Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 7.005
DOI: https://doi.org/10.24867/30BE05Volaric

KOLABORATIVNA PLATFORMA ZA RAZVOJ REACT APLIKACIJA

COLLABORATIVE PLATFORM FOR DEVELOPING REACT APPLICATIONS

Filip Volarić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Cilj rada jeste istraživanje, projektovanje i razvijanje veb alata koji će omogućiti modelovanje i generisanje React koda za klijentske aplikacije. Kreiran je alat koji omogućuje korisnicima da u timovima razvijaju modele za React projekte, i da na osnovu modela generišu funkcionalne projekte. Osnovna gradivna jedinica React projekta je komponenta, pa je za njihovo modelovanje pružena najveća podrška. Generisani projekat se može pokrenuti pomoću Vite alata za izgradnju, dok se klijentske komponente mogu koristiti i u okviru Next.js radnog okvira.*

Ključne reči: *React, model, generator koda, Vite, komponente*

Abstract – *The work aims to research, design, and develop a web tool for modeling React applications, and generating the code for client applications. The tool enables users to work in teams when developing models for React projects. The basic building block of React application is a component. Therefore, their modeling offers the greatest support. Generated project can be ran with the Vite build tool, but also, generated components can be used within Next.js framework.*

Keywords: *React, model, code generator, Vite, components*

1. UVOD

Motivacija za razvoj aplikacije opisane u radu leži u rastućem broju tehnologija i radnih okvira za razvoj klijentskih aplikacija. Okviri često sadrže mnogo *boilerplate* koda što otežava razvojnim timovima da se fokusiraju na rešavanje domenskih problema. *Loosely-coupled* komponente omogućuju bolju ponovnu upotrebu koda i ubrzavaju razvoj novih funkcionalnosti [1]. Cilj opisanog alata je da korisnicima omogući vizualizaciju sistema, ili njegovih delova na klijentskoj strani, kako bi u ranoj fazi razvoja identifikovali potencijalne probleme u dizajnu.

Rad opisuje rešenje koje pokušava da prevaziđe ove probleme, i pruža integrisani generator koda koji definiše uniformnu strukturu komponenti i drugih resursa kroz celu aplikaciju. Vizualni editor pomaže korisnicima da identifikuju previše povezane komponente i bolje organizuju kod.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bila prof. dr Gordana Milosavljević.

Sistem je razvijen kao veb aplikacija, kako bi bio dostupan širokom broju korisnika, bez potrebe za dodatnim zavisnostima. Funkcionalnost generisanja koda je omogućena pomoću obrađivača šablona, koji kombinuje šablone sa podacima kako bi se proizvele izlazne datoteke. Alat takođe uključuje izbor tehnologije za stilizovanje i integraciju *Redux*-a.

2. REACT

React [2] je prvi put predstavljen javnosti 2013. godine od strane kompanije *Meta* (tada *Facebook*). Iako je bio dostupan, značajan porast u popularnosti zabeležen je nekoliko godina kasnije. Važna prekretnica u razvoju *React*-a bila je uvođenje funkcijskih komponenti i *hook*-ova u verziji 16.8.

Jedan od ključnih koncepata u *React*-u je deklarativno programiranje, koje se primenjuje kroz organizaciju koda u komponente, *props* i *callback* funkcije. U *React* aplikacijama, komponente se posmatraju kao osnovne gradivne jedinice korisničkih interfejsa. *React* elementi, sa druge strane predstavljaju nepromenljive opise onoga što treba da se iscrta na stranici [1].

Kao relativno mlada tehnologija, *React* nema visok nivo standardizacije, i postoji mnogo načina da se implementira identična funkcionalnost. Ovo doprinosi problemu teškog održavanja koda, posebno u okruženjima gde se članovi tima menjaju i gde baza koda brzo raste. Ipak, postoje neke smernice, kao i noviji dizajn obrasci koji pomažu da kod postane lakši za održavanje i podstiču pravu namenu *React*-a kao tehnologije, a to su ponovno iskoristive komponente (*reusable components*).

