Izrada CRUD web aplikacija pomoću React-a i REDUX-a

Gelić, Edita

Sveučilište
Sjever

Završni rad br. 836/MM/2023

Izrada CRUD web aplikacije pomoću React-a i REDUX-a

Edita Gelić, 0336046021

Varaždin, rujan 2023. godine

Sveučilište
Sjever

Odjel za Multimediju, oblikovanje i primjenu

Završni rad br. 836/MM/2023

Izrada CRUD web aplikacije pomoću React-a i REDUX-a

Student

Edita Gelić, 0336046021

Mentor

mr.sc Vladimir Stanisavljević, dipl, ing.

Varaždin, rujan 2023. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR-42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL: Odjel za multimediju

STUDIJ: preddiplomski stručni studij Multimedija, oblikovanje i primjena

PRISTUPNIK: Edita Gelić | MBAG: 0336046021

DATUM: 21.08.2023 | KOLEGIJ: Programski alati 3

NASLOV RADA: Izrada CRUD web aplikacija pomoću React-a i REDUX-a

NASLOV RADA NA ENGL. JEZIKU: CRUD Web Application Development with React and Redux

MENTOR: Vladimir Stanisavljević | ZVANJE: viši predavač

ČLANOVI POVJERENSTVA:

1. doc.dr.sc. Andrija Bernik - predsjednik povjerenstva
2. pred. Dražen Crnić, dipl.ing. - član povjerenstva
3. mr.sc. Vladimir Stanisavljević, v.pred. - mentor
4. doc.dr. sc. Domagoj Frank - zamjenski član
5.

Zadatak završnog rada

BROJ: 836/MM/2023

OPIS:

Većina dinamičkih web aplikacija treba rad s bazom. Osnovne operacije s podacima u bazi obično se označavaju skraćenicom CRUD (od engl. Create-Read-Update-Delete). Njih treba moći koristiti i pri izradi web aplikacija pomoću React(.js) razvojnog okvira što se olakšano postiže s dodatnim komponentama poput Redux-a.

U ovom radu potrebno je:
* opisati osnove rada s bazama podataka u sklopu CRUD paradigme,
* opisati povijesni razvoj i osnove React programske zbirke te njezinu primjenu za izradu web aplikacija,
* analizirati prednosti i ograničenja React-a i Redux-a i usporediti ih sa sličnim tehnologijama,
* detaljno opisati dostupne komponente i osnovnu proceduru izrade aplikacija pomoću odabranih tehnologija,
* osmisliti i oblikovati prilzvoljnu aplikaciju na kojoj će biti demonstrirane glavne mogućnosti rada razvojnih okvira s bazama podataka.

Detaljno dokumentirati sve korištene tehnologije i izrađeni programski kod te opisati stečena iskustva i postignute rezultate.

ZADATAK URUČEN: 20.08.2023 | POTPIS MENTORA: Vladi St

SVEUČILIŠTE SJEVER

Predgovor

Želim se zahvaliti obitelji i prijateljima koji su mi bili podrška tijekom studiranja prethodne tri godine preddiplomskog studija. Njihova podrška i motivacija tijekom pisanja ovog završnog rada bili su neizmjerno važni za mene. Veliku zahvalu dugujem i svom mentoru, mr. sc. Vladimiru Stanisavljeviću, na njegovoj strpljivosti, susretljivosti i pomoći tijekom izrade ovog rada.

Sažetak

Ovaj rad istražuje primjenu JavaScript razvojnih okvira React-a i Reduxa web aplikacija koje koriste CRUD (engl. Create…). Fokusirajući se na tehnologiju React, ovaj rad proučava kako ovi alati zajedno doprinose razvoju modernih web aplikacija. React je popularna JavaScript biblioteka koja omogućava izradu dinamičkih i responzivnih korisničkih sučelja, a Redux je moćan alat za upravljanje stanjem aplikacije. Bootstrap, sa svojom bibliotekom CSS komponenti, čini dizajniranje korisničkog sučelja brzim i jednostavnim.

Prvi dio rada uvodi nas u razumijevanje CRUD operacija, zatim detaljno opisuje React i Redux te njihove prednosti i nedostatke. Bootstrap koristi za stvaranje privlačnog korisničkog sučelja. Ova istraživanja pokazuju da korištenje React-a, Reduxa i Bootstrap-a u kombinaciji pruža snažan alat za razvoj web aplikacija. Njihove prednosti nadilaze nedostatke, omogućavajući programerima da brzo razvijaju visoko kvalitetne web aplikacije koje su funkcionalne i privlačne korisnicima. Ova kombinacija tehnologija ostaje ključna za razvoj modernih web aplikacija u današnjem digitalnom okruženju.

Ključne riječi: React, Redux, Front-end razvoj, JavaScript biblioteka, CRUD aplikacija

Abstract

This paper explores the application of JavaScript development frameworks React and Redux in web applications that utilize CRUD (Create, Read, Update, Delete) operations. Focusing on the React technology, this paper examines how these tools collectively contribute to the development of modern web applications. React is a popular JavaScript library that enables the creation of dynamic and responsive user interfaces, while Redux is a powerful state management tool for applications. Bootstrap, with its CSS component library, makes designing user interfaces quick and straightforward.

The first part of the paper introduces us to understanding CRUD operations, then provides a detailed description of React and Redux, along with their advantages and disadvantages. Bootstrap is used to create appealing user interfaces. This research demonstrates that the use of React, Redux, and Bootstrap in combination provides a robust toolkit for web application development. Their advantages outweigh the drawbacks, allowing developers to rapidly create high-quality web applications that are functional and appealing to users. This technology combination remains crucial for the development of modern web applications in today's digital environment.

Keywords: React, Redux, Front-end development, JavaScript library, CRUD application

Popis korištenih kratica

HTML HyperText Markup Language
Sintaksa za obilježavanje hipertektualnih dokumenata.

CRUD Stvaranje, Čitanje, Ažuriranje i Brisanje
(engl Create, Read, Update, Delete)

JSX JavaScript XML
Mogu se definirati HTML elemente unutar JavaScript koda

DOM Document Object Model - je programski model koji predstavlja strukturu web stranice kao stablo objekata i omogućuje JavaScriptu i drugim programskim jezicima da manipuliraju i interagiraju s elementima na web stranici.

VDOM Virtual Document Object Model - je tehnika koja se koristi u web razvoju kako bi se efikasnije ažurirala web stranica, stvarajući virtualnu kopiju stvarnog DOM-a radi optimizacije performansi.

CSS Cascading Style Sheets - jezik koji se koristi za oblikovanje i stilizaciju web stranica, pomažući definirati izgled elemenata na web stranici, uključujući boje, fontove i raspored

UI Using Interface – Korisničko sučelje

Sadržaj

1. Uvod

U današnjem digitalnom dobu, tehnologija i informacijski sustavi igraju ključnu ulogu u gotovo svim aspektima našeg života. Od komunikacije do poslovanja, digitalni svijet oblikuje naše svakodnevne aktivnosti. U tom kontekstu, razvoj aplikacija i sustava za učinkovito rukovanje podacima postaje sve važniji zadatak. Ovaj završni rad usmjerava se na istraživanje i analizu ključnih aspekata rada u okviru CRUD [1] (*engl. Create, Read, Update, Delete*) paradigme, te primjene React programske zbirke za izradu modernih web aplikacija.

Ključni izazov u informacijskom društvu je učinkovito rukovanje i upravljanje podacima. CRUD paradigma predstavlja osnovnu paradigmu za manipulaciju podacima i predstavlja temelj za mnoge aplikacije. Na drugoj strani, razvoj web aplikacija postao je neizostavan dio modernog poslovanja i interakcije s korisnicima. U tom kontekstu, React programski okvir i Redux biblioteka postaju ključni alati za izradu dinamičkih i interaktivnih web aplikacija.

Cilj ovog završnog rada je pružiti sveobuhvatno razumijevanje CRUD paradigme, React-a i Reduxa te njihovu primjenu u izradi web aplikacija. Konkretni ciljevi uključuju:

- Detaljno istražiti osnove rada CRUD paradigme, analizirati svaku od četiri osnovne operacije CRUD te razumjeti njihovu ulogu u upravljanju podacima.
- Proučiti povijesni razvoj React [2]programske zbirke, osnovne koncepte i komponente te istražiti njegovu primjenu za izradu modernih web aplikacija.
- Analizirati prednosti i ograničenja Reacta [3] i Reduxa [4] u usporedbi sa sličnim tehnologijama, poput Angulara [5] i Vue.js. [6]
- Detaljno opisati dostupne komponente i osnovne korake u izradi web aplikacija pomoću Reacta i Reduxa, uključujući upravljanje stanjem.
- Osmisliti i oblikovati proizvoljnu web aplikaciju kao primjer koji će demonstrirati glavne mogućnosti rada razvojnih okvira, te principa CRUD paradigme.

Ovaj rad koristi se različitim metodama istraživanja, uključujući analizu literature, primjenu praktičnih primjera i eksperimentiranje s izradom web aplikacije. Rad je strukturiran kako bi omogućio postupno razumijevanje i primjenu obrađenih tema. Nakon uvoda, slijede poglavlja posvećena CRUD funkcijama i povijesti Reacta, analizi prednosti i ograničenja, te detaljnom opisu dostupnih komponenata i postupku izrade web aplikacija. Na kraju, prikazat će se proizvoljna web aplikacija koja će poslužiti kao demonstracija korištenja Reacta i Reduxa u stvarnom okruženju.

U idućim dijelovima ovog rada, detaljno ćemo istražiti i analizirati navedene teme kako bismo stekli dublje razumijevanje rada s bazama podataka unutar CRUD paradigme i primjene Reacta i Reduxa u izradi web aplikacija.

2. CRUD

CRUD paradigma, često nazvana i "osnovnim operacijama", predstavlja ključni koncept u razvoju softverskih aplikacija i upravljanju podacima. Skraćenica CRUD predstavlja četiri fundamentalne operacije koje se izvršavaju nad podacima: stvaranje (*engl. Create*), čitanje (*engl. Read*), ažuriranje (*engl. Update*) i brisanje (*engl. Delete*)[1]. Ova paradigma je osnova za efikasno i organizirano rukovanje podacima u aplikacijama, omogućavajući korisnicima i programerima da manipuliraju informacijama na jasan i dosljedan način. Implementacija CRUD operacija je ključna u razvoju aplikacija koje zahtijevaju interakciju sa bazama podataka, čuvanje i ažuriranje informacija, kao i pristup istim. Bez ove paradigme, upravljanje podacima može postati neefikasno i teško održivo.

Svaka aplikacija koja koristi podatke mora pružiti ove četiri osnovne funkcionalnosti kako bi korisnicima omogućila potpuno upravljanje podacima. CRUD funkcionalnosti se obično integriraju u korisničko sučelje aplikacije, tako da korisnici mogu lako pristupiti i upravljati podacima. Koncept CRUD-a se koristi u razvoju bilo koje vrste aplikacija koje upravljaju podacima, kao što su aplikacije za upravljanje inventarom, aplikacije za upravljanje korisnicima, aplikacije za praćenje narudžbi i mnoge druge.

2.1. Stvaranje

Stvaranje (*engl. Create*) predstavlja operaciju dodavanja novog zapisa u bazu podataka ili drugu vrstu spremišta podataka. Budući da se ovaj završni rad bavi reduxom, podaci se pohranjuju u Redux Store [7], koji je centralno spremište za stanje (*engl. state*) aplikacije. Redux Store je objekt koji sadrži trenutno stanje aplikacije, a on se održava i ažurira pomoću Redux akcija (*engl. actions*) i reduktora (*engl. reducers*).

Ova funkcionalnost se koristi kada korisnik želi unijeti nove podatke u aplikaciju, kao što su novi korisnički računi, nove narudžbe, nove proizvode itd. Kada podatke predajemo putem obrazaca, zahtjev za objavu se šalje našem API-ju i podaci će biti pohranjeni u bazi podataka. HTTP protokol koji implementira operaciju stvaranja je POST metoda. [1] Primjer: ruta za GET zahtjev je (/appointments/new).

2.2. Čitanje

Čitanje (engl. Read) omogućava korisnicima da pregledaju postojeće podatke u aplikaciji. Ova funkcionalnost se koristi kada korisnik želi pregledati podatke, kao što su popisi proizvoda, detalji korisničkog računa, povijest narudžbi itd. Dakle, unos može biti bilo što, od korisničkih

informacija do objava na društvenim mrežama i drugim podacima. HTTP protokol koji implementira operaciju čitanja je GET metoda [1]. Primjer: ruta za GET zahtjev je (/appointments).

