-link>|
<router-link to="/registracija">Registracija</router-link>|
... </nav><main> <router-view/> </main>
```

Element <router-view/> predstavlja čvor unutar kojeg se izmjenjuju pogledi i komponente. Time se realizira implementacija jednostranične aplikacije jer se korisnik kreće između različitih pogleda i komponenti koje se ustvari mijenjaju na jednom čvoru.

7.5.2. Registracija

Proces registracije realizira se popunjavanjem obrasca u koji korisnik opcionalno unosi ime i prezime te obvezno vjerodajnice adresu e-pošte i lozinku koje će unositi prilikom prijave u aplikaciju. Nakon što korisnik unese podatke, provodi se validacija obrasca na klijentu te se ovisno o rezultatu validacije podaci šalju poslužitelju na obradu ili se korisniku ispisuje poruka o neispravno popunjenom obrascu. Svaki obrazac se prvo provjerava na klijentu kako bi se smanjio broj neispravnih zahtjeva prema poslužitelju i tako osnažila sigurnost i optimizirale performanse aplikacije. Kod u nastavku prikazuje funkciju koja ovisno o rezultatu validacije, ili

šalje zahtjev s podacima prema poslužitelju, ili ispisuje poruku pogreške za neispravno popunjena polja.

```
<script>
async registriraj() {
this.errorMessage = ''; this.v$.$validate();
try {
     await Korisnik.registracija({
     ime: this.ime, prezime: this.prezime, email: this.email,
     lozinka: this.lozinka});
     this.$router.push('/prijava');
} catch (error) {
     if(error.response && error.response.data &&
     error.response.data.pogreska){
          this.errorMessage = error.response.data.pogreska;
     }else{
          this.errorMessage = 'Pogreška prilikom
          registracije'.;}
}}... </script>
```

Validaciju provodi Vuelidate biblioteka unutar ugrađene validate() funkcije i korištenjem ugrađenih svojstava pogrešaka. Deklarativnim renderiranjem se ispisuje poruka pogreške kada validate() funkcija otkrije pogrešku. Isječak iz predloška prikazuje definiranje svojstava pogrešaka za varijablu adrese e-pošte i deklarativno renderiranje odgovarajuće pogreške:

```
<p class="error" v-if="v$.email.$invalid &&
!v$.email.required.$invalid">Ne ispravna adresa e-
računa</p><p class="error" v-if="
v$.email.required.$invalid&& v$.email.$dirty">Obvezno
polje</p><br>
```

Pravila validacije definiraju se ručno unutar validations() funkcije koja je samostalni opcijski objekt komponente.

```
<script>
validations(){
```

```
        return{
                email: {required, email},
                lozinka: { required, minLength: minLength(8),
                maxLength: maxLength(16) }
        }},
... </script>
```

Ako je korisnik ispravno popunio obrazac, poziva se asinkrona funkcija registracija() iz klase Korisnik. Za svaki entitet u aplikaciji kreirana je posebna klasa koja apstrahira Axios funkcije slanja zahtjeva prema poslužitelju. Isječak koda u nastavku prikazuje klasu Api i Korisnik koje implementiraju funkcije slanja zahtjeva prema poslužitelju.

Klasa Api.js:

```
import axios from 'axios'
const Api = () => {
return axios.create({
        baseURL: 'http://localhost:3000/',
        withCredentials:true});};
export default Api;
```

Klasa Korisnik.js:

```
import Api from '@/services/Api'
export default{
        registracija(vjerodajnice){
        return Api().post('/rest/korisnici', vjerodajnice)},
        dohvatiKorisnike(){
        return Api().get('/rest/korisnici');},
        dohvatiPrijavljenog(){
        return Api().get('/rest/korisnik');
}...
```

Pristiglim zahtjevom na poslužitelju rukuje Express funkcija post() kojoj se kao prvi argument prosljeđuje putanja do pripadajućeg resursa i kao drugi argument funkcija povratnog poziva (*eng. Callback*).

Datoteka server.mjs:

```
...function pripremiPutanjeKorisnici(server)
{server.get("/rest/korisnici/", restKorisnik.getKorisnici);
server.post("/rest/korisnici", restKorisnik.postKorisnici);
server.put("/rest/korisnici", restKorisnik.putKorisnici);
server.delete("/rest/korisnici", restKorisnik.deleteKorisnici);
... }
```

Funkcija povratnog poziva postKorisnici obrađuje pristigli zahtjev i pripadajuće podatke. Za svaki entitet u aplikaciji postoji objekt restNazivEntiteta koji apstrahira funkcije povratnih poziva za rukovanje zahtjevima, odgovorima i podacima. U nastavku slijedi isječak koda funkcije post korisnici:

```
exports.postKorisnici = async function(zahtjev, odgovor) {
odgovor.type("application/json"); let podaci = zahtjev.body;
if(!podaci.email || !podaci.lozinka || podaci.lozinka.length < 8
|| podaci.lozinka.length > 16){
odgovor.status(400);return odgovor.send(JSON.stringify({
pogreska: "Neispravno popunjen obrazac" }));}
let kdao = new KorisnikDAO();
try {
     const jedinstvenEmail = await
     kdao.jedinstvenEmail(podaci.email);
if (!jedinstvenEmail) {
     odgovor.status(400);
     return odgovor.send(JSON.stringify({ pogreska: `Korisnik s
     e-računom ${podaci.email} već postoji.
     }));}
const poruka = await kdao.dodaj(podaci);
```

