Univerzitet u Beogradu

Fakultet organizacionih nauka

Katedra za elektronsko poslovanje

ReactJS

Domaći zadatak 3

|  |  |  |
| --- | --- | --- |
| Ime | Prezime | Broj indeksa |
|  |  |  |
| Link ka Github-u |  | |

Sadržaj

[1 Korisnički zahtev 3](#_Toc117932679)

[2 Implementacija 3](#_Toc117932680)

[3 Korisničko uputstvo 3](#_Toc117932681)

# Korisnički zahtev

Potrebno je kreirati web platformu za posredovanje u zapošljavanju, sa fokusom na IT sektor. Glavna funkcionalnost aplikacije uključuje mogućnost pregledanja i filtriranja liste poslovnih ponuda, kao i procese prijave i registracije korisnika.

Kroz registracioni obrazac, korisnici mogu da kreiraju svoj nalog, unosom osnovnih informacija kao što su ime, e-mail, lozinka i uloga (npr. student ili firma). Jednom registrovani, korisnici imaju mogućnost prijave na platformu, gde mogu da unesu svoj e-mail i lozinku kako bi pristupili svojim ličnim stranicama.

Jedna od centralnih funkcionalnosti aplikacije je pregled i pretraga poslovnih ponuda. Korisnici mogu da pregledaju aktuelne poslovne ponude, koje su prikupljene sa različitih izvora, uključujući Jooble API. Ove ponude su prikazane kroz pregledne kartice sa ključnim informacijama o poslovima, kao što su naziv pozicije, ime kompanije, opis posla, lokacija, plata i datum ažuriranja. Korisnici imaju opciju da filtriraju poslove po ključnim rečima, lokaciji ili platnom rangu, kao i da sortiraju poslove prema različitim kriterijumima (npr. po nazivu, plati ili datumu objave). Takođe, aplikacija podržava paginaciju, omogućavajući korisnicima da pregledaju poslove na više stranica.

# Implementacija

Fajl app.js nam daje pregled svih ruta koje su dostupne u aplikaciji

import React, { useState } from 'react';

import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

import './App.css';

import Login from './komponente/login/Login';

import JobList from './komponente/jobs/JobList';

import Registration from './komponente/Registration/Registration';

import Navbar from './komponente/Navbar/Navbar';

import JoobleListings from './komponente/joobleAPI/JoobleListings';

function App() {

  const [token,setToken]= useState(null);

  return (

    <Router>

      <div className="App">

        <Navbar token={token}  setToken={setToken}  ></Navbar>

        <Routes>

          <Route path="/" element={<Login setToken={setToken}/>} />

          <Route path="/registration" element={<Registration />} />

          <Route path="/jobs" element={<JobList />} />

          <Route path="/jooble" element={<JoobleListings />} />

          {/\* pod komentarom zbog ogranicenog broja upita \*/}

        </Routes>

      </div>

    </Router>

  );

}

export default App;

Komponenta Login je funkcionalna komponenta koja koristi React Hooks za upravljanje stanjima. Definiše dva stanja, email i password, koristeći useState hook. Ova stanja se koriste za prikupljanje korisničkih unosa iz input polja za e-poštu i lozinku. Takođe, koristi useNavigate hook za programsku navigaciju nakon uspešne prijave.

Glavna funkcija unutar komponente, handleLogin, je asinhrona funkcija koja upravlja procesom prijave. Kada se aktivira, funkcija šalje HTTP POST zahtev na određeni URL (http://127.0.0.1:8000/api/login) koristeći axios. Zahtev uključuje unesene korisničke podatke - e-poštu i lozinku. U slučaju uspešnog odgovora, funkcija ažurira token u stanju roditeljske komponente pomoću setToken, čuva token i korisničke podatke u sessionStorage za daljnje korištenje u aplikaciji, i navigira korisnika na stranicu /jobs. U slučaju neuspeha, funkcija hvata grešku i ispisuje poruku o grešci u konzoli.