2.3. Ažuriranje

Ažuriranje (engl. Update) omogućava korisnicima da ažuriraju postojeće podatke u aplikaciji. Ova funkcionalnost se koristi kada korisnik želi promijeniti podatke koji su već pohranjeni, kao što su promjene adrese za dostavu, ažuriranje cijena proizvoda, promjene osobnih podataka itd. PUT je HTTP protokol koji se može koristiti za implementaciju ažuriranja, ovisno o tome što nam je potrebno. [1] PUT bi trebao biti korišten kada se želi potpuno ažurirati unos.

Primjer: ruta za PUT zahtjev je (/appointments/:id). Id u ruti određuje koji unos u zapisu treba biti ažuriran.

2.4. Brisanje

Brisanje(engl. Delete) omogućava korisnicima da uklone postojeće podatke iz aplikacije. Ova funkcionalnost se koristi kada korisnik želi ukloniti podatke koji su više nevažni ili nepotrebni, kao što su izbrisani korisnički računi, izbrisani proizvodi ili otkazane narudžbe. DELETE je HTTP protokol koji se koristi za implementaciju operacije brisanja. [1] Za brisanje bilo kojeg zapisa, svaki unos ima jedinstveni identifikator id, a id u zahtjevu u nastavku identificira određeni zapis koji treba ukloniti iz baze podataka. Primjer: ruta za DELETE zahtjev je (/appointments/:id)

3. React js

React.js [3] je popularna i moćna JavaScript biblioteka za izradu korisničkih sučelja u web aplikacijama. React, razvijen od strane Facebook-a, brzo je postao ključna tehnologija u razvoju modernih interaktivnih web stranica i aplikacija.. U uvodu ćemo istražiti osnovne koncepte i karakteristike React-a, kao i njegov utjecaj na razvoj web aplikacija..

3.1. Povijest

React je razvijan od strane Facebook-a i počeo je kao interni projekt 2011. godine. Tijekom vremena, postao je jedan od najutjecajnijih i najpopularnijih JavaScript alata za izradu korisničkih sučelja. Evo osnovnih događaja u povijesti React-a: [4]

1. **Počeci (2011-2012**):
- React je prvotno razvijen od strane Jordana Walkea, inženjera u Facebook-u, kako bi se nosio s problemima performansi i skalabilnosti u njihovom rastućem broju korisničkih sučelja.
- Prvi naziv za ovaj projekt bio je "FaxJS".
2. **Prva Javna Objava (2013**):
- React je prvi put predstavljen javnosti na konferenciji Facebook-a pod nazivom "JSConf US" 2013. godine. Nakon toga, Facebook je objavio React kao open-source projekt.
3. **Virtual DOM (2013**):
- Ključna inovacija React-a je bila uvođenje Virtual DOM-a, tehnike koja omogućava brže ažuriranje korisničkih sučelja putem efikasnog upravljanja promjenama u virtualnom DOM-u.
4. **Flux Arhitektura (2014):**
- Flux je arhitekturni obrazac koji je Facebook razvio kao komplement React-u za upravljanje stanjem u aplikacijama. Flux je kasnije evoluirao u Redux, popularnu biblioteku za upravljanje stanjem u React aplikacijama.
5. **React Native (2015):**
- Facebook je predstavio React Native, tehnologiju koja omogućava razvoj mobilnih aplikacija koristeći React koncepte i JavaScript. Ovo je otvorilo vrata za razvoj cross-platform mobilnih aplikacija.

6. **React Fiber (2017):**

- React Fiber je unapređenje React-ovog jezgre, koje je omogućilo bolju kontrolu nad procesom renderiranja i bolje rukovanje prekidima.

7. **React Hooks (2018):**

- U verziji 16.8, React je uveo kuke, koji omogućavaju funkcionalnim komponentama da upravljaju stanjem i životnim ciklusom, dosad ograničenim na klasne komponente.

8. **React 17 (2020 - 2021):**

    - React 17 objavljen je kao verzija s naglaskom na bolju kompatibilnost između različitih verzija Reacta. Ova verzija nije donijela mnogo novih značajki, već se usmjerila na poboljšanje razmjene komponenata.
    - **React Server Components (eksperimentalno):** U 2021. godini, React tim predstavio je eksperimentalnu značajku nazvanu React Server Components. Ovo je obećavajuća tehnologija koja omogućava renderiranje komponenata na poslužiteljskoj strani za poboljšanje brzine aplikacija.

9. **Stalni Razvoj i Popularnost (2023):**

    - React je i dalje u stalnom razvoju s redovitim izdanjima i velikom zajednicom razvojnih inženjera. Postao je popularan ne samo u Facebook-u, već i u mnogim drugim kompanijama širom svijeta.
    - U 2023. godini očekuje se daljnji razvoj Reacta s fokusom na poboljšanje performansi, sigurnosti i razvoja ekosustava. Nova rješenja i alati mogli bi se pojaviti kako bi se olakšao rad s Reactom.

React je ostavio dubok trag na razvoj web aplikacija, promovirajući komponentnu arhitekturu, jednostavnost i efikasnost u razvoju korisničkih sučelja, te je ostao jedna od najtraženijih tehnologija za frontend razvoj.

3.2. Virtualni DOM i komponente

Virtualni DOM i komponente su dva ključna koncepta u React.js, popularnoj JavaScript biblioteci za izradu korisničkih sučelja u web aplikacijama. Uvod u VDOM i komponente u React-u pomaže razumjeti kako React radi iza platna i kako se gradi reaktivno i efikasno korisničko sučelje u modernim web aplikacijama. U nastavku će biti navedene i opisane tablice tipičnih komponenti koje se koriste u react aplikacijama.

Komponenta	Opis
„App“	Glavna komponenta koja obuhvaća cijelu aplikaciju
„Header“	Komponenta za prikaz gornjeg dijela aplikacije.
„Footer“	Komponenta za prikaz donjeg dijela aplikacije.
„Sidebar“	Komponenta za prikaz bočne trake (sidebar-a).
„Button“	Komponenta za prikaz gumba.
„Input“	Komponenta za unos teksta.
„Form“	Komponenta za obrazac (formular).
„List“	Komponenta za prikaz liste stavki.
„ListItem“	Komponenta za prikaz pojedinačne stavke u listi.

TABLICA 1 Prikaz pregleda komponenti

	Komponente za navigaciju	Svojstva	Funkcije
1.	Header	Može prihvatiti naslov ili logotip aplikacije kao svojstvo.	Može sadržavati navigacijske poveznice ili funkcije za prijavu/odjavu.
2.	Sidebar	Može prihvatiti stavke za navigaciju ili filtre kao svojstva.	Može sadržavati funkcije za filtriranje ili navigaciju kroz aplikaciju.

TABLICA 2 Prikaz komponenti za navigaciju

	Komponente za unos podataka	Svojstva	Funkcije
1.	Input	Može prihvatiti trenutnu vrijednost unosa, tip unosa (tekst, broj, itd.), i funkciju za ažuriranje vrijednosti unosa.	Često će sadržavati funkciju za praćenje promjena unosa.
2.	Form	Može prihvatiti definiciju obrazaca, kao i funkciju za rukovanje podacima unesenim u obrazac.	Obično će sadržavati funkcije za validaciju i slanje podataka.

TABLICA 3 Prikaz komponenti za unos podataka

	Komponente za prikaz sadržaja	Svojstva	Funkcije
1.	List	Može prihvatiti listu stavki za prikaz.	Može sadržavati funkcije za dodavanje, brisanje ili uređivanje stavki u listi
2.	ListItem	Može prihvatiti informacije o pojedinačnoj stavci, kao što su naslov, opis, i slika.	Može sadržavati funkcije za dodatne akcije na stavkama, kao što su brisanje ili uređivanje.

TABLICA 4 Prikaz komponenti za prikaz sadržaja

	Komponente za interakciju	Svojstva	Funkcije
1.	Button	Može prihvatiti tekst na dugmetu, stilove ili funkciju za obradu klika kao svojstvo.	Obično će imati funkciju za rukovanje klika ili događaja.
2.	Modal	Može prihvatiti sadržaj i vidljivost modalnog prozora.	Obično će imati funkciju za otvaranje, zatvaranje i rukovanje događajima u modalnom prozoru.

TABLICA 5 Prikaz komponenti za interakciju

	Komponente za strukturu sadržaja	Svojstva	Funkcije
1.	App	Može primati globalne konfiguracijske informacije koje će biti dostupne svim podkomponentama.	Može sadržavati funkcije za upravljanje globalnim stanjem i akcijama u aplikaciji.
2.	Footer	Može prihvatiti informacije o autorskim pravima ili kontakt informacije kao svojstvo.	Obično ne zahtijeva posebne funkcije osim prikaza statičkog sadržaja

TABLICA 6 Prikaz komponenti za strukturu sadržaja

Ovo su samo primjeri tabele i grupe komponenti za React.js aplikaciju. Svaka komponenta može imati svoje specifične potrebe u vezi sa svojstvima i funkcijama, ovisno o njenoj svrsi u

aplikaciji. Ovo su samo općenite smjernice koje pomažu pri projektiranju React komponenata. U nastavku će biti opisano kako se komponente grade i koje su to komponente koje se najviše koriste.

3.2.1. Virtualni dom

Virtualni DOM (engl. Virtual Document Object Model) [7] je ključna komponenta React.js biblioteke koja igra centralnu ulogu u brzini i efikasnosti ažuriranja korisničkog sučelja. VDOM je apstraktno stablo u memoriji koje predstavlja stvarni DOM (engl. Document Object Model) web stranice ili aplikacije. React koristi VDOM kako bi efikasno ažurirao korisničko sučelje, smanjujući nepotrebno ponovno iscrtavanje i manipulaciju stvarnim DOM-om.

U React-u Virtualni DOM radi tako da prilikom prvog renderiranja komponente ili aplikacije, React stvara VDOM za cijelo korisničko sučelje. Kada se dogodi promjena stanja u aplikaciji (npr. korisnički unos, podaci iz API-ja itd.), React stvara novi VDOM koji predstavlja očekivani rezultat tog ažuriranja. React zatim uspoređuje stari i novi VDOM kako bi otkrio razlike između njih. Na temelju razlika između Virtualnih DOM-ova, React generira minimalni skup operacija koje su potrebne za ažuriranje stvarnog DOM-a. Ovo se radi kako bi se minimizirala potrošnja resursa i vrijeme potrebno za ažuriranje korisničkog sučelja. Konačno, React primjenjuje ove operacije na stvarni DOM, ažurirajući samo dijelove koji su stvarno promijenjeni.

Prednosti VDOM-a u React-u su te da omogućava brže ažuriranje korisničkog sučelja, jer minimizira manipulaciju stvarnim DOM-om. Osigurava dosljednost između različitih web preglednika, smanjujući probleme s kompatibilnošću. Razvojni proces postaje jednostavniji jer programeri mogu razmišljati o ažuriranju VDOM-a umjesto izravnog rukovanja stvarnim DOM-om.

3.2.2. Komponente

Komponente su ključni koncept u React.js okviru za izgradnju korisničkih sučelja. One predstavljaju samodostatne dijelove korisničkog sučelja koji se mogu ponovno koristiti i kombinirati kako bi se izgradila složena funkcionalnost. Svaka komponenta u Reactu ima svoju definiciju koja može sadržavati HTML kod, JavaScript logiku i CSS stilove. Komponente se mogu smatrati "blokovima izgradnje" koje zajedno grade korisničko sučelje. Komponente imaju svoje stanje (*engl. state*) i mogućnost primanja i obrade podataka putem svojstava (*engl. props*). Stanje je interna memorija komponente koja se može ažurirati i utjecati na prikazani sadržaj. S druge strane, svojstva su ulazni podaci koje komponenta prima od roditeljskih komponenti.

U React-u, postoje dvije osnovne vrste komponenata [3]: funkcionalne komponente (*engl function components*) i klasne komponente (*engl. class components*). Ove komponente se koriste za izgradnju korisničkih sučelja i imaju različite načine definiranja i upotrebe.