Ponovna iskoristivost se postiže pomoću parametara komponente (*props*). U *React*-u postoji poseban *prop* koji se može vezati za bilo koju komponentu, a to je *children*. Taj *prop* je zapravo *JSX* koji se može propagirati u dečiju komponentu i prikazati na proizvoljnom nivou *JSX*-a dečije komponente. Posebno je koristan za podelu komponenti na logičke celine, jer nekad nije poželjno kroz parametre deliti informacije koje semantički nemaju smisla da postoje u komponenti.

Kada komponenta prosledi *props* drugoj komponenti, ona se naziva vlasnikom, bez obzira na njihovu relaciju roditelj-dete [1]. U nastavku teksta, vlasnik predstavlja komponentu koja prosleđuje *prop* dok je roditelj direktni čvor *DOM*-a. Principi koji se vezuju za paradigmu funkcionalnog programiranja, poput nepromenljivosti čistih funkcija i funkcija višeg reda pomažu u pisanju koda koji je lakši za održavanje i testiranje.

3. DIZAJN OBRASCI

3.1. Container presentational

Komponente u *React*-u tipično sadrže i logiku i prezentaciju. Pod logikom se podrazumeva sve što je nevezano za *UI*, poput *API* poziva, manipulacije podataka, obrađivača događaja i sl. Sa takvim komponentama, vrlo često minimalna promena zahteva da se logika proširi dodatnim uslovima i parametrima. Jedan način za kreiranje jasne granice između logike i prezentacije je poznat kao *container and presentational*. Ideja je da se složena komponenta, podeli na dve datoteke. Prva datoteka, koja predstavlja prezentacioni deo, ostaje sa istim imenom, dok se na novu datoteku dodaje sufiks *Container*.

Na ovaj način, ne samo da se preventivno sprečava prezasićenje komponente, već se i olakšava testiranje. Nova komponenta koja se koristi za prezentaciju samo prima i iscrtava podatke. Ovaj obrazac može da napravi veliku razliku kada je u pitanju održivost projekta [1]. Alat ovaj obrazac podržava, ali ga ne nameće korisniku. Korisnik ovaj obrazac može da primeni u alatu, tako što će definisati dve zasebne komponente, i potencijalno ih imenovati po konvenciji. Na ovaj način, prezentaciona komponenta može da se kreira vrlo brzo, ukoliko je markap za komponentu već definisan.

3.2. Komponente višeg reda

Primena koncepta funkcija višeg reda na *React* rezultuje komponentama višeg reda (*Higher-Order Components - HOCs*). *HOC* je funkcija koja prima komponentu kao ulaz i vraća proširenu komponentu kao izlaz. Ideja je proširiti ponašanje komponente bez narušavanja njene izolovane logike. Alat podržava kreiranje komponenti višeg reda, bez ograničenja.

3.3. React hooks

Namena *hooks* jeste da olakša način praćenja stanja unutar jedne komponente, kao i da pruži intuitivniji interfejs za baratanje životnim ciklusom komponente. *React* dolazi sa skupom ugrađenih *hook*-ova koji ovo podržavaju i to su *useState* i *useEffect*. Alat se fokusira na pravilnu primenu ova dva *hook*-a, jer su često dovoljni za interfejse proizvoljne složenosti.

Ugrađeni *React hook*-ovi olakšavaju razvoj novih komponenti, ali korisnik ima opciju da kreira sopstveni. Alat ne podržava direktno kreiranje *hook*-ova, jer se njihovo korišćenje u drugim komponentama razlikuje od uključivanja komponenti.

3.4. Function as a child

Function as a child je obrazac u *React*-u koji se koristi za dinamičko renderovanje sadržaja i omogućava komponentama da kontrolišu način na koji deca bivaju renderovana. U ovom obrascu, komponenta koristi funkciju kao dete, umesto *JSX*-a, što omogućava fleksibilnost u prenošenju podataka između komponenti. Funkcija obično prima parametre koji dolaze iz komponente vlasnika. Primer jeste komponenta koja upravlja stanjem, ili dobavlja podatke preko zahteva ka *API*-ju. Takva komponenta može da koristi ovaj obrazac, i omogući svojim potrošačima da odluče kako će ti podaci biti prikazani.