Render deo komponente Login vraća JSX koji opisuje strukturu prijavnog obrasca. Forma sadrži dva input polja za unos e-pošte i lozinke, te dugme za prijavu koje aktivira handleLogin funkciju. Stilizacija forme se upravlja putem uvezenog CSS-a.

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

import axios from 'axios';

import './Login.css'

import { useNavigate } from 'react-router-dom';

const Login = ({setToken}) => {

  const [email, setEmail] = useState('');

  const [password, setPassword] = useState('');

  const navigate = useNavigate();

  const handleLogin = async () => {

    try {

      const response = await axios.post('http://127.0.0.1:8000/api/login', {

        email,

        password,

      });

      console.log(response.data);

      setToken(response.data.token)

      sessionStorage.setItem("token",response.data.token)

      sessionStorage.setItem("user",response.data.user)

      navigate('/jobs')

    } catch (error) {

         console.error('Login failed:', error.response.data.error);

    }

  };

  return (

    <div className="login-container">

      <h2>Login</h2>

      <div className="input-group">

        <label>Email:</label>

        <input

          type="email"

          value={email}

          onChange={(e) => setEmail(e.target.value)}

          required

        />

      </div>

      <div className="input-group">

        <label>Password:</label>

        <input

          type="password"

          value={password}

          onChange={(e) => setPassword(e.target.value)}

          required

        />

      </div>

      <button onClick={handleLogin}>Login</button>

    </div>

  );

};

export default Login;

Glavna funkcionalnost komponente Registration je handleRegister funkcija. Ova asinhrona funkcija koristi axios za slanje HTTP POST zahteva na određeni URL (http://127.0.0.1:8000/api/register). Zahtev sadrži podatke korisnika potrebne za registraciju - ime, e-poštu, lozinku i ulogu. U slučaju greške tokom zahteva, funkcija prikazuje greške koristeći alert i ispisuje ih u konzoli.

U JSX delu Registration komponente, koristi se niz InputField komponenti za unos imena, e-pošte i lozinke. Takođe, postoji dropdown meni za odabir uloge korisnika, sa opcijama "firma" i "student". Sve ove komponente su stilizovane koristeći CSS definisan u ./Registration.css.

Druga komponenta, InputField, je takođe funkcionalna React komponenta koja služi za prikaz pojedinačnog input polja. Ova komponenta prima label, type, value i onChange kao propove. label određuje tekst etikete input polja, type vrstu input polja (kao što su "text", "email", "password"), value trenutnu vrednost input polja, a onChange funkciju koja se poziva kada se vrednost input polja promeni. Ova komponenta omogućava ponovnu upotrebu input polja unutar različitih delova aplikacije, čime se poboljšava modularnost i smanjuje dupliranje koda.

import React from 'react';

const InputField = ({ label, type, value, onChange }) => {

  return (

    <div className="input-group">

      <label>{label}:</label>

      <input

        type={type}

        value={value}

        onChange={onChange}

      />

    </div>

  );

};

export default InputField;

import React, { useState } from 'react';

import axios from 'axios';

import './Registration.css';

import InputField from './InputField';

const Registration = () => {

  const [name, setName] = useState('');

  const [email, setEmail] = useState('');

  const [password, setPassword] = useState('');

  const [role, setRole] = useState('');

  const handleRegister = async () => {

    try {

      const response = await axios.post('http://127.0.0.1:8000/api/register', {

        name,

        email,

        password,

        role,

      });

      console.log(response.data);

    } catch (error) {

        const errors = error.response.data.errors;

        const errorMessages = Object.values(errors).flat(); // Spajamo sve poruke grešaka u jedan niz

        alert(`Registration failed:\n${errorMessages.join('\n')}`);

        console.error('Registration failed:', errors);

    }

  };

  return (

    <div className="login-container">

      <h2>Registration</h2>

      <InputField label="Name" type="text" value={name} onChange={(e) => setName(e.target.value)} />

      <InputField label="Email" type="email" value={email} onChange={(e) => setEmail(e.target.value)} />