Funkcijske komponente su najjednostavniji oblik komponenti u Reactu. One su obične JavaScript funkcije koje prihvaćaju ulazne podatke (*engl. props*) i vraćaju JSX (*JavaScript XML*) koji opisuje kako će se komponenta prikazati. Funkcijske komponente nemaju vlastito stanje (engl. state) i ne koriste metode životnog ciklusa. One se često koriste za jednostavne ili ponovno upotrebljive dijelove [6] korisničkog sučelja.

Primjer:

```
import React from 'react';

function Greeting(props) {
 return <h1>Hello, {props.name}!</h1>;
}

export default Greeting;
```

Programski kod 1 Primjer funkcijske komponente

Klasne komponente su stariji način definiranja komponenata u Reactu. Imaju vlastito stanje (*engl. state*) i koriste metode životnog ciklusa za manipulaciju stanjem i prikazivanje JSX-a. Klasne komponente se često koriste za složene komponente s kompleksnom logikom i stanjem. [6]

Primjer klasne komponente:

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

class Counter extends Component {
  constructor(props) {
      super(props);
      this.state = { count: 0 };
  }
  handleClick() {
      this.setState({ count: this.state.count + 1 });
  }
  render() {
      return (
      <div>
      <p>Count: {this.state.count}</p>
      <button onClick={() => this.handleClick()}>Increment</button>
      </div>
      );
  }
}
export default Counter;
```

Programski kod 2 Primjer klasne komponente brojača

U ovom primjeru, **Counter** komponenta je klasna komponenta koja koristi stanje za praćenje broja klikova na gumb i reagira na događaje pomoću metode **handleClick()**.

Korištenje komponenti u React-u omogućava stvaranje složenih aplikacija s mnogo funkcionalnosti i mogućnosti, a zbog modularnog dizajna, komponente se mogu ponovno koristiti na drugim mjestima u aplikaciji ili čak u drugim projektima.

Prednost korištenja komponenti je u njihovoj modularnosti, fleksibilnosti i ponovnom korištenju koda. Komponente omogućuju programerima da razbiju korisničko sučelje na manje dijelove, što olakšava razvoj, održavanje i testiranje aplikacija.

React pruža mogućnost kreiranja funkcionalnih komponenata i klasnih komponenata. Funkcionalne komponente su jednostavnije i fokusirane samo na prikazivanje sadržaja, dok klasne komponente omogućuju dodatne funkcionalnosti, poput upravljanja stanjem.

3.3. Stanje i svojstva

Stanje (engl. *State*) i svojstvo (engl. *props*) su ključni koncepti u React.js koji omogućavaju komponentama da komuniciraju i upravljaju svojim podacima. Kombinacija *state* i *props* omogućava React komponentama da budu dinamične, ponovno upotrebljive i prilagodljive za različite scenarije upotrebe.

3.3.1. Stanje

Stanje (engl. *state*) je ključna koncepcija u React.js, popularnoj JavaScript biblioteci za izradu korisničkih sučelja u web aplikacijama. Ovaj koncept omogućava komponentama da čuvaju i upravljaju dinamičkim podacima koji se mogu promijeniti tokom vremena. Kroz upotrebu **state**, React omogućava izgradnju interaktivnih i dinamičkih korisničkih sučelja, gdje se prikazi automatski ažuriraju kako bi odražavali promjene u stanju aplikacije. U uvodnom dijelu ćemo istražiti osnovne principe i upotrebu **state** u React komponentama. [7]

Stanje (engl. state) u React.js predstavlja interni objekt komponente koji sadrži podatke koji se mogu promijeniti tokom vremena i utječu na prikaz korisničkog sučelja komponente. Ovo stanje može biti dinamičko, što omogućava komponenti da se automatski ažurira kada se promijeni njezino stanje. Evo jednostavnog objašnjenja i primjera korištenja **state**:

Objašnjenje state-a:

- **state** je poseban objekt u svakoj React komponenti koji čuva podatke vezane za tu komponentu.

- Promjena **state**-a pokreće ponovno iscrtavanje (re-render) komponente, ažurirajući prikaz na temelju novog stanja.
- Komponenta može imati više različitih **state**-ova, svaki čuvajući različite informacije.

Primjer korištenja state-a:

```
import React, { Component } from 'react';
class Brojač extends Component {
  constructor(props) {
      super(props);
      // Inicijalizacija stanja
      this.state = {
      broj: 0,
      };
  }
  // Metoda za povećanje broja
  povećajBroj = () => {
      this.setState({ broj: this.state.broj + 1 });
  };
  render() {
      return (
      <div>
      <p>Trenutni broj: {this.state.broj}</p>
      <button onClick={this.povećajBroj}>Povećaj</button>
      </div>
      );
  }
}
export default Brojač;
```

Programski kod 3 Primjer korištenja stanja

U ovom primjeru, **Brojač** komponenta ima **state** objekt koji sadrži **broj**. Klikom na gumb "Povećaj," **povećajBroj** metoda ažurira **state**, a React automatski osvježava prikaz broja na korisničkom sučelju.

3.3.2. Svojstvo

Props ili svojstva su ključni koncept u React.js koji omogućava komunikaciju između komponenata tako da se podaci prenose iz roditeljske komponente u dječju komponentu. [7] Ovi podaci se šalju putem atributa u JSX-u i koriste se za prilagodbu ponašanja i izgleda komponenata. Važno je napomenuti da su *props* obično nepromjenjivi unutar samih komponenata kako bi se očuvala konzistentnost i predvidljivost ponašanja.

Primjer:

```
import React from 'react';

function Pozdrav(props) {
  return <h1>Dobrodošli, {props.ime}!</h1>;
}
```

```
function App() {
  return <Pozdrav ime="gosti" />;
}
export default App;
```

Programski kod 4 Primjer korištenja svojstva

U ovom primjeru, komponenta **Pozdrav** prima **ime** kao *props* i koristi ga za personaliziranu poruku. Komponenta **App** koristi **Pozdrav** komponentu i prosljeđuje joj **ime** kao svojstvo. Kada se komponenta **App** renderira, prikazat će se poruka "Dobrodošli, gosti!".

3.4. JSX

JSX (JavaScript XML) [8] je jezik za opisivanje korisničkog sučelja u React.js aplikacijama. On omogućava programerima da pišu kod sličan HTML-u unutar JavaScript datoteka, što olakšava izradu korisničkog sučelja. U JSX-u, HTML elementi se mogu napisati unutar JavaScript koda, čineći kodiranje čitljivim i intuitivnim.

Može se koristiti JSX za dinamički prikazivanje podataka tako da ugnježđuje JavaScript izraze unutar JSX-a. Na primjer, možete dinamički generirati tekstove, stilove ili komponente pomoću ovih izraza.

```
const ime = "Pero";
const element = <h1>Dobrodošao, {ime}!</h1>;
```

Programski kod 5 Jednostavni prikaz primjera JSX

JSX se ne izvršava direktno u pregledniku, već se koristi alat za transpilaciju (kao što je Babel) kako bi se pretvorio JSX u čisti JavaScript kod. Ovaj JavaScript kod zatim se izvršava u pregledniku. Ključna komponenta JSX-a je njegova integracija s React.js bibliotekom. React koristi JSX za definiranje komponenata i njihovih prikaza, te za stvaranje i upravljanje VDOM-om, čime se postiže učinkovito ažuriranje korisničkog sučelja u React [8] aplikacijama.

Primjer:

```
import React from 'react';

function Dobrodošao(props) {
  return <h1>Dobrodošli, {props.ime}!</h1>;
}
function App() {
  return <Dobrodošao ime="Pero" />;
}
export default App;

 ;
```

Programski kod 6 Primjer ugniježđena u JSX kodu

U ovom primjeru, koristimo JSX za definiranje dviju React komponenata. Prva komponenta **Dobrodošao** prikazuje personaliziranu poruku dobrodošlice s imenom koje prima putem **props**. Druga komponenta **App** koristi **Dobrodošao** komponentu i prosljeđuje joj ime "Pero" putem **props**. Kada se komponenta **App** renderira, prikazat će se poruka "Dobrodošao, Pero!".

Ovaj primjer ilustrira kako JSX omogućava jednostavno ugnježđivanje JavaScript izraza (u ovom slučaju, **props.ime**) unutar HTML-om sličnog koda kako bi se dinamički generirao prikaz korisničkog sučelja.

Zahvaljujući ovim karakteristikama, JSX je moćan alat koji olakšava izradu modernih web aplikacija.

3.5. Kuke

React kuke (*engl. hooks*) predstavljaju revolucionaran dodatak React biblioteci koji omogućuje programerima korištenje stanja i drugih React mogućnosti unutar funkcionalnih komponenata. Prije nego što su kuke uvedene, programeri su morali koristiti klase kako bi upravljali stanjima i životnim ciklusnim metodama. Međutim, s uvođenjem React kuka u verziji 16.8, funkcionalne komponente [9] su dobile mogućnost upravljanja stanjima i izvođenja sporednih efekata bez potrebe za klasnim komponentama.

Kuke omogućuju kraći i elegantniji način pisanja komponenti jer dozvoljavaju kapsuliranje ponovno iskoristive logike. Programeri mogu podijeliti kompleksne komponente na manje, upravljive funkcije, što olakšava razumijevanje i održavanje koda.

Najčešće korištene ugrađene React kuke su:

- useState,
- useEffect,
- useContext.

Kuka useState omogućuje komponentama da drže i ažuriraju stanje. Primjer:

```
import React, { useState } from 'react';
    function Counter() {
     const [count, setCount] = useState(0);
return (
    <div>
          <p>You clicked {count} times</p>
                <button onClick={() => setCount(count + 1)}>
                      Click me
                </button>
    </div>
);
}
```

Programski kod 7 Primjer useState kuke

Ovaj kod uvozi React i useState kuka iz Reacta. Funkcija Counter je definirana kao funkcionalna komponenta koja prikazuje brojač i gumb za povećavanje brojača.

Nadalje, kuka useEffect omogućuje komponentama da izvrše neku akciju nakon što se renderira. Primjer:

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

function Example() {
  const [count, setCount] = useState(0);
      useEffect(() => {
      document.title = `You clicked ${count} times`;
  });
       return (
      <div>
            <p>You clicked {count} times</p>
                  <button onClick={() => setCount(count + 1)}>
                        Click me
                  </button>
      </div>
  );
}
```

Programski kod 8 Primjer useEffect kuke

Ovaj kod uvozi React, useState i useEffect kuka iz React biblioteke. Funkcija Example je definirana kao funkcionalna komponenta koja prikazuje brojač i gumb za povećavanje brojača. Kuka useState se koristi kako bi se stvorilo stanje unutar funkcionalne komponente, baš kao i u prethodnom primjeru. U ovom slučaju, funkcija useState(0) stvara varijablu 'count' s početnom vrijednosti 0 i funkciju 'setCount' koja se može koristiti za ažuriranje 'count' varijable. Međutim, ovaj primjer također koristi useEffect kuka. useEffect kuka se koristi kako bi se izvršila neka akcija nakon što se komponenta prikaže na zaslonu ili se ažurira. U ovom primjeru, funkcija useEffect se poziva svaki put kada se komponenta ažurira. Funkcija useEffect mijenja naslov dokumenta (document.title) koji se prikazuje u kartici preglednika. Tekst se mijenja kako bi se prikazao broj klikova (count varijabla). Ovo osigurava da se naslov dokumenta ažurira svaki put kada korisnik klikne na gumb i poveća brojač. Na kraju, funkcija Example vraća JSX koji prikazuje brojač i gumb za povećavanje brojača. Tekst koji prikazuje broj klikova (count varijabla) se ažurira svaki put kada korisnik klikne na gumb.