```
odgovor.send(JSON.stringify(poruka));
} catch (error) {
      odgovor.status(500).send(error.message);
}}
```

Podaci obrasca registracije šalju se u tijelu HTTP zahtjeva te se na poslužitelju još jednom provodi validacija pristiglih podataka. Nakon toga se provjerava jedinstvenost adrese e-računa. Registracija se odbija ako adresa e-računa nije jedinstvena. Ako je adresa jedinstvena, poziva se funkcija dodaj(podaci) objekta za pristupanje podacima (*eng. Data Access Object* – u nastavku akronim DAO). Funkcije DAO objekta definiraju SQL upite za manipulaciju podacima u bazi. Važno je za napomenuti da su REST funkcije definirane unutar try{...}catch(pogreska){...} blokova koji hvataju sve emitirane pogreške nastale prilikom rukovanja s pristiglim podacima. Time se minimizira rizik nedostupnosti REST servisa u slučaju da se dogodi pogreška.

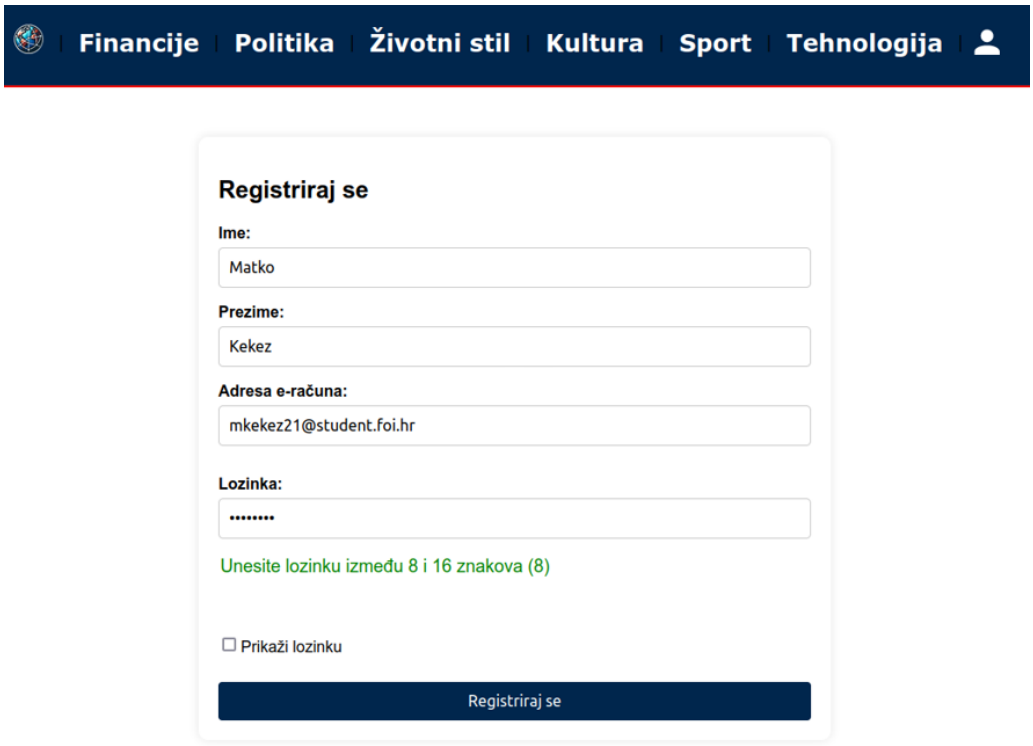
Za svaki entitet razvijena je posebna DAO klasa. Isječak koda u nastavku prikazuje KorisnikDAO klasu i dodaj(podaci) funkciju:

```
class KorisnikDAO{
constructor(){
this.baza = new Baza("./database/novinski_portal.sqlite");} ...
dodaj = async function(korisnik){
      this.baza.spojiSeNaBazu();
      let hashLozinke = await bcrypt.hash(korisnik.lozinka, 10);
      let sqlUpit = "INSERT INTO korisnici (email, lozinka, ime,
      prezime, ulogaID) VALUES (?,?,?,?,?)";
      let podaci = [korisnik.email,hashLozinke,korisnik.ime,
      korisnik.prezime,2];
      await this.baza.izvrsiUpit(sqlUpit, podaci);
      this.baza.zatvoriVezu();
      return {"Izvrseno":"ok"};
}}
module.exports = KorisnikDAO;
```

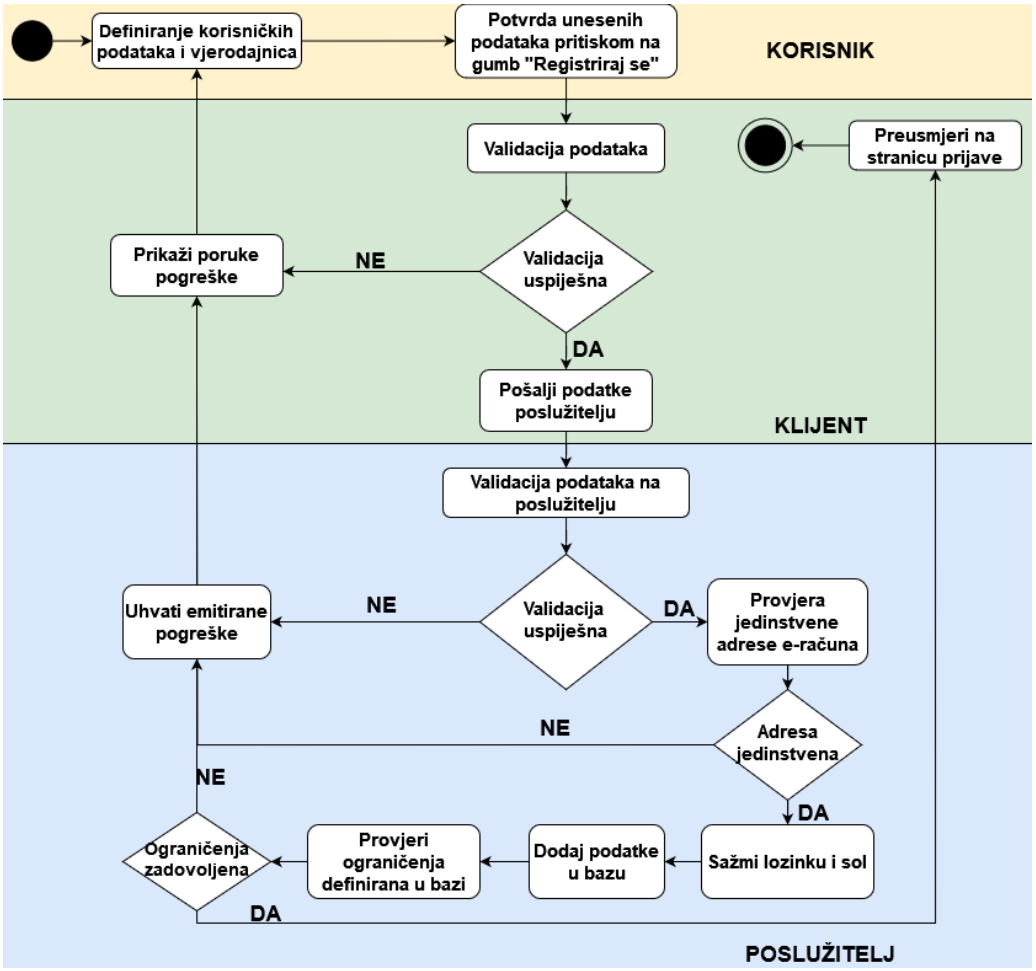
Funkcija izvrsiUpit(sqlUpit, podaci) šalje upit i pripadajuće podatke prema SQLite datoteci:

```
const SQLite = require("sqlite3").Database;
class Baza {
constructor(putanjaSQLliteDatoteka) {
      this.vezaDB = new SQLite(putanjaSQLliteDatoteka);
      this.putanjaSQLliteDatoteka = putanjaSQLliteDatoteka;
      this.vezaDB.exec("PRAGMA foreign_keys = ON;");
}
...izvrsiUpit(sql,podaciZaSQL){
return new Promise((uspjeh,neuspjeh)=>{
this.vezaDB.all(sql,podaciZaSQL,(pogreska,rezultat)=>{
if(pogreska){
      console.log(pogreska); neuspjeh(pogreska);
}else {
      uspjeh(rezultat);
}});
});
}...zatvoriVezu(){
this.vezaDB.close();
}}
module.exports = Baza;
```

Slika 8 prikazuje izgled korisničkog sučelja stranice registracije i popunjenog obrasca registracije. Nakon uspješne registracije korisnik se preusmjerava na stranicu prijave.



Slika 8 Sučelje popunjenog obrasca registracije [autorski rad]



Slika 9 Dijagram aktivnosti za funkcionalnost registracije [autorski rad]

7.5.3. Dodavanje novog članka

Novi članak na portalu može napisati i objaviti samo korisnik s ulogom novinara. Pri pisanju članka novinar može koristiti članke dohvaćene s vanjskog News API servisa ili može samostalno napisati kompletni članak. U nastavku će se proučiti isječci koda, izgled sučelja i scenarij korištenja kada novinar dodaje novi članak korištenjem vanjskog servisa.

Na stranici „pretraži članke“ novinar u tražilicu unosi ključnu riječ temeljem koje se s servisa dohvaćaju relevantni članci. Servis može vratiti najviše 100 relevantnih članaka, a ako se za unesenu ključnu riječ ne pronađe niti jedan rezultat, od novinara se zahtjeva da ponovno unese ključnu riječ. Isječak koda iz komponente pretraži članke:

```
<template> ... <label for="pretraziClanke">Pretraži članke
NewsAPI:</label> <input type="text" id="filter"
placeholder="Pretraži članak"
@keyup.enter="pretraziClanke()" v-model="pretrazivanje"
class="input-pretrazi"><h2 v-if="clanci.length > 0"
class="pretrazivanje-naslov">Rezultati pretraživanja</h2><div
class="kartice-sekcija"><div class="kartica" v-for="clanak in
clanciPaginated" :key="clanak.id"><img :src="clanak.slika_url"
alt="Card Image" class="kartica-slika"><div class="kartica-
tekst"><h2 class="kartica-naslov">{{ clanak.naslov }}</h2><p
class="kartica-sazetak">{{ clanak.sazetak }}</p><p class=
"kartica-sadrzaj">{{clanak.sadrzaj}}</p><p class="poveznica">
Više na poveznici:</p><a :href="clanak.url" target="_blank"
rel="noopener noreferrer">{{ clanak.url }}</a></div><div class=
"action-gumb"><button @click.prevent="odaberiClanak (clanak)
">Dodaj članak</button></div></div></div> ... </template>
```

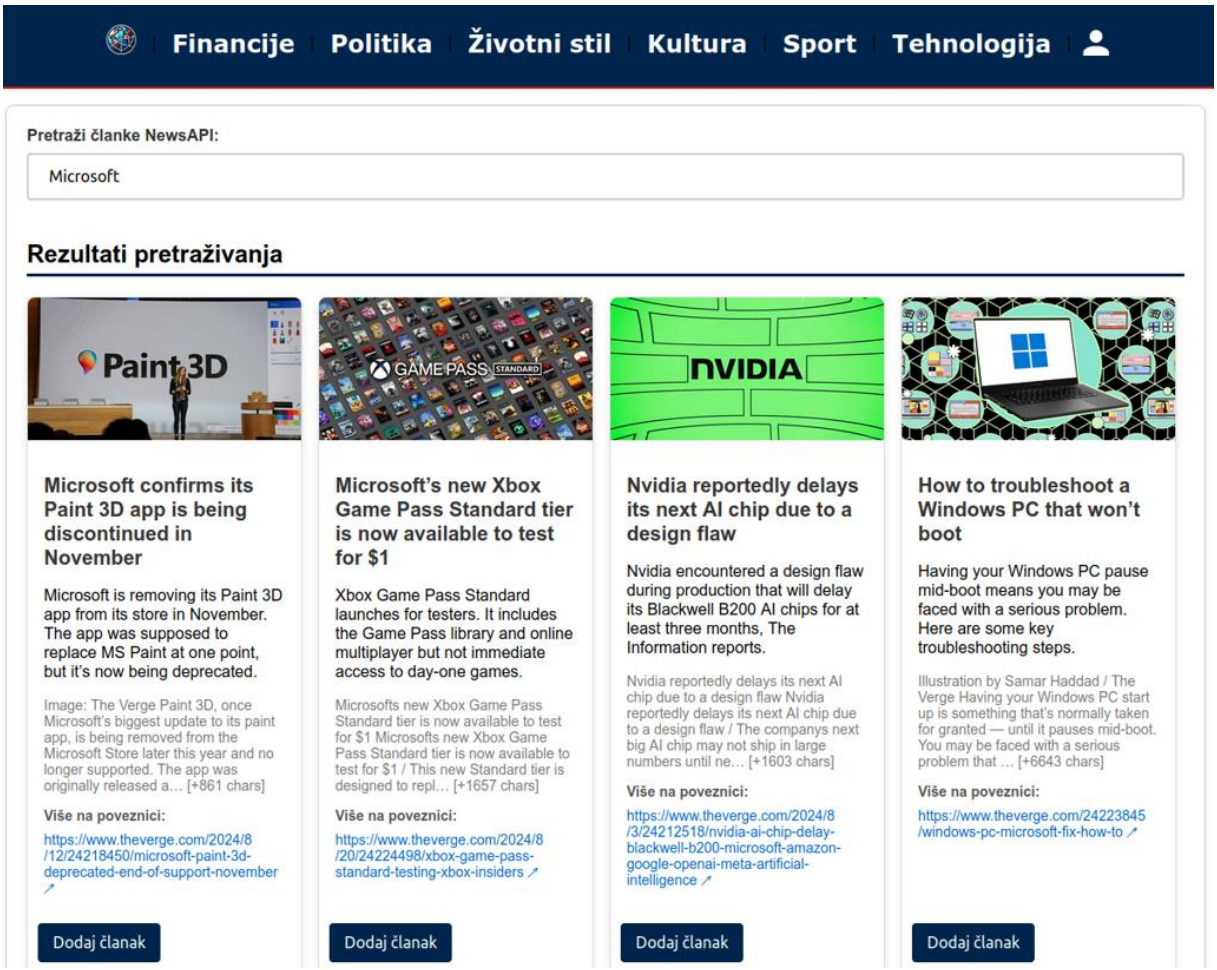
Isječak koda iz predloška implementira tražilicu i kontejner koji sadrži kartice u kojima se prezentiraju rezultati pretraživanja. Podebljani dio koda prikazuje kako se iterira kroz dohvaćene rezultate te se za svaki dohvaćeni članak renderira nova kartica. Kod u nastavku isječak je iz <script> bloka iste komponente:

```
<script> ...
data(){
```

```
return{
      pretrazivanje: '', clanci: [],selectedClanak: null,
      errorMessage: null,trenutnaStranica: 1,
      clanciPoStranici: 20,
      }
}... methods:{
async pretraziClanke() {
      try {
            const odgovor = await
            Clanak.dohvatiClankeApi(this.pretrazivanje);
            this.clanci = odgovor.data
            this.errorMessage = this.clanci.length === 0 ?
            `Pretraga - ${this.pretrazivanje} - nije pronašla niti
            jedan članak. Pokušajte ponovno.` : ''}
      catch (error) {
            this.errorMessage = error.message;}},
... </script>
```

Isječak koda prikazuje varijable u koje se pohranjuju podaci komponente unutar data() opcijskog objekta te funkciju pretraziClanke() koja se poziva kada polje za unos teksta detektira pritisak i puštanje tipke enter.

Dohvaćeni članci spremaju se u polje clanci te se u predlošku v-for direktivom iterira kroz polje i za svaki element renderira nova kartica. Ako je polje prazno, novinaru se ispisuje odgovarajuća poruka. Slika 9 prikazuje rezultate pretraživanja za ključnu riječ „Microsoft“.



Slika 10 Sučelje komponente za pretraživanje i pregled članaka s News API servisa [autorski rad]

Klikom na gumb dodaj članak novinaru se otvara nova komponenta koja sadrži obrazac za dodavanje novog članka, a koji je popunjen podacima članka koji je novinar odabrao. Novinar sve podatke može promijeniti i prilagoditi vlastitim preferencijama. Zbog ograničenja besplatne verzije News API servisa, novinar mora samostalno napisati sadržaj članka. Obrazac za dodavanje novog članka posebna je komponenta koja podatke o članku prima od roditeljske komponente kroz props opcijski objekt. Kod u nastavku prikazuje kako roditeljska komponenta (PretraziClankeKomponenta) prosljeđuje podatke komponenti djetetu odnosno komponenti za dodavanje članaka (DodajClanakKomponenta).

Komponenta pretraživanje članaka – roditelj:

```
<DodajClanakKomponenta v-else-if="selectedClanak"
:clanakApi="selectedClanak":sadrzajUrl="sadrzajUrl"/>
```

Komponenta dodavanje članka – dijete:

```
<template> ... <div class="form-group">
```

```
<label for="naslov">Naslov članka:</label><small>Znakova:{{
naslov.length }} (min:10, max:90)</small>
<input type="text" name="naslov"  v-model="naslov"
placeholder="Naslov"/> <div class="message-input">
<p class="error" v-if="v$.naslov.required.$invalid &&
v$.naslov.$dirty">Obvezno polje</p> ... </template>
```

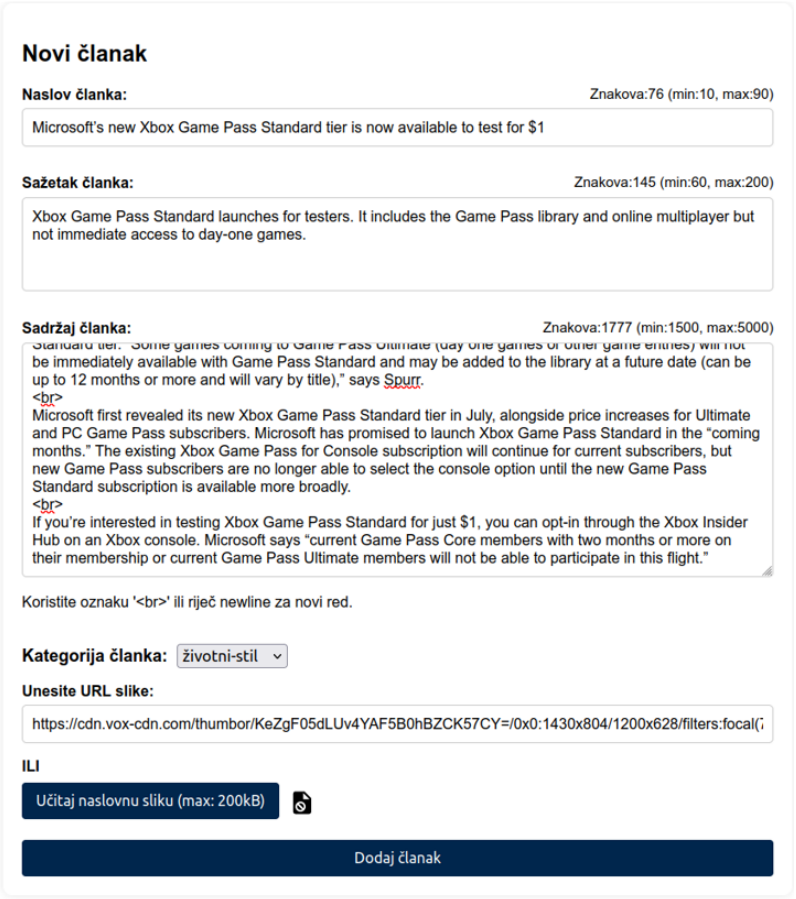
Direktivom v-model ostvaruje se dvosmjerno vezivanje podataka između varijable naslov i <input> elementa za unos naslova. Naslov varijabli pridružuje se vrijednost naslova koja je primljena kroz clanakApi prop od roditeljske komponente. Ako od roditeljske komponente nije primljen nikakav prop pretpostavlja se da novinar želi samostalno popuniti članak te se polje za unos naslova ostavlja prazno. Podebljane linije koda u nastavku prikazuju inicijalizaciju naslov varijable i prop opcijski objekt koji prima podatke od roditeljske komponente.

```
<script> export default {
data(){
      return{
            v$: useVuelidate(),
            naslov: this.clanakApi.naslov || '',
            sazetak: this.clanakApi.sazetak || '',
            sadrzaj: this.sadrzajUrl || '', ...
            }
...}props: {
      clanakApi: {type: Object,default: () => ({}) },
      sadrzajUrl: {type: String}, },
```

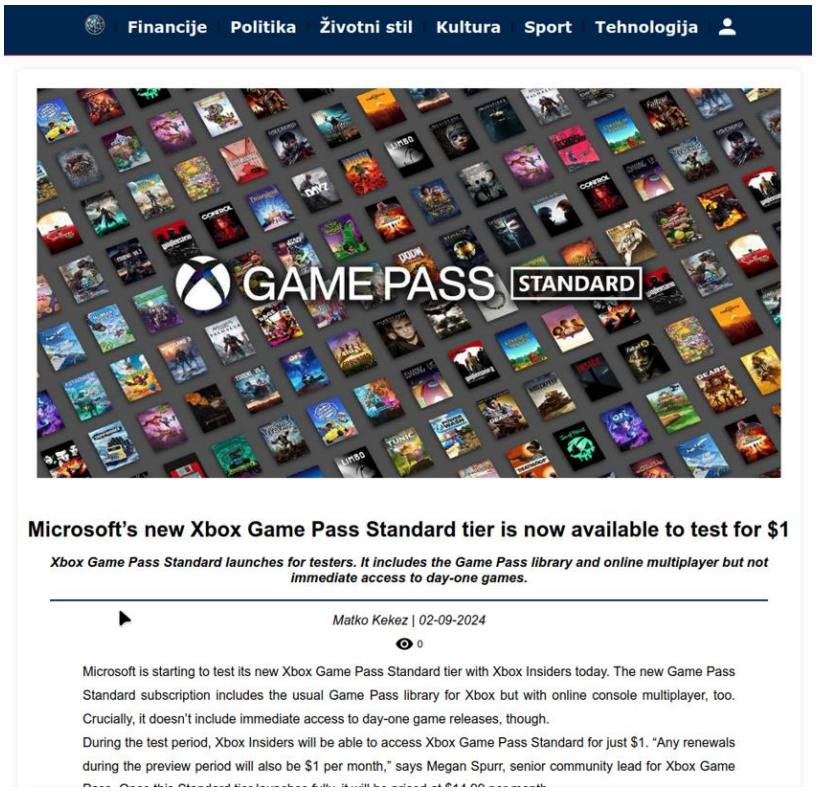
Prilikom pisanja članka novinar se mora pridržavati znakovnih ograničenja kojima se definira duljina naslova, sažetka i sadržaja. Naslovna slika može se pohraniti kao URL do resursa na webu ili kao datoteka slike koja ne smije biti veća od 200 kilobajta. Ako su sva ograničenja zadovoljena, klikom na gumb dodaj članak podaci se šalju poslužitelju gdje se još jednom provodi validacija te ovisno o rezultatu validacije članak se dodaje u bazu ili se odbija dodavanje. Isječak koda u nastavku prikazuje slanje podataka novog članka prema poslužitelju:

```
async dodajClanak(){
this.errorMessage = '';this.v$.$validate();
if(this.v$.$invalid) return;
try{
     if (this.slika && this.slikaUrl) {
          this.objeSlikeOdabrane = true; return;
     } else {
          this.objeSlikeOdabrane = false;}
     const formData = new FormData();
     if (this.slika && !this.slikaUrl) {
          formData.append('slika', this.slika);
     }else if (this.slikaUrl && !this.slika) {
          formData.append('slika_url', this.slikaUrl);
     }
     formData.append('naslov', this.naslov);
     formData.append('sazetak', this.sazetak);
     formData.append('sadrzaj', this.sadrzaj);
     formData.append('kategorija', this.kategorija);
     const odgovor = await Clanak.dodajClanak(formData);
     const NoviClanak = odgovor.data;this.$router.push(
     {name:'clanak',params:{id: NoviClanak.idclanci } });
}catch (error) {
     if(error.response && error.response.data &&
     error.response.data.pogreska){
          this.errorMessage = error.response.data.pogreska;}
     else{
          this.errorMessage = 'Došlo je do pogreške prilikom
          dodavanja članka.';
}}}
```

Slika 10 prikazuje popunjeni obrazac za dodavanje novog članka. Ako je novi članak uspješno dodan, novinara se preusmjerava na stranicu istog – slika 11.



Slika 11 Sučelje popunjenog obrasca za dodavanje novog članka [autorski rad]



Slika 12 Sučelje stranice članka [autorski rad]

7.5.4.Reagiranje na članak

Klikom na ikonu palca prema gore ili palca prema dolje, korisnik ostavlja povratnu informaciju na pročitani članak. Ovisno o reakciji korisnika, ikonama se dinamički pridružuju stilske upute koje korisniku daju povratnu informaciju o tome da je već reagirao na pročitani članak. Isječci koda u nastavku prikazuju dio predloška i skripte kojima se implementira korisnikova reakcija na članak.

```
<v-icon @click.prevent="azurirajReakciju(1)" size:1.5
:icon="svidaIkona" :class="reakcijaKorisnika === 1 ?
'svidaAktivno' : 'neaktivno'" ></v-icon>
<p :class="reakcijaKorisnika === 1 ? 'svidaAktivno' :
'neaktivno'" > {{ clanak.svida_broj }} </p><div class="ocjena">
<v-icon size: :icon="ocjenaIkona" id="ocjena-ikona"></v-icon>
<p>{{ clanak.ocjena }}</p></div> <v-icon
@click.prevent="azurirajReakciju(-1)" size:1.5
:icon="neSvidaIkona" :class="reakcijaKorisnika === -1 ?
'neSvidaAktivno' : 'neaktivno'"></v-icon>
<p :class="reakcijaKorisnika === -1 ? 'neSvidaAktivno' :
'neaktivno'"> {{ clanak.ne_svida_broj }} </p>
```

Klikom na ikonu palca prema gore ili palca prema dolje, poziva se funkcija azurirajReakciju(argument) kojoj se kao argument prosljeđuje borj 1 ako je reakcija pozitivna ili -1 ako je reakcija negativna. Funkcija azurirajReakciju šalje podatke o novoj reakciji prema poslužitelju i prima od poslužitelja odgovor s ažuriranim podacima o broju pozitivnih i negativnih reakcija i ocjeni članka.

```
async azurirajReakciju(reakcija){
if(!this.korisnik) return
try{
      let podaci = await Clanak.dodajReakciju({
      id: this.clanak.idclanci, reakcijaBody: reakcija});
      if(this.reakcijaKorisnika === reakcija){
            this.reakcijaKorisnika = null;
      }else if(this.reakcijaKorisnika === null){
```

```
            this.reakcijaKorisnika = reakcija;}
        else{
            this.reakcijaKorisnika = reakcija;
        }
        this.clanak.svida_broj = podaci.data.svidaBroj;
        this.clanak.ne_svida_broj = podaci.data.neSvidaBroj;
        this.clanak.ocjena = podaci.data.ocjena;
    }catch(error){
        if(error.response && error.response.data &&
        error.response.data.pogreska){
            this.errorMessage = error.response.data.pogreska}
        else{
            this.errorMessage = error.message;
            console.log(this.errorMessage);}
    }}
```

Podaci se nakon validacije na poslužitelju spremaju u bazu te se izračunava nova ocjena članka koja se zajedno s podacima o ažuriranom broju pozitivnih i negativnih reakcija šalje klijentu:

```
exports.postReakcija = async function(zahtjev, odgovor){
odgovor.type("application/json");
let korisniciID = zahtjev.session.userId;
let clanciID = zahtjev.body.id;
let reakcija = zahtjev.body.reakcijaBody;
if(!korisniciID){
    return odgovor.status(401).send(JSON.stringify({pogreska:
    "Prijavite se ili registrirajte kako bi mogli reagirati."
}));}
let cdao = new ClanciDAO();
try{
    cdao.baza.spojiSeNaBazu();
    await cdao.baza.zapocniTransakciju();
```