      <InputField label="Password" type="password" value={password} onChange={(e) => setPassword(e.target.value)} />

      <div className="input-group">

        <label>Role:</label>

        <select value={role} onChange={(e) => setRole(e.target.value)}>

          <option value="firma">Firma</option>

          <option value="student">Student</option>

        </select>

      </div>

      <button onClick={handleRegister}>Register</button>

    </div>

  );

};

export default Registration;

JobList je funkcionalna komponenta koja koristi prilagođeni hook useJobs za dohvatanje liste poslova sa zadatog URL-a. Pored toga, komponenta upravlja nizom lokalnih stanja koristeći Reactov useState hook. Ova stanja uključuju filter za filtriranje poslova, salaryRange za određivanje opsega plata, sortKey i sortOrder za sortiranje poslova, kao i view za promenu prikaza liste poslova (lista ili mreža).

Komponenta sadrži funkcije za upravljanje promenama u filteru (handleFilterChange), promenama opsega plata (handleSalaryChange), kao i funkciju za sortiranje (handleSortChange). Pored toga, postoji i funkcija salaryFilter koja filtrira poslove na osnovu zadatog opsega plata.

Poslovi se prikazuju u sortiranom i filtriranom obliku, gde se sortiranje i filtriranje vrše na osnovu korisničkih unosa. Komponenta takođe omogućava korisniku da prebaci prikaz između liste i mreže koristeći funkciju toggleView.

JSX deo komponente JobList uključuje formu za filtriranje, dugmiće za sortiranje i kontejner u kome se prikazuju JobCard komponente za svaki posao koji odgovara kriterijumima filtriranja i sortiranja.

JobCard je takođe funkcionalna komponenta koja prikazuje detalje o svakom poslu. Ova komponenta prima objekat job kao prop i prikazuje informacije kao što su naslov posla, opis, rok za prijavu, platu, lokaciju, zahteve za posao, i ime kompanije.

import React, { useState } from 'react';

import './JobList.css';

import JobCard from './JobCard';

import useJobs from '../customHooks/useJobs';

const JobList = () => {

  const { jobs } = useJobs('http://127.0.0.1:8000/api/jobs');

  const [filter, setFilter] = useState({});

  const [salaryRange, setSalaryRange] = useState({ min: '', max: '' });

  const [sortKey, setSortKey] = useState('title');

  const [sortOrder, setSortOrder] = useState('asc');

  const [view, setView] = useState('list'); // 'grid' for grid view

  const handleFilterChange = (e) => {

    setFilter({ ...filter, [e.target.name]: e.target.value });

  };

  const handleSalaryChange = (e) => {

    setSalaryRange({ ...salaryRange, [e.target.name]: e.target.value });

  };

  const salaryFilter = (job) => {

    const minSalary = salaryRange.min ? parseInt(salaryRange.min) : 0;

    const maxSalary = salaryRange.max ? parseInt(salaryRange.max) : Infinity;

    return job.salary >= minSalary && job.salary <= maxSalary;

  };

  const handleSortChange = (key) => {

    setSortOrder(sortOrder === 'asc' ? 'desc' : 'asc');

    setSortKey(key);

  };

  const sortedJobs = [...jobs]

    .filter(job => Object.entries(filter).every(([key, value]) =>

      !value || job[key]?.toString().toLowerCase().includes(value.toLowerCase())

    ) && salaryFilter(job))

    .sort((a, b) => {

      if (a[sortKey] < b[sortKey]) return sortOrder === 'asc' ? -1 : 1;

      if (a[sortKey] > b[sortKey]) return sortOrder === 'asc' ? 1 : -1;

      return 0;

    });

  const toggleView = () => {

    setView(view === 'list' ? 'grid' : 'list');

  };

  return (

    <div>

      <div className="filter-form">

        <input type="text" name="title" placeholder="Title" onChange={handleFilterChange} />

        <input type="text" name="location" placeholder="Location" onChange={handleFilterChange} />