3.5. Redux

Redux je kontejner stanja za *JavaScript* aplikacije i služi kao šablon i biblioteka za upravljanje stanjima unutar aplikacije [3]. Ovaj mehanizam se realizuje upotrebom događaja, odnosno akcija. *Redux* omogućava upravljanje globalnim stanjima, što smanjuje potrebu za prosleđivanjem podataka među komponentama. Ovo dovodi do čitljivijeg koda, jasnije separacije logike i veće granularnosti komponenti.

Stanje aplikacije se čuva u globalnom objektu koji se naziva *store*. Kreator akcija je jednostavna funkcija koja vraća objekat sa opisom akcije i opciono propratnim sadržajem (*payload*). *Reducer* je funkcija koja prima trenutno stanje aplikacije i objekat akcije, odlučujući kako da ažurira trenutno stanje. *Reducer* funkcije predstavljaju primer čistih funkcija, što je važan aspekt funkcionalnog programiranja. *Reducer* se može posmatrati kao osluškivač koji obrađuje događaje u zavisnosti od tipa akcije. Standardni tok podataka u *Redux* arhitekturi obuhvata sledeće korake:

1. Registrovanje događaja. Može biti bilo koja akcija nad elementom korisničkog interfejsa, kao što je klik na dugme ili promena vrednosti u polju za unos.
2. *Dispatch* akcije. *Event handler* šalje odgovarajuću akciju ka *store*-u.
3. *Reducer* obrađuje akciju. *Reducer* prima akciju zajedno sa trenutnim stanjem i, u zavisnosti od tipa akcije, redefiniše postojeće stanje.
4. Osvežavanje interfejsa. Korisnički interfejs se ažurira novim stanjem.

Glavni problem koji se na ovaj način rešava je *prop-drilling*. *Prop-drilling* se javlja kada komponenta prosleđuje *props* detetu, koje ih zatim bez internog korišćenja prosleđuje svom detetu, što čini kod redundantnim i kompleksnim za održavanje. *Redux* i *useReducer* omogućavaju da svaka komponenta može direktno da pristupi globalnom stanju, bez potrebe za *prop-drilling*-om, čime se rešava problem komunikacije između tzv. *sibling* komponenti.

Za čitanje podataka iz *store*-a koriste se *selector* funkcije, koje se obično definišu u istom fajlu gde se definiše *slice*. *Selector* funkcije omogućavaju komponentama da pristupe delovima globalnog stanja na jednostavan način. *Hook*-ovi *useDispatch* i *useSelector* omogućavaju komponentama da šalju akcije i pristupaju stanju, čime se olakšava rad sa *Redux*-om u *React* aplikacijama.

4. KONFIGURACIJA PROJEKTA

4.1. Create React app

Do 2023. godine standardni način za kreiranje *React* aplikacije je bio *create-react-app (CRA)* alat [4]. Alat je vrlo jednostavan za korišćenje i omogućavao je korisniku da brzo započne sa razvojem svoje klijentske aplikacije. Njegova minimalna postavka i jednostavan proces kreiranja aplikacije je upravo razlog zašto je bio *de facto* standard. Iako nije pružao automatsku podršku za stvari koje su tipične za klijentske aplikacije, poput rutiranja i slanja zahteva ka *API*-ju, dolazio je sa konfigurisanim *webpack*-om [5] i *Babel*-om [6]. Otuda, iako na prvi pogled konfiguracija deluje jednostavno, dolazio je sa dosta zavisnosti, i malim prostorom za unapređenje.

Performanse su predstavljale najveći problem, jer su moderniji alati poput *Vite* [7] i radni okviri poput *Next.js* [8], dolazili sa malo više koraka u procesu kreiranja projekta, ali su pružali drastično bolje performanse [1].

4.2. Vite

Vite ima brže vreme pokretanja servera jer dolazi sa sopstvenim bundlerom i ne koristi *webpack* kao *CRA* [7]. Pruža direktnu podršku za rad sa *TypeScript*-om i bolju podršku za upravljanje zavisnostima (*dependency management*). Pored toga, dimenzije projekta kreiranog pomoću *Vite* zahtevaju 80% manje memorije od onih koji su kreirani sa *CRA* [7]. *Vite* prilikom pokretanja izvršava *dependency pre-building* i za to koristi *esbuild* koji je napisan u *Go* programskom jeziku i bundluje zavisnosti od deset do sto puta brže od *JavaScript* bundlera [7].