Za kraj slijedi useContext. Ova kuka omogućuje komponentama da pristupe kontekstu koji je definiran na vrhu stabla komponenti. Primjer:

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

const ThemeContext = React.createContext('light');

      function ThemedButton() {
```

```
      const theme = useContext(ThemeContext);
      return (
     <button
style={{background:theme.background,color:theme.foreground }}>
     I am styled by theme context!
     </button>
  );
}
```

Programski kod 9 Primjer useContext kuke

Ovaj kod uvozi React i useContext Hook iz Reacta. Također, stvara se ThemeContext pomoću React.createContext metode koja omogućuje dijeljenje podataka između komponenti koje su dio istog stabla komponenti. Funkcija ThemedButton je definirana kao funkcionalna komponenta koja koristi useContext kuku kako bi dobila pristup ThemeContextu i koristila vrijednosti iz nje. U ovom slučaju, tema (engl. theme) se postavlja na vrijednost koju pruža ThemeContext.

Komponenta ThemedButton vraća JSX koji prikazuje gumb čija se pozadina i boja teksta mijenjaju ovisno o temi koja se prenosi putem ThemeContext-a. Primjećuje se da su boje određene vrijednostima pozadine (engl. background) i teksta (engl. foreground) koje su definirane u objektu "style" i koje se preuzimaju iz vrijednosti teme.

Dakle, ovaj kod demonstrira upotrebu useContext kuka za dobivanje vrijednosti iz ThemeContexta. Komponenta ThemedButton koristi temu iz ThemeContexta kako bi stilizirala prikazani gumb, a tema se može mijenjati iz bilo koje komponente koja dijeli ThemeContext u istom stablu komponenti.

Zahvaljujući React kukama, programeri mogu stvarati moćne i performantne aplikacije, maksimizirajući ponovnu upotrebljivost koda i smanjujući korištenje klasnih komponenti. Ova paradigma značajno je unaprijedila iskustvo razvoja u Reactu, čineći ga preferiranim izborom za moderne web aplikacije.

4. Redux

Redux je JavaScript biblioteka za upravljanje globalnim stanjem aplikacije koju je razvila tvrtka Dan Abramov-a i Andrew Clark-a. Biblioteka je prvi put objavljena 2015. [10] godine kao odgovor na izazove upravljanja stanjem u većim aplikacijama i kompleksnijim korisničkim sučeljima.

Kao što je spomenuto, Redux se koristi za upravljanje globalnim stanjem aplikacije, što znači da se svi podaci pohranjeni u aplikaciji nalaze na jednom mjestu, u tzv. Redux Store-u. Ovo olakšava upravljanje podacima i njihovo dijeljenje između različitih dijelova aplikacije.

Također, Redux se najčešće koristi u kombinaciji s Reactom, ali se može koristiti i s drugim JavaScript bibliotekama i okvirima. Redux se koristi za upravljanje različitim vrstama podataka, poput podataka o korisniku, stanju korisničkog sučelja i drugim vrstama podataka koje je potrebno dijeliti između različitih dijelova aplikacije.

Vrlo je popularan među razvojnim zajednicama i koriste ga mnoge tvrtke, poput Applea, Microsofta, Netflix-a i mnogih drugih.

Podjela u Reduxu:

1. Akcije (*engl. Action*): Objekti koji opisuju događaj ili akciju koju korisnik izvršava u aplikaciji.
2. Reduktor (*engl. Reducer*): Funkcije koje primaju trenutno stanje aplikacije i akciju koja se dogodila te vraćaju novo stanje aplikacije.
3. Store: Objekt koji sadrži sve podatke o aplikaciji i koji se može ažurirati uz pomoć akcija i reducer funkcija.

4.1. Akcije

Kako se stanje ne smije direktno mijenjati, Redux uvodi strogi princip nepromjenjivosti podataka i zahtijeva da se svaka promjena stanja odvija putem akcija (*engl. actions*).Akcije predstavljaju događaje ili promjene koje se žele izvršiti u aplikaciji. One su jednostavni JavaScript objekti koji sadrže informacije o vrsti promjene (tip akcije) i eventualne dodatne podatke potrebne za izvršenje te promjene. [10] Tipično, tip akcije definira se kao niz znakova kako bi se jasno prepoznala svrha akcije (npr. "DODAJ_ELEMENT", "UKLONI_ELEMENT", itd.).

```
    switch(action.type){
         case 'ADD_CONTACT':
             state= [...state, action.payload];
             return state;
         case 'UPDATE_CONTACT':
             const updateState = state.map(contact => contact.id ===
action.payload.id ? action.payload: contact)
```

```
            state= updateState;
            return state;
        case 'DELETE_CONTACT':
            const filterContacts = state.filter(contact => contact.id
!== action.payload && contact);
            state = filterContacts;
            return state;

        default:
            return state;
    }
```

Programski kod 10 Primjer akcije u reduxu

4.2. Reduktor

Reduktor (*engl. reducer*) je ključni koncept u Reduxu, koji se koristi za upravljanje promjenama stanja u aplikaciji. Redux je popularna biblioteka za upravljanje stanjem u JavaScript aplikacijama, a reduktori igraju ključnu ulogu u definiranju kako se stanje aplikacije mijenja u odgovoru na akcije.

Reduktor je čista JavaScript funkcija koja prima dva argumenta - trenutno stanje aplikacije i akciju. Akcija, kao što je prije spomenuto, opisuje promjenu koja se događa i obično sadrži tip akcije (npr. "DODAJ_ELEMENT", "UKLONI_ELEMENT") te dodatne podatke koji opisuju promjenu. U skladu s principom nepromjenjivosti, reduktor ne mijenja izravno postojeće stanje, već koristi informacije iz akcije kako bi izračunao novo stanje aplikacije. To novo stanje se potom vraća iz reduktora i zamjenjuje prethodno stanje u Redux store-u. Reduktori omogućavaju jednostavno praćenje i upravljanje stanjem aplikacije, jer se svaka promjena jasno definira putem akcija, a logika za obradu tih akcija je centralizirana u reduktorima. [11]

```
const contactReducer = (state = initialState, action) => {
    switch(action.type){
        case 'ADD_CONTACT':
            state= [...state, action.payload];
            return state;
        case 'UPDATE_CONTACT':
            const updateState = state.map(contact => contact.id ===
action.payload.id ? action.payload: contact)
            state= updateState;
            return state;
        case 'DELETE_CONTACT':
            const filterContacts = state.filter(contact => contact.id
!== action.payload && contact);
            state = filterContacts;
            return state;

        default:
            return state;
    }
};
```

Programski kod 11 Primjer reduktora u reduxu

Reducer funkcija (contactReducer) je funkcija koja prima dva parametra: trenutno stanje (*engl. state*) i akciju (*engl. action*) koja se primjenjuje na to stanje. Ova funkcija definira kako će se promjene primjeniti na stanje.

- **ADD_CONTACT akcija:** Kada se kreira akcija tipa 'ADD_CONTACT', nova kontakt informacija se dodaje na trenutno stanje. Koristi se *spread* operator (**[...state]**) da bi se napravila kopija trenutnog stanja i dodala nova kontakt informacija (**action.payload**) na kraj tog niza.
- **UPDATE_CONTACT akcija:** Kada se kreira akcija tipa 'UPDATE_CONTACT', trenutno stanje se mapira tako da se pronađe kontakt sa odgovarajućim ID-jem (prema **action.payload.id**) i zamjeni ga novim kontaktom (**action.payload**). Tako se simulira ažuriranje kontakta u stanju.
- **DELETE_CONTACT akcija:** Kada se kreira akcija tipa 'DELETE_CONTACT', koristi se metoda **filter** da bi se iz trenutnog stanja uklonila kontakt informacija koja ima ID jednak **action.payload**. Na ovaj način se simulira brisanje kontakta iz stanja.
- **Default slučaj:** Ukoliko se primjeni akcija koja nije prepoznata, reduktor će jednostavno vratiti trenutno stanje.

4.3. Store

Store je ključni koncept u Reduxu i predstavlja centraliziranu komponentu koja čuva stanje aplikacije. Redux je popularna biblioteka za upravljanje stanjem u JavaScript aplikacijama, a store je mjesto gdje se pohranjuje sveukupno stanje aplikacije u jednom objektu. Stanje aplikacije u Reduxu je nepromjenjivo, što znači da ga ne možemo direktno mijenjati. Umjesto toga, svaka promjena stanja se odvija putem akcija (*engl. actions*) koje se šalju *store-u*. *Store* potom koristi reduktore (*engl. reducers*) kako bi obradio akciju i izračunao novo stanje aplikacije. Komponente u aplikaciji mogu pristupiti trenutnom stanju aplikacije tako što se pretplate na *store*. Kada se stanje promijeni, *store* automatski obavještava sve pretplaćene komponente i omogućuje im da se osvježe s najnovijim podacima. Zahvaljujući *store-u*, Redux pruža centraliziran i predvidljiv način upravljanja stanjem aplikacije, što olakšava razvoj i održavanje složenih aplikacija. [12]

```
const store = createStore(contactReducer);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Provider store={store}>
    <Router>
    <App />
    </Router>
  </Provider>
);
```

Programski kod 12 Primjer store-a u reduxu

Ovaj kod se odnosi na implementaciju Redux store-a u React aplikaciji. Evo objašnjenja svakog dijela koda:

1. **Kreiranje Redux Store-a:** Linija **const store = createStore(contactReducer);** stvara Redux Store. Store je centralno skladište stanja (engl. state) aplikacije. Prvi argument funkcije **createStore** je reducer funkcija (**contactReducer** u ovom slučaju), koja upravlja stanjem aplikacije.
2. **Prikazivanje komponente unutar Providera i Routera:** U slijedećim linijama, metoda **root.render()** se koristi za prikazivanje glavne komponente (**App**) unutar komponenti **Provider** i **Router**.
    - **<Provider store={store}>**: Ova komponenta omotava cijelu aplikaciju i omogućuje komponentama pristup Redux Store-u. Proslijeđuje se **store** kao svojstvo. To znači da sve komponente unutar ovog **Provider**-a mogu koristiti Redux Store.
    - **<Router>**: Ovdje se koristi React Router komponenta za omogućavanje rutiranja unutar aplikacije. To omogućuje aplikaciji da se prebacuje između različitih "ruta" bez potpunog osvježavanja stranice.

<App />: Ovdje se prikazuje glavna komponenta aplikacije, koja je vjerojatno osnovna točka za sve ostale komponente i rute unutar vaše aplikacije

4.4. Povezivanje React-a sa Redux-om

Povezivanje Reacta s Reduxom omogućuje nam efikasno upravljanje stanjem aplikacije unutar React komponenti. Redux je biblioteka za upravljanje stanjem, dok je React popularna biblioteka za izradu korisničkog sučelja. Integracijom Reduxa s React-om, možemo centralizirano upravljati stanjem i olakšati reaktivno osvježavanje komponenti kad god se stanje promijeni.

Kako bismo povezali React s Redux-om, koristimo "react-redux" paket koji nam pruža pomoćne funkcije za povezivanje Redux store-a s React komponentama. Tri ključne značajke React Redux paketa koje olakšavaju integraciju Redux-a s React-om :

- Provider
- useSelector
- useDispatch

Redux nam pruža jasne smjernice za upravljanje stanjem, dok React pruža moćno sučelje za izradu korisničkog iskustva. Povezujući ove dvije biblioteke, dobivamo snažan alat za izradu skalabilnih i efikasnih aplikacija koje se lako održavaju.

4.4.1. Provider

Provider [13] je ključna komponenta u Redux-u koja omogućuje povezivanje Redux *store*-a s React aplikacijom. Ona osigurava da sve komponente unutar aplikacije imaju pristup globalnom stanju koje se nalazi u Redux *store*-u, bez potrebe za ručnim prosljeđivanjem stanja kroz komponente kao svojstva (*props*). Provider je dio "react-redux" paketa i koristi se na najvišoj razini aplikacije (obično oko korijenske komponente) kako bi omotao cijelu aplikaciju. Kroz Provider, definira se koji Redux *store* koristi aplikacija.

```
import { createStore } from 'redux';
import contactReducer from './redux/reducers/contactReducer';
import { Provider } from 'react-redux';

const store = createStore(contactReducer)

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Provider store={store}>
    <Router>
    <App />
    </Router>
  </Provider>
);
```

Programski kod 13 Primjer provider-a

4.4.2. useSelector

Kuka (*engl. hook*) useSelector dostupna je u "react-redux" paketu i koristi se u React funkcionalnim komponentama kako bi se dobilo stanje iz Redux store-a.

useSelector je React Redux kuka koja se koristi kako bi komponente mogle pristupiti dijelovima stanja (*engl. state*) iz globalnog Redux store-a. Ova kuka omogućava komponentama da selektiraju određene podatke iz stanja i koriste ih unutar komponente bez potrebe za pretplatom na cijeli *store*.

Koristeći useSelector, možemo selektivno dohvaćati određene dijelove stanja iz Redux *store*-a i koristiti ih u funkcionalnim komponentama kao što koristimo ostale React kuke.

Prilikom korištenja useSelector [14], također možemo definirati funkciju za usporedbu koja će odrediti treba li osvježiti komponentu kad se stanje promijeni. Ovo omogućuje optimizaciju performansi, jer se komponenta neće osvježavati ako su relevantni dijelovi stanja nepromijenjeni.