        <input type="number" name="min" placeholder="Min Salary" onChange={handleSalaryChange} />

        <input type="number" name="max" placeholder="Max Salary" onChange={handleSalaryChange} />

      </div>

      <div className="sort-buttons">

        <button onClick={() => handleSortChange('title')}>Sort by Title</button>

        <button onClick={() => handleSortChange('salary')}>Sort by Salary</button>

        <button onClick={() => handleSortChange('date')}>Sort by Date</button>

        <button onClick={toggleView}>Toggle View</button>

      </div>

      <div className={`job-list ${view}`}>

        {sortedJobs.map((job) => (

          <JobCard key={job.id} job={job} />

        ))}

      </div>

    </div>

  );

};

export default JobList;

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

const JobCard = ({ job }) => {

  return (

    <div className="job-card">

      <h3>{job.title}</h3>

      <p>{job.description}</p>

      <p>Deadline: {job.deadline}</p>

      <p>Salary: {job.salary}</p>

      <p>Location: {job.location}</p>

      <p>Requirements: {job.requirements}</p>

      <p>Company: {job.company.name}</p>

    </div>

  );

};

export default JobCard;

Na početku, hook useJobs koristi useState da inicijalizuje stanje jobs kao prazan niz. Ovo stanje će kasnije biti korišćeno za čuvanje podataka o poslovima koji se dohvate sa servera.

Unutar useEffect hooka, definisana je asinhrona funkcija fetchJobs. Ova funkcija koristi axios biblioteku za slanje HTTP GET zahteva na prosleđeni URL. Cilj ove funkcije je da dohvati podatke o poslovima sa servera. Po uspešnom dohvatanju, dobijeni podaci se postavljaju u stanje jobs koristeći setJobs funkciju. U slučaju greške tokom dohvatanja podataka, greška se beleži u konzoli.

useEffect hook je konfigurisan tako da se asinhrona funkcija fetchJobs poziva svaki put kada se promeni vrednost url argumenta. Ovo osigurava da će, ukoliko se URL promeni, hook ponovo dohvatiti podatke sa novog URL-a.

Na kraju, hook useJobs vraća objekat koji sadrži jobs i setJobs, omogućavajući komponentama koje koriste ovaj hook da pristupe i upravljaju podacima o poslovima.

import { useState, useEffect } from 'react';

import axios from 'axios';

const useJobs = (url) => {

  const [jobs, setJobs] = useState([]);

  useEffect(() => {

    const fetchJobs = async () => {

      try {

        const response = await axios.get(url);

        setJobs(response.data.data);

      } catch (error) {

        console.error('Error fetching jobs:', error);

      }

    };

    fetchJobs();

  }, [url]);

  return { jobs, setJobs };

};

export default useJobs;

Komponenta JoobleListings inicijalizuje niz stanja pomoću Reactovog useState hooka. Stanja uključuju jobPostings (sve dohvaćene poslove), displayedJobs (poslovi koji se trenutno prikazuju), loading (status učitavanja), currentPage (trenutna stranica paginacije) i totalPages (ukupan broj stranica).

Prvi useEffect hook se koristi za asinhrono dohvatanje podataka sa Jooble API-ja. U njemu se definiše asinhrona funkcija fetchData koja šalje POST zahtev na Jooble API koristeći axios. Ova funkcija koristi URL i ključ za pristup API-ju i postavlja određene parametre za zahtev, kao što su ključne reči i limit broja poslova. Nakon dohvatanja podataka, stanje jobPostings se ažurira sa dobijenim poslovima, a totalPages se izračunava na osnovu broja poslova i broja poslova po stranici. U slučaju greške, ispisuje se poruka o grešci, a nakon dohvatanja podataka ili greške, status učitavanja se postavlja na false.

Drugi useEffect hook se koristi za ažuriranje poslova koji se prikazuju na osnovu trenutne stranice. Ovaj hook prati promene u currentPage i jobPostings, izračunavajući koji poslovi treba da budu prikazani.