```
            letocjenaParametri = await
            cdao.azurirajReakciju(korisniciID, clanciID, reakcija);
            if(ocjenaParametri){
                  let zbrojSvida = ocjenaParametri.svida_zbroj * 10;
                  let zbrojNeSvida = ocjenaParametri.ne_svida_zbroj;
                  let svidaBroj = ocjenaParametri.svida_broj;
                  let neSvidaBroj = ocjenaParametri.ne_svida_broj;
                  let ukupno = svidaBroj + neSvidaBroj;
                  let ocjena = ukupno > 0 ?
                  (zbrojSvida+zbrojNeSvida)/ukupno:0;
                  await cdao.azurirajOcjenu(ocjena, clanciID);
                  await cdao.baza.izvrsiTransakciju();
                  return odgovor.status(200).json({
                  ocjena: ocjena.toFixed(2),
                  svidaBroj: svidaBroj, neSvidaBroj: neSvidaBroj});}
      }catch(error){
            await cdao.baza.vratiTransakciju();
            console.log(error);
            return odgovor.status(500).send(JSON.stringify({pogreska:
            "Dogodila se pogreska."}))
      }finally{
            cdao.baza.zatvoriVezu();
      }}
```

Morgana ili kontaktirajte najbližu poslovnicu. Financijska stabilnost i sigurnost su nadohvat ruke s JP Morgan kreditima.

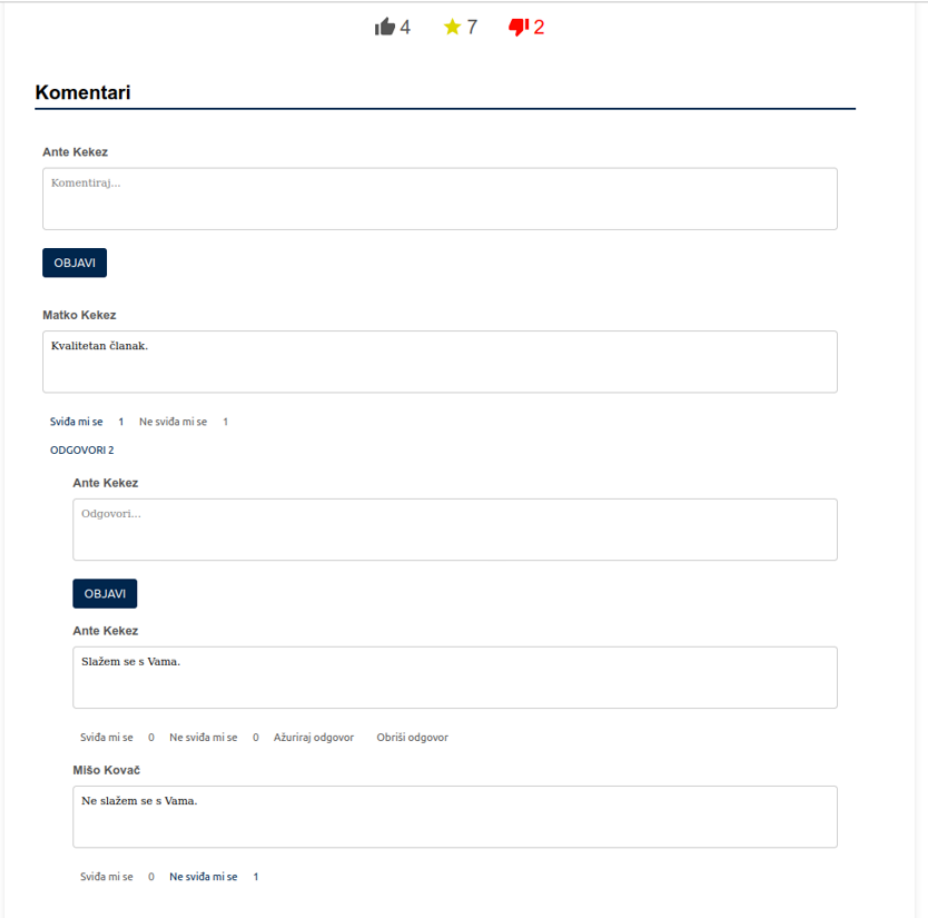
6 ★ 8.71 1

Slika 13 Sučelje sekcije reakcija na članak i pripadajuće ocjene [autorski rad]

7.5.5.Komentari i odgovori

Komentari i odgovori implementirani su pridržavajući se istih principa kao i prethodne funkcionalnosti te se stoga izvorni kod neće detaljno objašnjavati. Važno je za napomenuti da se komentari učitavaju zajedno sa člankom unutar pogleda ClanakView dok je za odgovore

implementirano lijeno učitavanje (*eng. Lazy Loading*) pa se dohvaćaju tek kada korisnik klikne gumb odgovori ispod specifičnog komentara. Slika 13 prikazuje sučelje sekcije s komentarima i pripadajućim odgovorima.



Slika 14 Sučelje sekcije s komentarima i pripadajućim odgovorima [autorski rad]

7.5.6. Dodavanje pregleda

Pregledi se za svaki članak dodaju samo ako prijavljeni korisnik na stranici specifičnog članka provede više od 30 sekundi. Kuka životnog ciklusa created() dodjeljuje varijabli pocetak vrijeme u trenutku kada se korisniku prikaže odabrani članak. Novi pregled se dodaje u trenutku kada korisnik napusti stranicu trenutno otvorenog članka, napusti trenutnu karticu, minimizira ili zatvori preglednik. To se realizira kukom životnog ciklusa breforeMount() koja poziva funkciju za dodavanje pregleda neposredno prije nego što se komponenta novinskog članka uništi. Funkcija za dodavanje pregleda dodajPregled() računa vrijeme provedeno na stranici kao razliku vremena od trenutka stvaranja komponente do uništavanja komponente. Pregled se dodaje ako je reakcija veća od 30:

```
methods: {
async dodajPregled(){
```

```
        if (this.pregledPoslan || this.pregledSeSalje) return;
        this.pregledSeSalje = true this.provedenoVrijeme =
        Math.floor((Date.now() - this.pocetak) / 1000);
        if(this.provedenoVrijeme > 30){
              try{
              await Clanak.dodajPregledClanak({
              id:this.id,provedenoVrijeme: this.provedenoVrijeme})
              this.pregledPoslan = true;}
              catch(error){
              console.log(error.message);
              }finally {
              this.pregledSeSalje = false;}}
        },
        upravljajPromjenomKartice(){
        if(document.visibilityState==='hidden'){
              this.dodajPregled();
        }}},
...created(){
        this.dohvatiClanak();this.pocetak = Date.now();
        document.addEventListener('visibilitychange',
        this.upravljajPromjenomKartice);
},beforeUnmount(){
        this.dodajPregled();
        document.removeEventListener('visibilitychange',
        this.upravljajPromjenomKartice);
}
```

8. Zaključak

Proporcionalno snažnoj migraciji poslovnih i društvenih aktivnosti na web, povećavaju se i zahtjevi za kvalitetnijim web aplikacijama koje će uspješno podržati te aktivnosti. Nesklad između ponude i potražnje za web stranicama rezultirao je pojavom novih tehnologija koje olakšavaju i ubrzavaju razvoj web aplikacija čime se taj raskorak minimizira jer razvojni inženjeri više ne moraju pisati velike količine robusnog koda i razumjeti kako stroj interpretira kod u pozadini čitavog procesa. Svi okviri korišteni za razvoj aplikacije u sklopu ovog rada mogu se klasificirati kao te tehnologije. Tako, na primjer, okvir Express omogućuje jednostavno upravljanje HTTP zahtjevima i odgovorima pa razvojni inženjer ne mora ručno konfigurirati zahtjeve i odgovore pa čak ni u potpunosti razumjeti kako u pozadini Express njima rukuje. To rezultira čišćim kodom te bržim i jednostavnijim razvojem web aplikacija. Ista teza može se aplicirati i na Vue okvir koji je središnja tema rada.

Vue tako ugrađuje reaktivnost i deklarativno renderiranje koji osiguravaju razvoj modernih i složenih klijentskih web aplikacija minimizirajući pri tome količinu programskog koda. Oba koncepta apstrahiraju JavaScript kod koji prilikom interpretacije aplikacije rukuje ažuriranjem i vezivanjem podataka na DOM. Razvojni inženjer tako ne mora ručno razvijati slične funkcije čime se minimizira količina koda za redundantne aktivnosti. Time se uočava prva značajnija prednost korištenja Vue okvira: minimiziranje vremena i napora potrebnih za razvoj redundantnih funkcionalnosti koje su kroz svaku web aplikaciju identične.

Isto tako, Vue omogućuje jednostavan razvoj jednostraničnih web aplikacija. Razvojni inženjer je tako ograničen razvijati komponente i poglede koje će se proporcionalno korisnikovoj navigaciji kroz aplikaciju izmjenjivati na istom čvoru DOM-a. Pri tome se čitava arhitektura aplikacije zasniva na stablu povezanih komponenata čiji se odnosi mogu opisati analogijom roditelj-dijete. Svaka komponenta može se promatrati kao komponenta roditelj ili kao komponenta dijete (osim korijenske komponente koja nema roditelja). Ovakva arhitektura definira tok podataka od roditelja prema djetetu i onemogućuje direktnu komunikaciju između komponenata što znači da je svaka komponenta neovisna te joj se ne može pristupiti iz okoline. Ovakvo ograničenje arhitekture može se promatrati kao prednost jer razvojni inženjer može jednostavno iskoristiti obrazac dizajna koji definira okvir i uz minimalne napore razviti modernu web aplikaciju. Isto tako, ovakva arhitektura može predstavljati i veliki problem jer ograničava fleksibilnost u razvoju pa se u napori i vrijeme povećavaju u situacijama kada određene funkcionalnosti zahtijevaju odstupanje od arhitekture definirane okvirom.

Posljednja značajna prednost Vue okvira njegova je progresivnost što omogućuje jednostavnu inkrementalnu nadogradnju postojećih aplikacija i njegova jednostavna integracija u aplikacije koje su razvijene različitim tehnologijama.

U konačnici se može zaključiti da tehnologije kao Vue okvir značajno ubrzavaju i olakšavaju razvoj modernih web aplikacija uz cijenu vremena potrebnog da se nauče njihovi koncepti i sintaksa. Isto tako, Vue i slične tehnologije formiraju nove, i eliminiraju tradicionalne kompetencije potrebne za razvoj aplikacija pa tako više nije nužno znatno baratanje matematičkim vještinama da bi se razvila kvalitetna aplikacija. S druge strane, razvojni inženjeri moraju konstantno održavati korak s turbulentnom pojavom novih tehnologija iz čega proizlazi možda i najveći izazov, ali i zadovoljstvo ove industrije – neprestano učenje.

Popis literature

[1] J. N. Robbins, "*Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics*", 5th Edition, O'Reilly Media, 2018.

[2] „A short history of the Web | CERN". Pristupljeno: 30. lipanj 2024. Dostupno na: https://home.cern/science/computing/birth-web/short-history-web

[3] Pramod Hanumappa, „Naslovna slika - What is Frontend and Backend : Data Flow in Software: What is Software - YouTube". Pristupljeno: 08. srpanj 2024. Dostupno na: https://www.youtube.com/watch?v=YGxrvHGCJ2Y

[4] H. Ramon. Ribeiro, *Vue.js 3 Cookbook*. Packt Publishing, 2020.

[5] E. Brown, „*Web Development with Node and Express*", 1st Edition, O'Reilly Media, 2014.

[6] Huzefa Chawre, „Top Web Development Challenges & Solutions to Overcome Them". Pristupljeno: 01. srpanj 2024. Dostupno na: https://www.turing.com/resources/top-web-development-challenges

[7] Joel Olawanle, „What is a Framework? Software Frameworks Definition". Pristupljeno: 01. srpanj 2024. Dostupno na: https://www.freecodecamp.org/news/what-is-a-framework-software-frameworks-definition/

[8] Vivian Cromwell, „Between the Wires: An interview with Vue.js creator Evan You". Pristupljeno: 01. srpanj 2024. Dostupno na: https://www.freecodecamp.org/news/between-the-wires-an-interview-with-vue-js-creator-evan-you-e383cbf57cc4/

[9] Alex Kugell, „15 Global Websites Using Vue.js in 2024 - Trio". Pristupljeno: 01. srpanj 2024. Dostupno na: https://trio.dev/websites-using-vue/

[10] „Introduction | Vue.js". Pristupljeno: 08. srpanj 2024. Dostupno na: https://vuejs.org/guide/introduction.html

[11] „Releases | Vue.js". Pristupljeno: 01. srpanj 2024. Dostupno na: https://vuejs.org/about/releases

[12] John. Au-Yeung, „*Vue. js 3 by Example Blueprints to Learn Vue Web Development, Full-Stack Development, and Cross-platform Development Quickly".* Packt Publishing, Limited, 2021.

[13] F. Vue, H. Djirdeh, N. Murray, i A. Lerner, „The Complete Guide to Vue.js and Friends", 2018.

[14] „Lifecycle Hooks | Vue.js". Pristupljeno: 21. kolovoz 2024. Dostupno na: https://vuejs.org/guide/essentials/lifecycle.html

Popis slika

Popis tablica

Prilog 1

Kompletan izvorni kod aplikacije razvijene u sklopu praktičnog dijela rada dostupan je na sustavu FOI radovi u zip arhivi.