4.3. Stilizovanje

Od modernih aplikacija se očekuje mnogo kada je reč o stilovima i izgledu. Posledica toga je veliki broj opcija kada je reč o stilovima. Iako rezultantna aplikacija gore navedenih alata, podrazumevano podržava *CSS* (*Cascading Style Sheets*), vrlo često se timovi odlučuju za neku alternativu.

Neke od popularnijih varijanti u React aplikacijama su *TailwindCSS*, *SASS*, i *styled-components*. Prednosti koje se dobijaju nekim od pomenutih varijanti su lokalni stilovi, koji su lakši za održavanje od globalnih. *TailwindCSS* funkcioniše po principu atomskih pravila, gde svaka klasa predstavlja jedno pravilo. Svaka od ovih varijanti zahteva dodatnu konfiguraciju, što alat automatski podržava, tako da se izbor i podrazumevana konfiguracija dobijaju jednim klikom.

4.4. Zavisnosti

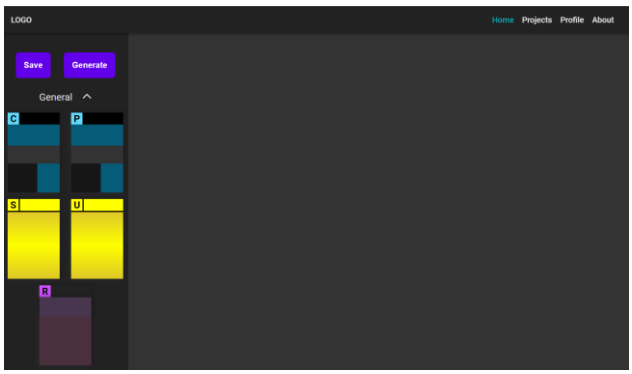
Datoteka koja sadrži meta-podatke o projektu, u *JavaScript* okruženjima je *package.json*. U njemu se nalaze informacije poput autora, verzije, licence, ali i zavisnostima projekta. Posao *package manager*-a je da te zavisnosti nabavi sa odgovarajućeg *npm* registra i smesti ih u specifičan folder *node_modules*.

Alat pored opcije za izbor *package manager*-a pruža korisniku opciju da odmah odabere neke zavisnosti koje planira da koristi u svom projektu. Slično vrlo popularnom *SpringInitializer*-u [9], u sekciji prikazanoj na slici 7. se korisniku nudi opcija pretrage. Sa *NPM* repozitorijuma se dodaju zavisnosti koje odgovaraju korisničkom unosu, i na taj način se smanjuje broj potrebnih koraka kako bi se nova zavisnost dodala u sam projekat.

5. OPIS REŠENJA

5.1. Editor

Grafički editor je implementiran kao *React* aplikacija pomoću *Vite* alata, sa funkcionalnostima poput zumiranja, pomeranja, prevlačenja elemenata, promene dimenzija i interaktivnih dijaloga. Nakon kreiranja novog projekta, korisniku se prikazuje interfejs sa mogućnostima za dodavanje elemenata na radnu površinu.
Klikom na element otvara se komponenta "fioka" (*Drawer*) koja omogućava detaljan opis elementa u zavisnosti od njegovog tipa. Na slici 5.1. prikazana je prazna radna površina editora.