Funkcija goToPage omogućava navigaciju kroz stranice paginacije.

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

import axios from 'axios';

import './JoobleListings.css';

const JoobleListings = () => {

  const [jobPostings, setJobPostings] = useState([]);

  const [displayedJobs, setDisplayedJobs] = useState([]);

  const [loading, setLoading] = useState(true);

  const [currentPage, setCurrentPage] = useState(1);

  const [totalPages, setTotalPages] = useState(0);

  const itemsPerPage = 5;

  useEffect(() => {

    const fetchData = async () => {

      try {

        const url = "https://jooble.org/api/";

        const key = "22a77a17-c8b0-44ce-b553-006853f5077e";

        const params = {

          keywords: 'it',

          limit: 20,

        };

        const response = await axios.post(url + key, params, {

          headers: {

            "Content-Type": "application/json"

          }

        });

        setJobPostings(response.data.jobs);

        setTotalPages(Math.ceil(20 / itemsPerPage));

      } catch (error) {

        console.error("Error fetching data: ", error);

      } finally {

        setLoading(false);

      }

    };

    fetchData();

  }, []);

  useEffect(() => {

    // Update displayed jobs when currentPage changes

    const startIndex = (currentPage - 1) \* itemsPerPage;

    const endIndex = startIndex + itemsPerPage;

    setDisplayedJobs(jobPostings.slice(startIndex, endIndex));

  }, [currentPage, jobPostings]);

  const goToPage = (pageNumber) => {

    setCurrentPage(pageNumber);

  };

  if (loading) {

    return <div>Loading...</div>;

  }

  console.log(totalPages)

  return (

    <div className="jooble-listings">

      <h2>Job Listings</h2>

      <ul>

        {displayedJobs.map((job, index) => (

          <li key={job.id || index}>

            <h3>{job.title}</h3>

            <p><strong>Company:</strong> {job.company}</p>

            <p><strong>Description:</strong> <span dangerouslySetInnerHTML={{ \_\_html: job.snippet }}></span></p>

            <p><strong>Location:</strong> {job.location}</p>

            <p><strong>Salary:</strong> {job.salary || "Not specified"}</p>

            <p><strong>Source:</strong> {job.source}</p>

            <p><strong>Updated:</strong> {new Date(job.updated).toLocaleDateString()}</p>

            <a href={job.link} target="\_blank" rel="noopener noreferrer">More details</a>

          </li>

        ))}

      </ul>

      <div className="pagination">

      {

        Array.from({ length: totalPages }, (\_, i) => i + 1).map(page => (

          <button

            key={page}

            onClick={() => goToPage(page)}

            className={currentPage === page ? 'active' : ''}

          >

            {page}

          </button>

        ))

      }

    </div>

    </div>

  );

};

export default JoobleListings;

# Korisničko uputstvo

Prilikom pokretanja aplikacije korisnicima izlazi početna stranica preko koje mogu da se uloguju, kao na slici ispod.

Slika na kojoj se nalazi snimak ekrana, tekst, Četvorougao, dizajn

Opis je automatski generisan

U navigacionom meniju postoji opcija za registraciju, ovo je prikazano na slici ispod.

Slika na kojoj se nalazi tekst, snimak ekrana, dizajn

Opis je automatski generisan

Kada se korisnik uloguje otvara mu se stranica na kojoj su prikazani svi poslovi koji postoje u bazi podataka. Na ovoj stranici korisnici mogu da filtriraju poslove po raznim kriterijumima, da ih sortiraju, kao i da menjaju izgled prikaza kartica.

Slika na kojoj se nalazi tekst, snimak ekrana, softver, Font

Opis je automatski generisan

Odabirom opcije Jooble korisnicima se otvara stranica na kojoj su učitani svi poslovi sa Jooble Apija, kao na slici ispod. Na ovoj stranici korisnicima je dostupna opcija paginacije (na dnu).

Slika na kojoj se nalazi tekst, snimak ekrana, softver, broj

Opis je automatski generisan