```
import {useSelector } from 'react-redux'

const AddContact = () => {
    const contacts = useSelector((state) => state);
…
```

Programski kod 14 Primjer useSelectora

Funkcijska komponenta **const contacts = useSelector((state) => state);**: U ovoj liniji se koristi **useSelector** kako bi se selektiralo stanje iz Redux *store-a*. Funkcija unutar **useSelector** je selektor funkcija koja prima trenutno stanje (*state*) i vraća odabrani dio tog stanja. U ovom slučaju, **state** se vraća nepromijenjen, pa će **contacts** sadržavati cijeli objekt stanja aplikacije.

Ovaj kod zapravo selektira kompletno stanje aplikacije iz Redux *store-a* koristeći **useSelector** i sprema ga u **contacts** varijablu unutar komponente. Ovako selektirane podatke možete koristiti u komponenti za prikaz, manipulaciju i ostale potrebne operacije.

4.4.3. useDispatch

Kuka **useDispatch** koristi se u aplikacijama koje koriste React Redux za upravljanje stanjem. Ova kuka omogućava komponentama da pristupe Redux **dispatch** funkciji koja služi za slanje akcija *reducerima*. Kroz **useDispatch**, komponente mogu inicirati promjene u globalnom stanju aplikacije tako što šalju akcije koje zatim *reduceri* obrađuju. Ovaj koncept omogućava efikasno upravljanje stanjem i ažuriranje korisničkog sučelja u skladu sa promjenama podataka.

Kuka (*engl. hook*) useDispatch nalazi se "react-redux" paketu i koristi se u React funkcionalnim komponentama kako bi se pokretale Redux akcije. Omogućava komponentama da izvršavaju akcije i izmjenjuju stanje u Redux *store*-u.

Kada se useDispatch [14] koristi u funkcionalnoj komponenti, komponenta može pozvati "dispatch" s odgovarajućom akcijom, a Redux će automatski pozvati odgovarajući reduktor kako bi se ažuriralo stanje aplikacije.

```
const AddContact = () => {

    const dispatch = useDispatch();

    const handleSubmit = (e) => {
        e.preventDefault();

        const checkEmail  = contacts.find(contact => contact.email === 
email && contact);