Slika 5.1. Radna površina editora

Komponente, osnovne jedinice React aplikacija, korisnik može detaljno opisati kroz sedam koraka: definisanje imena i putanje, *props*-a, stanja, efekata, akcija, markapa (*JSX*) i vezama sa drugim komponentama. U okviru alata, *props*-i se unose uz validaciju da su jedinstveni i ispravno imenovani. Stanje komponente se definiše pomoću *useState hook*-a, i alat forsira konvencije imenovanja.
Efekti se definišu pomoću *useEffect hook*-a, omogućujući korisniku da postavi akcije koje se izvršavaju tokom životnog ciklusa komponente. Alat pomaže korisniku da automatski kreira i memoizuje *callback* funkcije pomoću *useCallback hook*-a. Akcije predstavljaju funkcije koje se pozivaju unutar *lifecycle hook*-ova ili kao obradivači događaja.
Na kraju, korisnik može definisati markap (*JSX*) koji je povratna vrednost komponente, što omogućava vizuelni prikaz na stranici. Takođe, alat pruža podršku za povezivanje komponente sa drugim delovima sistema, omogućavajući kreiranje složenih i modularnih aplikacija.
Pored komponenti, korisnik može da definiše i stranice, servise, *util* datoteke i *Redux slice*. Stranica predstavlja resurs koji je skoro identičan komponenti. Razlikuje se po tome što će se stranica automatski dodati u ruter *React SPA*, i nakon pokretanja generisanog koda, biti dostupna na definisanoj ruti.
Servisne datoteke sadrže funkcije koje izvršavaju *HTTP* zahtev pomoću *axios* biblioteke i vraćaju *HTTP odgovor*. Funkcija u servisnoj datoteci je klasična *JavaScript* funkcija koju definišu njen naziv i parametri. *Util* funkcije su takođe klasične *JavaScript* funkcije, ali im je namena drugačija od servisnih. Umesto da barataju *HTTP* zahtevima *util* funkcije su obično čiste funkcije, koje izvršavaju logiku koja nije vezana za pojedinačnu komponentu.
Poslednji element koji korisnik može da doda na radnu površinu predstavlja *Redux slice*. Biblioteke koje se automatski dodaju u projekat prilikom generisanja koda, ukoliko se koristi *Redux* slice, su *react-redux* i *reduxjs/toolkit*. Ove biblioteke olakšavaju implementaciju *Redux* toka podataka u aplikaciji koristeći *hook*-ove, što se uklapa u ostatak *React* ekosistema. *Redux Toolkit* je posebno značajan jer nudi unapređeni model za organizaciju koda i rešava problem prevelike količine *boilerplate* koda prilikom konfigurisanja *Redux*-a. Iako je *Redux* izuzetno moćan, jedna od njegovih mana jeste da korišćenje *react-redux* biblioteke ne pruža elegantan način za definisanje *slice*-a, tako da je logika centralizovana u jednom fajlu. *Hook*-ovi koji se koriste za

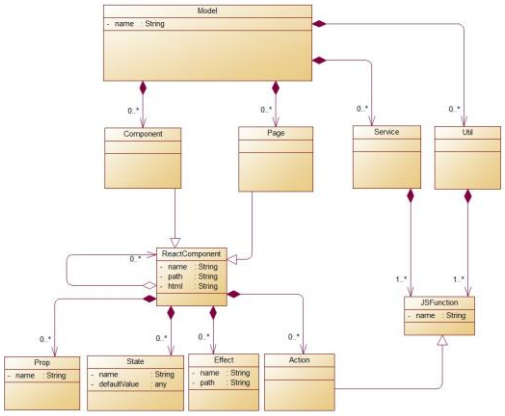
čitanje iz *store*-a i *dispatch*-ovanje akcija, se u komponentama uvoze direktno iz *react-redux* biblioteke.

5.2. Server

Server je implementiran korišćenjem *Express* [10] radnog okvira za *Node.js* [11], a kao baza podataka koristi se *MongoDB* [12]. *MongoDB* je vodeća *NoSQL* baza podataka koja čuva podatke u obliku *JSON* dokumenata, i omogućuje fleksibilnost i horizontalnu skalabilnost. *Express* omogućava postavljanje *RESTful API*-ja koji služi za komunikaciju sa klijentskom aplikacijom (editorom) i bazom podataka. Na ovaj način server obezbeđuje sve potrebne podatke editoru, kao i mogućnost skladištenja i ažuriranja modela projekata.