        const checkNumber = contacts.find(
            (contact) => contact.number === parseInt(number)
        );
        const data = {
            id: contacts[contacts.length - 1].id + 1,
            name,
            email,
            number,
        }
        dispatch ({type: 'ADD_CONTACT', payload : data});
        toast.success('Student je dodan uspješno!!!')
    }
```

Programski kod 15 Primjer useDispatch

Korištenjem **useDispatch** kuke, komponenta **AddContact** dobiva pristup Redux **dispatch** funkciji kako bi mogla slati akciju za dodavanje kontakta. Ova akcija će biti obrađena u odgovarajućem reduktoru koji će izvršiti promjene u stanju aplikacije.

Ovaj kod koristi React Redux biblioteku za upravljanje stanjem u aplikaciji. **useDispatch** je React kuka koji omogućava komponentama da pristupe Redux **dispatch** funkciji, koja služi za slanje akcija *reducerima*. Evo detaljnog objašnjenja koda:

1. **Definiranje komponente AddContact:** Funkcija **AddContact** predstavlja React komponentu koja se koristi za dodavanje novog kontakta.
2. **Korištenje useDispatch kuka:** Linija **const dispatch = useDispatch();** koristi **useDispatch** hook kako bi se dobio pristup Redux **dispatch** funkciji. Ova funkcija se koristi za slanje akcija reducerima, što dovodi do promjena u stanju aplikacije.
3. **Definiranje handleSubmit funkcije: handleSubmit** funkcija se izvršava kada korisnik pošalje formu za dodavanje novog kontakta. Evo šta ta funkcija radi:
    - **e.preventDefault();**: Sprječava osnovno ponašanje forme, tj. sprečava osvježavanje stranice prilikom klika na "submit" dugme.
    - **checkEmail** i **checkNumber** provjere da li već postoji kontakt s istom email adresom ili brojem telefona u trenutnom stanju kontakata (**contacts**). Ako postoje, to može biti korisno kako bi se izbjeglo dodavanje duplikata.

- **data** sadrži informacije o novom kontaktu koje će biti poslane akciji za dodavanje (**ADD_CONTACT**). **id** se generira tako da bude jedinstven (veći od najvećeg postojećeg **id**-ja).
- **dispatch**: Poziva Redux **dispatch** funkciju kako bi se poslala nova akcija. U ovom slučaju, šalje se akcija sa tipom **'ADD_CONTACT'** i sa sadržajem **payload** koji predstavlja podatke novog kontakta.
- **toast.success(...)**: Poziva funkciju za prikazivanje obavještenja (toast) kako bi se korisnika informiralo da je kontakt uspješno dodan.

5. Prednosti i nedostaci React-a i Redux-a

React i Redux su dva popularna JavaScript okvira koja se često koriste za razvoj web aplikacija, posebno za izradu korisničkih sučelja. Svaki od njih ima svoje prednosti i nedostatke, te odabir između njih ovisi o specifičnim zahtjevima projekta. U nastavku ćemo istražiti neke od glavnih prednosti i nedostataka Reacta i Reduxa kako bismo pomogli boljem razumijevanju njihove upotrebe i odabiru onog koji najbolje odgovara potrebama korisnika.

Prednosti React-a

Prednosti Reacta su mnogobrojne. Prvo, React omogućuje lakši i brži razvoj korisničkog sučelja. Koristi se komponentna arhitektura, što olakšava organizaciju i ponovnu upotrebu koda. Također, React podržava virtualni DOM, što znači da se promjene na stranici prikazuju brže i efikasnije, čime se poboljšava korisničko iskustvo. Osim toga, React ima veliku zajednicu i mnogo resursa za učenje, što olakšava razvoj web aplikacija.

Nedostaci React-a

Međutim, postoje i nedostaci Reacta. Prvo, učenje Reacta može biti izazovno za početnike, posebno ako nisu upoznati s JavaScriptom. Također, aplikacije izgrađene s Reactom mogu postati kompleksne s rastom projekta, što može otežati održavanje i upravljanje kodom. Konačno, React se fokusira na razvoj korisničkog sučelja, pa za upravljanje stanjem aplikacije često treba koristiti dodatne biblioteke ili alate poput Reduxa.

Prednosti Redux-a

Prednosti Reduxa uključuju bolje upravljanje stanjem aplikacije. Redux omogućuje centralizirano skladište podataka, što olakšava praćenje i upravljanje promjenama u stanju aplikacije. Također, Redux olakšava testiranje aplikacije jer se logika za upravljanje stanjem izdvaja iz komponenata. Osim toga, Redux je dobro dokumentiran i ima široku podršku u zajednici.

Nedostaci Redux-a

Ipak, Redux također ima svoje nedostatke. Implementacija Reduxa može biti kompleksna i zahtijeva dodatni kod za definiranje akcija, reduktora i akcijskih kreatora. Ovo može povećati količinu koda koji treba napisati i održavati. Također, za manje aplikacije Redux može biti preveliko rješenje, što može dovesti do nepotrebnog kompliciranja koda.

U zaključku, React i Redux su moćni alati za razvoj web aplikacija, ali imaju svoje prednosti i nedostatke. Izbor između njih ovisi o potrebama projekta i razumijevanju njihovih karakteristika. Važno je pravilno procijeniti koje tehnologije najbolje odgovaraju konkretnom zadatku kako bi se postigla uspješna implementacija web aplikacije.

5.1. Usporedba React-a s drugim tehnologijama

React je popularna JavaScript biblioteka za izgradnju korisničkih sučelja (engl. UI) i komponenti. Evo usporedbe Reacta s nekim drugim tehnologijama za izgradnju korisničkih sučelja:

1. **Angular**:

- **Sličnosti**: React i Angular su oba popularna okvira za izgradnju web aplikacija. Oba koriste komponentnu arhitekturu za izradu korisničkih sučelja.
- **Razlike**: React je JavaScript knjižnica, dok je Angular potpun okvir za razvoj koji koristi TypeScript. Angular ima ugrađene alate za upravljanje stanjem, rute i mnoge druge funkcionalnosti, dok React ostavlja veći dio ovih odluka na razvojnom timu ili zahtijeva korištenje dodatnih biblioteka. [15]

2. **Vue.js:**

- **Sličnosti**: Vue.js je također JavaScript knjižnica za izgradnju korisničkih sučelja koja koristi komponentnu arhitekturu, slično kao React.
- **Razlike**: Vue.js ima jednostavniju sintaksu i manji prag za učenje u usporedbi s Reactom. React je podržan većom zajednicom i ima veći broj dostupnih dodataka i biblioteka. [16]

5.2. Usporedba Redux-a s drugim tehnologijama

Redux je upravitelj stanja za JavaScript aplikacije, posebno popularan u kontekstu React aplikacija. Evo usporedbe Reduxa s nekim drugim tehnologijama za upravljanje stanjem:

1. **MobX**:

- **Sličnosti**: Oba Redux i MobX su tehnologije za upravljanje stanjem u React aplikacijama. Oba omogućavaju centralizirano pohranjivanje i upravljanje stanjem aplikacije.
- **Razlike**: Redux koristi strogi protokol za upravljanje stanjem, dok MobX koristi reaktivni pristup. MobX omogućava deklarativno praćenje promjena u stanju i automatsko ažuriranje komponenti, dok Redux zahtijeva eksplicitne akcije za promjenu stanja. [17]

2. **GraphQL**:

- **Sličnosti**: I Redux i GraphQL se koriste za upravljanje podacima u aplikacijama. Oba omogućavaju efikasno dohvaćanje podataka i ažuriranje stanja aplikacije.

- **Razlike**: Redux je prvenstveno fokusiran na upravljanje klijentskim stanjem, dok je GraphQL jezik za postavljanje upita za dohvaćanje podataka s servera. Redux može raditi zajedno s GraphQL-om kako bi se upravljalo klijentskim stanjem na klijentskoj strani. [18]

3. Appolo Client

- **Sličnosti**: Apollo Client se koristi za upravljanje podacima u React aplikacijama koje koriste GraphQL za dohvaćanje podataka.
- **Razlike**: Redux je općenitiji i može se koristiti s različitim izvorima podataka, dok je Apollo Client specifičan za GraphQL. Apollo Client olakšava upravljanje podacima koji dolaze s GraphQL servera, uključujući upite i mutacije. [19]

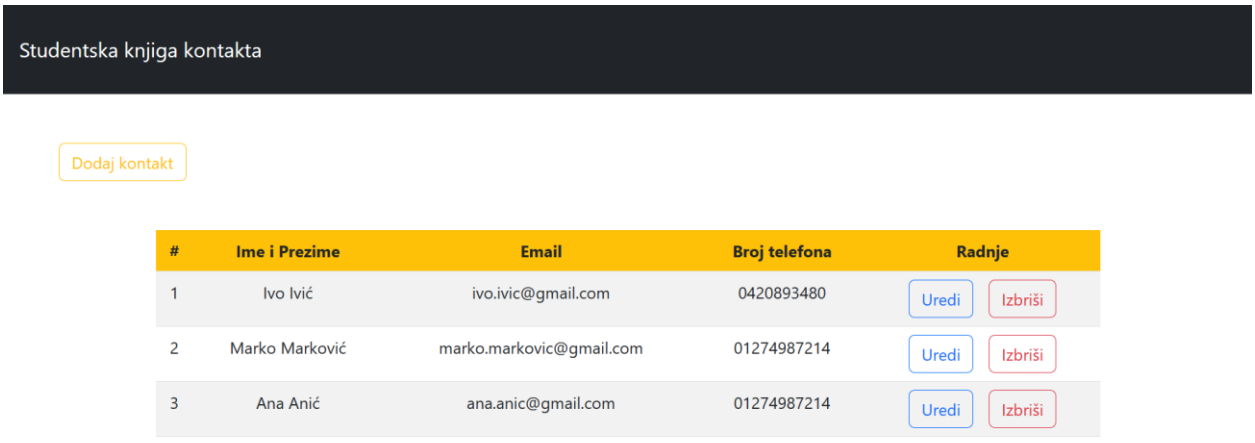
U konačnici, izbor između Reduxa i drugih tehnologija ovisi o specifičnim potrebama projekta, okolini u kojoj se radi i osobnom iskustvu i preferencijama.

6. Primjer izvedbe CRUD operacija

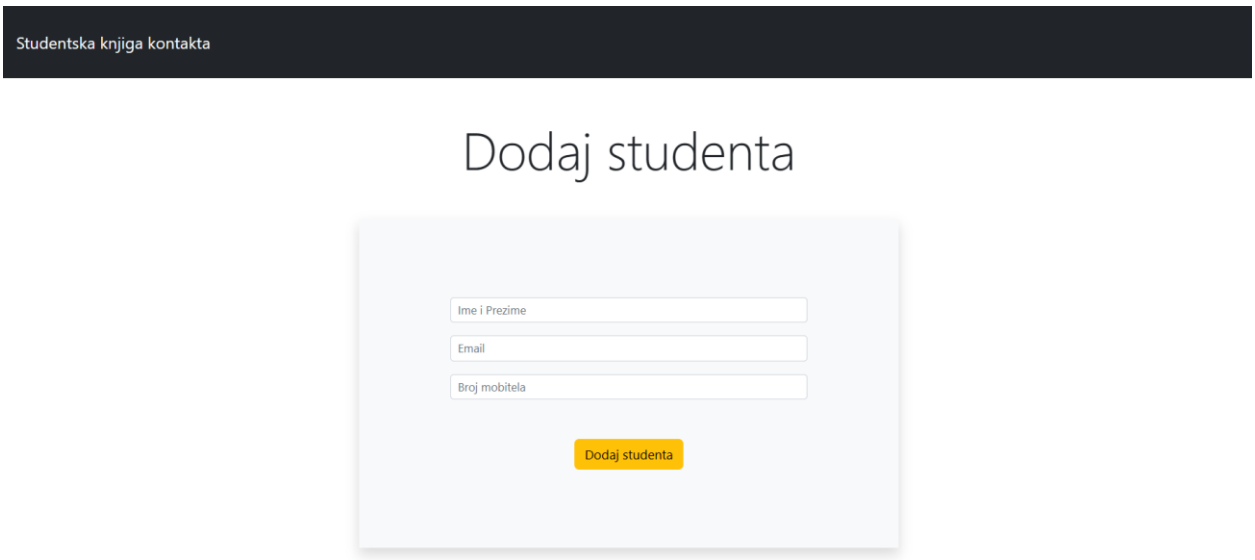
Praktični dio ovog završnog rada fokusira se na razvoj jednostavne CRUD aplikacije koja koristi React.js i Redux za učinkovito upravljanje podacima, dok se za oblikovanje korisničkog sučelja koristi Bootstrap. Ovaj projekt će demonstrirati primjenu ovih tehnologija u izradi funkcionalne web aplikacije, istražujući korake razvoja, integraciju Redux-a i primjenu Bootstrap-a kako bi se stvorila intuitivna i održiva aplikacija.

6.1. Koncept i dizajn aplikacije

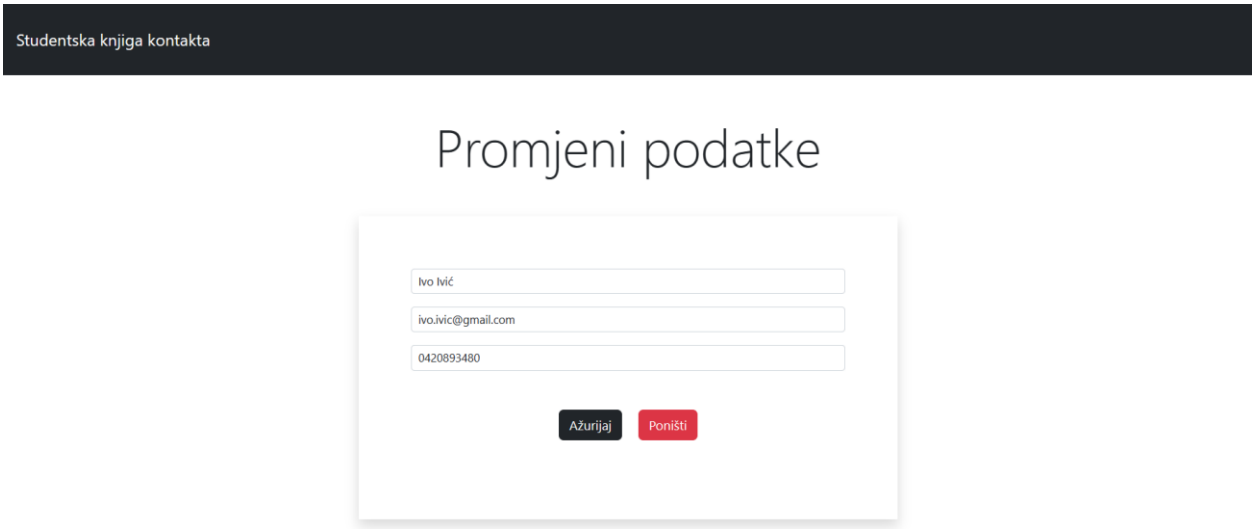
Ova aplikacija omogućuje korisnicima stvaranje, čitanje, ažuriranje i brisanje stavki unutar sustava. Ciljana publika su korisnici koji trebaju učinkovito upravljati svojim podacima putem sučelja koje je jednostavno za korištenje. Osnova aplikacije je React.js. Za efikasno upravljanje stanjem aplikacije koristi se Redux. Bootstrap se koristi za dizajn korisničkog sučelja s žutom i crnom bojom kao primarnim bojama. Poruke i obavijesti za korisnike implementiraju se pomoću biblioteke Toastify. Aplikacija se sastoji od komponenata za unos, prikaz, uređivanje i brisanje stavki. Redux upravlja globalnim stanjem, uključujući podatke o stavkama. Komunikacija između komponenata i Redux-a ostvarena je korištenjem akcija i reduktora. Dizajn korisničkog sučelja izveden je pomoću Bootstrap-a, s žutom i crnom bojom kao primarnim bojama. Sučelje je jednostavno i intuitivno za korisnike kako bi olakšalo stvaranje, uređivanje i brisanje stavki. Toastify se koristi za prikaz obavijesti i poruka korisnicima. U ovom primjeru, podaci se privremeno pohranjuju lokalno u obliku objekata. Aplikacija simulira osnovne CRUD operacije nad tim podacima. U nastavku će biti prikazan izgled web stranice.



SLIKA 1 Dizajn i prikaz početne stranice web aplikacije



SLIKA 2 Dizajn i prikaz web stranice dodaj kontakt

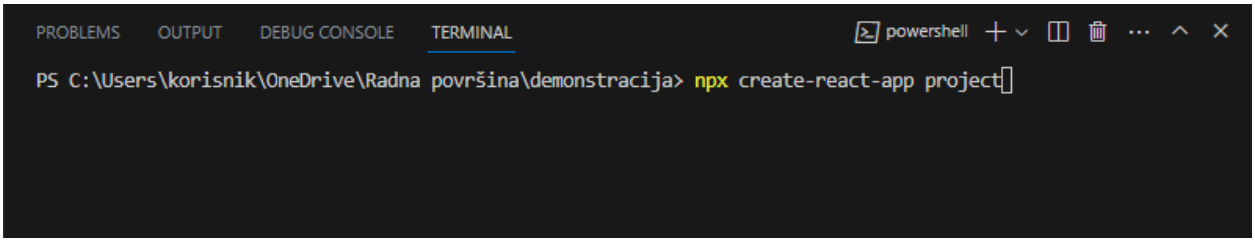


SLIKA 3 Dizajn i prikaz web stranice promjene podataka

6.2. Opis praktičnog dijela

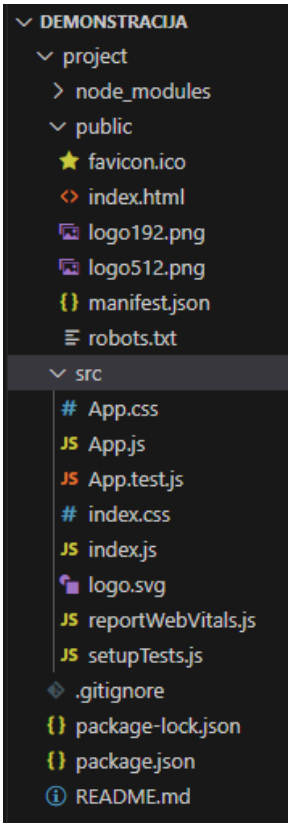
Na samom početku potrebno je instalirati na računalu ili laptopu Node js. Node.js se koristi u razvoju React.js aplikacija iz nekoliko razloga koji su izrazito korisni. Dakle, Node.js dolazi sa svojim svojstvenim paketnim menadžerom poznatim kao npm (engl. Node Package Manager). Ovaj alat omogućava programerima da lako upravljaju bibliotekama i zavisnostima koje su potrebne za izgradnju i rad njihovih React.js aplikacija. Kroz npm, programeri mogu jednostavno instalirati, ažurirati ili ukloniti biblioteke i pakete koji su neophodni za razvoj i funkcionalnost njihovih aplikacija. Ovo je posebno korisno za izvršavanje jednokratnih naredbi ili skripti. Na

primer, naredba "npx create-react-app" omogućava programerima da generiraju osnovnu strukturu React.js aplikacije bez potrebe za prethodnom instalacijom create-react-app paketa. npx će automatski preuzeti paket i pokrenuti odgovarajuću naredbu. Taj dio je prikazan na slici

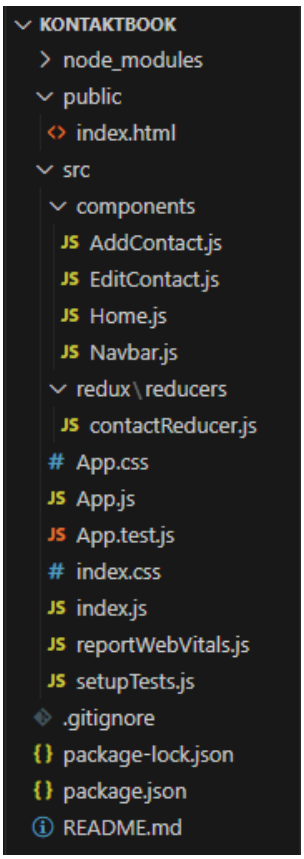


SLIKA 4 Prikaz terminala u VS kodu i kreiranje react aplikacije pomoću npx

Na slici 5 vidi se prikaz kako izgledaju react skripte koje su instalirane. U mom projektu većina ovoga nije potrebno tako da ćemo izbrisati neke od ovih skripti te će ovako struktura moga projekta izgledati prikazano na slici 6.



SLIKA 5 Prikaz izgrađenog projekta



SLIKA 6 prikaz kreiranih komponenti potrebnih za izradu projekta

Ovako izgleda struktura skripti web aplikacije. Prikazat ćemo samo neke kodove jer su svi uglavnom preopširni ili slični. Prvo ćemo prikazati kod početnog dijela stranice.

```
const Home = () => {
  return (
    <div className='container'>
        <div className='row'>
            <div className='col-md-12 my-5 text-right'>
                <Link  to='/add'  className='btn  btn-outline-warning
'>Dodaj kontakt</Link>
            </div>
            <div className='col-md-10 mx-auto'>
                <table className='table  table-striped table-hover'>
                  <thead className=' bg-warning text-center '>
                    <tr>
                      <th scope='col'>#</th>
                      <th scope='col'>Ime i Prezime</th>
                      <th scope='col'>Email</th>
                      <th scope='col'>Broj telefona</th>
                      <th scope='col'>Radnje</th>

                    </tr>
                  </thead>
                  <tbody >
                    {
                      contacts.map((contact, id) =>(
             <tr key={id}>
```

```
                    <td className='text-center '>{id + 1}</td>
                        <td className='text-center'>{contact.name}</td>
                            <td className='text-center'>{contact.email}</td>
                    <td className='text-center'>{contact.number}</td>
                            <td className='text-center '>
            <Link  to={`/edit/${contact.id}`} className='btn btn-outline-
primary' style={{ marginRight: '15px' }}>Uredi</Link>
                                <button
                                    type='button'
                                    onClick={()=deleteContact(contact.id}
                                 className='btn           btn-outline-danger
                                 '>Izbriši</button>
                                 </td>
                               </tr>

                             ))
                          }
                        </tbody>
                        </table>

                </div>
            </div>
        </div>
    )
}

export default Home;
```

Programski kod 16 Prikaz funkcijske komponente Home

Ovaj kod definira funkcionalnu komponentu Home. Kada se komponenta renderira, ona će vratiti JSX kod koji predstavlja korisničko sučelje. Komponenta Home je oblikovana kao div element s klasom "container", koji obuhvaća ostatak korisničkog sučelja. Unutar tog div elementa nalazi se div element s klasom "row", koji predstavlja redak u korisničkom sučelju. U prvom stupcu reda nalazi se div element s klasama "col-md-12 my-5 text-right". Ovaj element sadrži Link komponentu koja stvara poveznicu na putanju "/add" za dodavanje novog kontakta. Poveznica je stilizirana kao gumb s klasama "btn btn-outline-warning". U drugom stupcu reda nalazi se div element s klasama "col-md-10 mx-auto". Unutar ovog elementa smještena je tablica koja se stilizira s klasama "table table-striped table-hover". U zaglavlju tablice definirani su zaglavlje s podacima kao što su "Ime i Prezime", "Email" i "Broj telefona". U tijelu tablice koristi se map metoda za iteriranje kroz kontakte. Svaki kontakt se prikazuje kao redak tablice. Za svaki kontakt prikazuje se redni broj, ime i prezime, email, broj telefona, te gumbi "Uredi" i "Izbriši". Gumb "Uredi" je povezan s putanjom koja sadrži id kontakta za uređivanje, dok gumb "Izbriši" poziva funkciju deleteContact s id-em kontakta za brisanje. Na kraju funkcionalne komponente Home, JSX kod je zatvoren, a komponenta je izvezena kako bi bila dostupna drugim dijelovima aplikacije. Zatim logika vezan za komponentu Home nalazi se u kodu ispod.

```
const contacts = useSelector(state=>state);
 const dispatch = useDispatch();
 const deleteContact = (id) => {
   dispatch({type:'DELETE_CONTACT', payload:id});
   toast.success("Kontakt je uspješno izbrisan! ")
 };
```

Programski kod 17 Prikaz logičkog dijela funkcijske komponente Home

Ovaj kod koristi se u Reduxu za upravljanje stanjem aplikacije. Prva linija, **const contacts = useSelector(state => state);**, izvlači sve informacije iz Redux skladišta i sprema ih u promjenljivu **contacts**. Zatim se koristi **useDispatch** kako bi se dobila pristup Redux-ovoj **dispatch** funkcionalnosti, što omogućava slanje akcija Redux skladištu. Funkcija **deleteContact** se koristi za brisanje kontakta iz Redux skladišta. Prima ID kontakta kao argument. Kada se pozove **deleteContact(id)** funkcija, šalje se Redux akcija tipa **'DELETE_CONTACT'** sa payloadom koji sadrži ID kontakta koji se želi izbrisati. Nakon što se kontakt uspješno izbriše iz Redux skladišta, koristi se **toast.success** da bi se prikazala poruka "Kontakt je uspješno izbrisan!" korisniku. Ova poruka služi kao obavještenje o uspješnom brisanju kontakta.

Nadalje, kada stisnemo gumb “Dodaj kontakt”, aktivira se komponenta “Add Contact” čiji kod izgleda ovako:

```
const AddContact = () => {
  return (
    <div className='container'>
        <h1 className='display-3 my-5 text-center'>
                Dodaj studenta
            </h1>
        <div className='row'>
          <div className='col-md-6 shadow mx-auto p-5 bg-light '>
           <form onSubmit={handleSubmit} className='text-center m-5'>
                    <div className = 'form-group m-3'>

                    <input className="form-control form-control-sm"
                        type="text"
                        placeholder="Ime i Prezime"
                        aria-label=".form-control-sm example"
                        value={name}
                        onChange={e=> setName(e.target.value)}/>
                    </div>

                    <div className = 'form-group m-3'>
                    <input className="form-control form-control-sm"
                        type="email"
                        placeholder="Email"
                        aria-label=".form-control-sm example"
                        value={email}
                        onChange={e=> setEmail(e.target.value)} />
                    </div>

                    <div className = 'form-group m-3'>
                    <input className="form-control form-control-sm"
```

```
                        type="number"
                        placeholder="Broj mobitela"
                        aria-label=".form-control-sm example"
                        value={number}
                        onChange={e=> setNumber(e.target.value)}/>
                    </div>
                    <div className = 'form-group m-5'>
                     <input
                     type='submit'
                     value='Dodaj studenta'
                     className='btn btn-warning'/>
                    </div>
                </form>
            </div>
        </div>
    </div>
  )
}

export default AddContact
```

Programski kod 18 Prikaz komponente AddContact

Ovaj kod predstavlja React komponentu nazvanu "AddContact" koja omogućava dodavanje informacija o studentima. Kada se koristi u aplikaciji, ova komponenta će izgledati kao formular za unos podataka kao što je prikazano na slici 2. U komponenti se koristi Bootstrap okvir za stilizaciju i raspoređivanje elemenata kako bi se postigao odgovarajući izgled.

Naslov "Dodaj studenta" se prikazuje na sredini ekrana s veličinom fonta definiranom kao "display-3".

Forma za unos informacija o studentu sadrži tri polja: za unos imena i prezimena, email adrese i broja mobilnog telefona studenta. Svako od ovih polja je povezano s unutrašnjim stanjem komponente putem **value** i **onChange** svojstava.

Na dnu forme nalazi se dugme "Dodaj studenta" koje će aktivirati proces submitovanja forme kad se klikne. Međutim, funkcija **handleSubmit** koja bi trebala obraditi ovu akciju nije prikazana u ovom kodu. Unutar komponente nalazi se I logički dio koji je prikazan u kodu ispod.

```
    const [name, setName] = useState("");
    const [email, setEmail] = useState("");
    const [number, setNumber] = useState("");


    const contacts = useSelector((state) => state);
    const dispatch = useDispatch();

    const history = useHistory()


    const handleSubmit = (e) => {
        e.preventDefault();
```

```
        const checkEmail  = contacts.find(contact => contact.email === email && contact);

        const checkNumber = contacts.find(
            (contact) => contact.number === parseInt(number)
        );

        if(!email || !number || !name){
            return toast.warning('Molim popunite sva polja!')
        }
        if(checkEmail){
            return toast.error("Ovaj email već postoji!!!")
        }
        if(checkNumber){
            return toast.error("Ovaj broj već postoji!!!")
        }

        const data = {
            id: contacts[contacts.length - 1].id + 1,
            name,
            email,
            number,
        }
        console.log(data);


        dispatch ({type: 'ADD_CONTACT', payload : data});
        toast.success('Student je dodan uspješno!!!')
        history.push('/');
          }
```

Programski kod 19 Prikaz logičkog dijela funkcijske komponente AddContact

Ovaj kod predstavlja React komponentu nazvanu "AddContact" koja omogućava unos informacija o studentima. Evo detaljnog objašnjenja svake linije koda:

1. **const [name, setName] = useState("");**:
    - Ova linija koda koristi React kuku **useState** za definiranje promjenljive **name** i funkcije **setName**. Promjenljiva **name** će sadržavati ime studenta, dok će funkcija **setName** omogućiti postavljanje vrijednosti **name**. Početna vrijednost **name** je postavljena na prazan string ("").
2. **const [email, setEmail] = useState("");**:
    - Slično kao prethodno, ovdje se koristi **useState** za **email** i funkciju **setEmail**. Promjenljiva **email** će sadržavati email adresu studenta, a početna vrijednost je također prazan string.
3. **const [number, setNumber] = useState("");**:
    - Ovdje se opet koristi **useState** za promjenljivu **number** i funkciju **setNumber**. Promjenljiva **number** će sadržavati broj mobilnog telefona studenta, a početna vrijednost je također prazan string.

4. **const contacts = useSelector((state) => state);**:
    - Ova linija koristi Redux kuku **useSelector** kako bi izvukla objekt **contacts** iz globalnog stanja aplikacije. **useSelector** omogućava pristup stanju u Redux skladištu.
5. **const dispatch = useDispatch();**:
    - Ovdje se koristi Redux kuka **useDispatch** kako bi se dobila pristup **dispatch** funkciji. **dispatch** se koristi za slanje Redux akcija kako bi se izmijenilo stanje aplikacije.
6. **const history = useHistory();**:
    - Ova linija koristi React Router kuku **useHistory** za pristup povijesti preglednika. To omogućava preusmjeravanje korisnika na drugu rutu nakon što se unesu informacije o studentu.
7. **const handleSubmit = (e) => { ... }**:
    - Ova funkcija se poziva kada korisnik pošalje formu za unos informacija o studentu. Prima event objekt **e** koji se koristi za sprečavanje podnošenja forme putem preglednika (koristi se **e.preventDefault()**).
8. **const checkEmail = contacts.find(contact => contact.email === email && contact);**:
    - Ova linija koda provjerava postojanje unesene email adrese unutar postojećih kontakata. Ako se pronađe ista email adresa, ona se smješta u promjenljivu **checkEmail**.
9. **const checkNumber = contacts.find((contact) => contact.number === parseInt(number));**:
    - Slično kao prethodno, ovdje se traži postojanje broja mobilnog telefona unesenog u formi unutar postojećih kontakata.
10. Postavljaju se uvjeti za provjeru unosa i postojanja istih email adresa/brojeva, uz ispisivanje odgovarajućih obavijesti kako bi se spriječilo dodavanje istih podataka.
11. Kreira se objekt **data** koji sadrži podatke o studentu (ime, email i broj mobilnog telefona). ID se računa na osnovu posljednjeg studenta u listi.
12. Šalje se Redux akcija putem **dispatch** funkcije sa tipom **'ADD_CONTACT'** i podacima o studentu kao payload.
13. Prikazuje se obavijest o uspješnom dodavanju studenta korisniku putem **toast.success**.
14. Korisnika se preusmjerava na početnu rutu ('/') putem **history.push('/')** kako bi se omogućila vidljivost dodanog studenta u aplikaciji.

Što se tiče dijela koji se odnosi na komponentu „Edit Contact“, njezin JSX kod isti je kao i u komponenti „Add Contact“ ali joj se razlikuje logički dio koji će biti objašnjen u nastavku.

```
  const [name, setName] = useState("");
    const [email, setEmail] = useState("");
    const [number, setNumber] = useState("");
    const {id} = useParams();

    const contacts = useSelector(state=>state);
    const currentContact = contacts.find(
        contact => contact.id === parseInt(id))

    useEffect(()=>{
        if (currentContact) {
            setName(currentContact.name);
            setEmail(currentContact.email);
            setNumber(currentContact.number);
        }
    }, [currentContact])

    const dispatch = useDispatch();
     const history = useHistory()

    const handleSubmit = (e) => {
        e.preventDefault();

        const checkEmail  = contacts.find((contact) => contact.id !==
parseInt(id) && contact.email === email);

        const checkNumber = contacts.find(
            (contact) => contact.id !== parseInt(id) && contact.number
=== parseInt(number)
        );

        if(!email || !number || !name){
            return toast.warning('Molimo popunite sva polja!')
        }
        if(checkEmail){
            return toast.error("Ovaj email već postoji!!!")
        }
        if(checkNumber){
            return toast.error("Ovaj broj već postoji!!!")
        }

        const data = {
            id: parseInt(id),
            name,
            email,
            number,
        }