5.3. Generator koda

Generator koda je ključna komponenta koja koristi *template engine* za generisanje izlaznih datoteka na osnovu definisanih modela. Kao *template engine* koristi se *Nunjucks* [13], koji omogućava procesiranje unapred definisanih šablona sa podacima iz modela. *Nunjucks* pruža fleksibilnost u dizajniranju šablona, omogućavajući jednostavno uključivanje dinamičkih podataka u generisani kod. Generator prihvata šablone i podatke, obrađuje ih pomoću *Nunjucks engine*-a i proizvodi izlazne datoteke koje su deo gotovog projekta. Jedan od problema kod dizajniranja generatora koda jeste pronalaženje minimalnog skupa podataka takvog da se obezbedi funkcionalna softverska komponenta koja se može izmeniti, ali takođe direktno iskoristiti. Jedna od prednosti upotrebe *MongoDB* baze podataka je što se u klijentskoj aplikaciji proširenja mogu direktno dodati, i prikazati prilikom učitavanja modela, bez primene migracija. Na slici 5.2. je prikazan klasni dijagram za meta-model na osnovu kojeg radi generator.



Slika 5.2. Dijagram klasa za meta-model izlaznog projekta

6. ZAKLJUČAK

U radu je predstavljen sistem za modelovanje i generisanje *React* projekta. Sistem je sastavljen od serverske strane aplikacije implementirane u *Node*-u, klijentske strane aplikacije implementirane u *React*-y i *Nunjucks* obrađivača šablona za generisanje koda. Opisane su korišćene tehnike i tehnologije, kao i na koji način alat podržava različita okruženja i dizajn obrasce. Korisniku je omogućeno da na osnovu modela dobije izlazni projekat koji se može proširiti, ali i pokrenuti odmah po preuzimanju. Pravci budućeg razvoja:

- Mogućnost proširenja grafičkog editora dodavanjem novih elemenata kao što su *hook*-ovi, koje korisnik može koristiti u procesu modelovanja..
- Uvođenje funkcionalnosti za praćenje promena u generisanom kodu i automatsko ažuriranje formi elemenata u editoru, čime bi se omogućilo kontinuirano generisanje koda bez potrebe za ručnim unosom promena.
- Prikaz finalnog izgleda komponente u fazi modelovanja, kako bi se smanjilo vreme potrebno za izmenu.
- Uvođenje *LLM*-ova u okruženje koji bi mogli pomoći korisnicima da generišu određene servisne, util ili handler funkcije na osnovu slobodnog teksta, uz mogućnost pregleda i odlučivanja o zadržavanju generisanog koda.

7. LITERATURA

[1] Carlos Santana Roldán. React 18 Design Patterns and Best Practices: Design, build, and deploy production-ready web applications with React by leveraging industry-best practices, 4th Edition.Packt Publishing, 2023. ISBN 978-1803233109

[2] React, https://reactjs.org/, na mreži, pristup: septembar 2024

[3] Redux, https://redux.js.org/ , na mreži, pristup: septembar 2024

[4] Create React app, https://reactjs.org/docs/create-a-new-react-app.html, na mreži, pristup: septembar 2024

[5] Webpack, https://webpack.js.org/, na mreži, pristup: septembar 2024

[6] Babel, https://babeljs.io/, na mreži, pristup: septembar 2024

[7] Vite, https://vitejs.dev/, na mreži, pristup: septembar 2024

[8] Next.js, https://nextjs.org/, na mreži, pristup: septembar 2024

[9] SpringInitializr, https://start.spring.io/, na mreži, pristup: septembar 2024

[10] Express, https://expressjs.com/, na mreži, pristup: septembar 2024

[11] Node.js, https://nodejs.org/en, na mreži, pristup: septembar 2024

[12] MongoDB, https://www.mongodb.com/, na mreži, pristup: septembar 2024

[13] Nunjucks, https://mozilla.github.io/nunjucks/, na mreži, pristup: septembar 2024

Kratka biografija:

Filip Volarić rođen je 25. septembra 1999. godine u Rumi. Pohađao je OŠ "Veljko Dugošević" koju je završio 2014. godine kao nosilac diplome "Vuk Karadžić". Iste godine upisuje Mitrovačku gimnaziju, smer za obdarene učenike u računarskoj gimnaziji. Gimnaziju je završio 2018. godine. Iste godine upisuje Fakultet tehničkih nauka, smer Softversko inženjerstvo i informacione tehnologije. Diplomirao je 2022. godine i stekao zvanje diplomirani inženjer elektrotehnike i računarstva.