        dispatch ({type: 'UPDATE_CONTACT', payload : data});
        toast.success('Student je dodan uspješno!!!')
        history.push('/');
    }
```

Programski kod 20 Prikaz logičkog dijela funkcijekse komponente EditContact

Ovaj kod predstavlja React komponentu nazvanu "EditContact" koja omogućava uređivanje informacija o studentima. U nastavku slijedi detaljnog objašnjenja svake linije koda:

1. **const [name, setName] = useState("");**, **const [email, setEmail] = useState("");**, **const [number, setNumber] = useState("");**:
    - Ove linije koda koriste React kuka **useState** za definiranje tri promjenljive (**name**, **email**, **number**) i tri odgovarajuće funkcije (**setName**, **setEmail**, **setNumber**). Ove promjenljive će čuvati informacije o studentu (ime, email, broj mobilnog telefona), a funkcije će omogućiti promjenu tih vrijednosti. Početne vrijednosti su postavljene na prazne stringove.
2. **const { id } = useParams();**:
    - Ova linija koristi React Router kuka **useParams** za izdvajanje **id** parametra iz URL-a. **id** će biti koristi za identifikaciju studenta koji se uređuje.
3. **const contacts = useSelector(state=>state);**:
    - Ova linija koristi Redux kuka **useSelector** za izvlačenje objekta **contacts** iz globalnog stanja aplikacije. **useSelector** omogućava pristup stanju u Redux skladištu.
4. **const currentContact = contacts.find(contact => contact.id === parseInt(id));**:
    - Ova linija koda koristi **Array.prototype.find** funkciju kako bi pronašla studenta čiji **id** odgovara **id** parametru iz URL-a. **currentContact** će sadržavati informacije o trenutnom studentu koji se uređuje.
5. **useEffect(() => { ... }, [currentContact]);**:
    - Ova funkcija se koristi za postavljanje početnih vrijednost unutar forme na osnovu trenutnog studenta koji se uređuje. Koristi se **useEffect** kuka kako bi se reagiralo na promjene **currentContact** promjenljive.
6. **const dispatch = useDispatch();**:
    - Ova linija koristi Redux kuka **useDispatch** kako bi se dobila pristup **dispatch** funkciji. **dispatch** se koristi za slanje Redux akcija kako bi se izmijenilo stanje aplikacije.
7. **const history = useHistory();**:
    - Ovdje se koristi React Router kuka **useHistory** za pristup povijesti preglednika. To omogućava preusmjeravanje korisnika na drugu rutu nakon što se sačuvaju izmjene o studentu.
8. **const handleSubmit = (e) => { ... }**:

- Ova funkcija se poziva kada korisnik pošalje formu za uređivanje informacija o studentu. Prima event objekta **e** koji se koristi za sprečavanje podnošenja forme putem preglednika (koristi se **e.preventDefault()**).

9. **const checkEmail = contacts.find((contact) => contact.id !== parseInt(id) && contact.email === email);**:
    - Ova linija koda provjerava postojanje unesene email adrese unutar postojećih kontakata, ali neće uzeti u obzir trenutnog studenta čiji se podaci uređuju.
10. **const checkNumber = contacts.find((contact) => contact.id !== parseInt(id) && contact.number === parseInt(number));**:
    - Slično kao i prethodna linija, ova linija traži postojanje broja mobilnog telefona unesenog u formi unutar postojećih kontakata, ali isključuje trenutnog studenta koji se uređuje.
11. Postavljaju se uvjeti za provjeru unosa i postojanja istih email adresa/brojeva, uz ispisivanje odgovarajućih obavještenja kako bi se spriječilo dodavanje istih podataka.
12. Kreira se objekt **data** koji sadrži ažurirane podatke o studentu (id, ime, email i broj mobilnog telefona).
13. Šalje se Redux akcija putem **dispatch** funkcije sa tipom **'UPDATE_CONTACT'** i podacima o studentu kao payload.
14. Prikazuje se obavijest o uspješnom ažuriranju studenta korisniku putem **toast.success**.
15. Korisnika se preusmjerava na početnu rutu ('/') putem **history.push('/')** kako bi se omogućila vidljivost ažuriranog studenta u aplikaciji.

Ovaj kod omogućava uređivanje informacija o studentu unutar aplikacije putem forme i upravlja tim procesom kroz Redux.

Dakle, u ovoj web aplikaciji, podaci se spremaju kao objekt pomoću reduxa, kod je prikazan i objašnjen u nastavku.

```
const initialState = [
    {
        id: 0,
        name: 'Ivo Ivić',
        number: '0420893480',
        email: 'ivo.ivic@gmail.com'
    },
    {
        id: 1,
        name: 'Marko Marković',
        number: '01274987214',
        email: 'marko.markovic@gmail.com'
    },
    {
        id: 2,
```

```
        name: 'Ana Anić',
        number: '01274987214',
        email: 'ana.anic@gmail.com'
    },
];

const contactReducer = (state = initialState, action) => {
    switch(action.type){
        case 'ADD_CONTACT':
            state= [...state, action.payload];
            return state;
        case 'UPDATE_CONTACT':
            const updateState = state.map(contact => contact.id ===
action.payload.id ? action.payload: contact)
            state= updateState;
            return state;
        case 'DELETE_CONTACT':
            const filterContacts = state.filter(contact => contact.id
!== action.payload && contact);
            state = filterContacts;
            return state;

        default:
            return state;
    }
};

export default contactReducer;
```

Programski kod 21 Prikaz reduktora gdje se podaci spremaju u obliku objekta i logički dio

Ovaj kod definira Redux reducer funkciju za upravljanje stanjem kontakata. Inicijalno stanje (engl. initialState) definira popis kontakata koji se sastoje od objekata s atributima "id", "name", "number" i "email". Svaki kontakt ima jedinstveni "id" kako bi ih se moglo jednostavno identificirati.

Reducer funkcija (*contactReducer*) prima trenutno stanje (*engl. state*) i akciju (*engl. action*) te na temelju akcije vrši odgovarajuće promjene nad stanjem. U switch bloku, ovisno o vrsti akcije, provode se različite operacije:

1. Akcija 'ADD_CONTACT': Dodaje novi kontakt u postojeće stanje. Kreira se novo stanje (novi niz) pomoću *spread* operatora (operator širenja) (...) koji kopira postojeće kontakte i dodaje novi kontakt iz akcije (action.payload). Novo stanje se zatim vraća.
2. Akcija 'UPDATE_CONTACT': Ažurira postojeći kontakt u stanju. Korištenjem metode map() prolazi se kroz sve kontakte u stanju. Ako je "id" kontakta jednak "id" iz akcije (action.payload), tada se koristi novi kontakt iz akcije kako bi se zamijenio postojeći kontakt. Inače, zadržava se postojeći kontakt nepromijenjen. Novo stanje se zatim vraća.
3. Akcija 'DELETE_CONTACT': Briše kontakt iz stanja. Metodom filter() prolazi se kroz sve kontakte u stanju i filtriraju se samo oni kontakti čiji "id" nije jednak "id" iz akcije

(action.payload). To znači da se kontakt s odgovarajućim "id"-om briše iz stanja. Novo stanje se zatim vraća.

4. Default case: Ako nijedna od prethodno navedenih akcija nije zadovoljena, tada se vraća nepromijenjeno trenutno stanje.

Na kraju, reducer funkcija se izvozi kao zadani (engl. default) izvoz, što znači da će biti dostupna drugim dijelovima aplikacije koji koriste taj reducer za upravljanje stanjem kontakata.

Kroz ovaj projekt, uspješno smo demonstrirali primjenu ovih tehnologija u izradi funkcionalne web aplikacije. Evo ključnih zaključaka iz ovog praktičnog iskustva:

- Korištenje React.js-a za izradu korisničkog sučelja omogućilo je brz i dinamičan razvoj komponenata aplikacije, stvarajući responzivno sučelje za modernu web aplikaciju.
- Redux se pokazao korisnim za efikasno upravljanje stanjem aplikacije. Centralizirano upravljanje podacima i akcijama putem Reduxa olakšava održavanje i proširivost aplikacije.

Bootstrap je olakšao oblikovanje jednostavnog korisničkog sučelja bez potrebe za dugotrajnim dizajniranjem. Njegove CSS komponente i reaktivnost olakšavaju stvaranje funkcionalnog i estetski privlačnog sučelja.

Poveznica za aplikaciju https://crud-app-zavrsni-rad.netlify.app/#/

7. Zaključak

Zaključak ovog rada ističe ključne elemente istraživanja koji se odnose na primjenu Reacta, Reduxa i Bootstrapa u izradi CRUD web aplikacija.

Prvo, React se pokazao iznimno snažnom i popularnom JavaScript bibliotekom za razvoj suvremenih web aplikacija. Njegova komponentna arhitektura omogućuje izradu modularnih i responsivnih korisničkih sučelja, čineći ga privlačnim izborom za front-end razvoj.

Drugo, Redux je pružio efikasno rješenje za upravljanje stanjem aplikacije. Njegovo centralizirano skladište podataka olakšava praćenje i ažuriranje stanja aplikacije, što je posebno važno u složenim web aplikacijama.

Treće, Bootstrap je demonstrirao svoju vrijednost kao alat za brzo i jednostavno oblikovanje korisničkog sučelja. Njegova biblioteka CSS komponenata omogućava dosljedan i atraktivan dizajn aplikacije, čime se štedi vrijeme i trud u procesu razvoja.

Praktični dio rada, koji uključuje izradu CRUD web aplikacije uz pomoć Reacta, Reduxa i Bootstrapa, ilustrira kako se ovi alati mogu integrirati u stvarnom razvojnom okruženju. Postupci izrade aplikacije detaljno su dokumentirani, a slike i objašnjenja olakšavaju razumijevanje procesa.

Zaključno, ovaj rad potvrđuje da su React, Redux i Bootstrap snažni resursi za izradu suvremenih web aplikacija koje zadovoljavaju zahtjeve učinkovitosti, funkcionalnosti i korisničkog iskustva. Njihova integracija omogućuje razvoj aplikacija koje su skalabilne i responzivne, istovremeno privlačne za korisnike.

8. Literatura

[1] K. Chris, »freeCodeCamp,« 15 Lipanj 2022. [Mrežno]. Available: https://www.freecodecamp.org/news/crud-operations-explained/. [Pokušaj pristupa 6 9 2023].

[2] »code academy,« [Mrežno]. Available: https://www.codecademy.com/article/what-is-crud.

[3] C. Gackenheimer, Introduction to React, Apress, 2015.

[4] A. Bastakoti, »Using Native Mobile Services in React JS.,« 2022.

[5] E. Saks, »JavaScript Frameworks: Angular vs React vs Vue.,« 2019.

[6] R. Wieruch, The road to react: Your journey to master plain yet pragmatic react. js., 2017.

[7] S. Aggarwal, »"Modern web-development using reactjs.",« International Journal of Recent Research Aspects 5.1, 2018.

[8] R. Wieruch, The road to react: Your journey to master plain yet pragmatic react. js, 2017.

[9] »React,« [Mrežno]. Available: https://legacy.reactjs.org/docs/hooks-intro.html. [Pokušaj pristupa 6 9 2023].

[10] M. K. Caspers, »React and redux.,« u *Rich Internet Applications w/HTML and Javascript 11*, 2017.

[11] T. A. V. a. N. T. A. T. Le, »Implementation of React-Redux in web application,« 2020.

[12] Redux, »Redux,« 25 Listopad 2021. [Mrežno]. Available: https://redux.js.org/api/store. [Pokušaj pristupa 6 rujan 2023].

[13] »React Redux,« 11 Lipanj 2023. [Mrežno]. Available: https://react-redux.js.org/api/provider. [Pokušaj pristupa 6 Rujan 2023].

[14] »React Redux,« 13 Lipanj 2023. [Mrežno]. Available: https://react-redux.js.org/api/hooks. [Pokušaj pristupa 6 Rujan 2023].

[15] »Angular,« [Mrežno]. Available: https://angular.io/. [Pokušaj pristupa 6 Rujan 2023].

[16] »Vue,« [Mrežno]. Available: https://vuejs.org/. [Pokušaj pristupa 6 Rujan 2023].

[17] »MobX,« [Mrežno]. Available: https://mobx.js.org/. [Pokušaj pristupa 6 rujan 2023].

[18] »GraphQL,« [Mrežno]. Available: https://graphql.org/. [Pokušaj pristupa 5 Rujan 2023].

[19] »Apollo Client,« [Mrežno]. Available: https://www.apollographql.com/docs/react/. [Pokušaj pristupa 5 Rujan 2023].

[20] D. M. S. a. D. C. Sengupta, »Getting Started with React,« Packt Publishing Ltd, 2016.

Popis slika

Popis programskih kodova

Popis tablica

Prilozi

Poveznica za aplikaciju :

https://crud-app-zavrsni-rad.netlify.app/#/

Sveučilište
Sjever

IZJAVA O AUTORSTVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, Edita Golić *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog *(obrisati nepotrebno)* rada pod naslovom Izrada CRUD web aplikacije pomoću Laravel radnog okvira *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)
Edita Golić
(vlastoručni potpis)

Sukladno čl. 83. Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Sukladno čl. 111. Zakona o autorskom pravu i srodnim pravima student se ne može protiviti da se njegov završni rad stvoren na bilo kojem studiju na visokom učilištu učini dostupnim javnosti na odgovarajućoj javnoj mrežnoj bazi sveučilišne knjižnice, knjižnice sastavnice sveučilišta, knjižnice veleučilišta ili visoke škole i/ili na javnoj mrežnoj bazi završnih radova Nacionalne i sveučilišne knjižnice, sukladno zakonu kojim se uređuje znanstvena i umjetnička djelatnost i visoko obrazovanje.