Sažetak

Tema ovog projekta je bila napraviti web platformu(stranicu) na kojoj se mogu generirati grafovi raznih vrsta I oblika kroz korisničko sučelje, te bloga sa novostima vezanih za tematiku vizualizacije podataka. Projekt je ostvaren sljedećim tehnologijama: HTML, CSS, JavaScript, React.js, Recharts, Bootstrap, Firebase. Ideja projekta je nastala zbog potrebe pojednostavljenja izrade samih grafova bez znanja programiranja, ali sa mogućnostima prilagođavanja raznih parametara. Dio web stranice sa novostima je ostvaren kao blog te služi za obavještavanje I objavljivanje raznih novosti i uputa za izradu. Pisati novosti na blogu mogu svi registrirani i prijavljeni korisnici, dok za grafove nije potrebna prijava.

Uvod I Motivacija

Glavne funkcionalnosti platforme jesu: Izrada I prilagodba Grafova, Upload .csv sa podacima, “Download” istih, te Blog sa novostima(Objavljivanje, Brisanje, Registracija, Prijava). Najveća prednost aplikacije za izradu grafova je upravo u njenoj jednostavnosti. Grafovi se nalaze na gornjem djelu ekrana dok se podaci I parametri za prilagodbu nalaze u donjem djelu ekrana (lijevo podaci, a desno sučelje za prilagodbu). Svaka promjena prikazuje se na grafu odmah, kako se mijenjaju podaci/parametri, kao i svaka promjena tipa grafova. Ovakav način rada znatno olakšava proces kreiranja grafova, radi lakoće upravljanja i prikazivanja podataka. Osim samih grafova, blog sa novostima daje upute kako izraditi određene promjene, objašnjava korištenje parametara koji možda nisu jasni svim korisnicima, te ih potiče na daljnje korištenje aplikacije, te daje bolji dojam interaktivnosti I komunikacije korisnika. Blog dio sa novostima jasno prikazuje naslove članaka sa datumima i autore članaka tako da korisnici mogu otvoriti I čitati njima interesantne članke ukoliko određene žele preskočiti.

Slični grafovi mogu se izrađivati u rješenjima kao što je Microsoft Excel, ali oni nisu toliko intuitivni za korištenje. Glavni konkurent jest web platforma Flourish, koja korisnicima nudi mnogo izbora oko izrade grafova I animiranih grafova, web karti I sl. Glavna prednost nad platformom Flourish jest I dalje jednostavnija izrada grafova (jedan korisnički ekran, dok za platformu Flourish je potrebno izvršiti više koraka prije same vizualizacije), Nedostatak da se unesene promijene odmah vide, nije potrebno se logirati u sustav za korištenje svih značajki, te “download” izrađenih grafova.

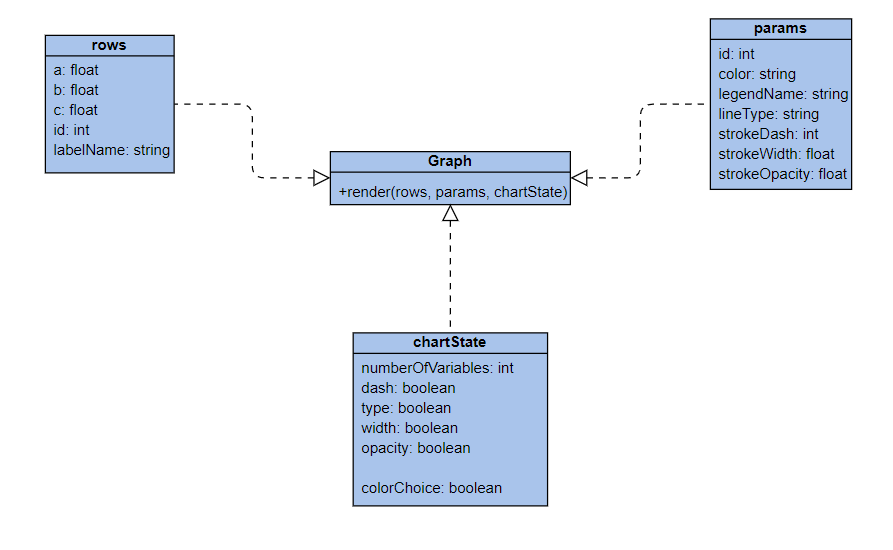
Ciljani korisnici je bilo tko kome treba brza i jednostavna izrada grafova iz postojećih podataka, a ciljano tržište je globalno iz razloga što se svi grafovi mogu prikazati na željeni način/jezik jednostavnom prilagodbom.

**Swot analiza:**

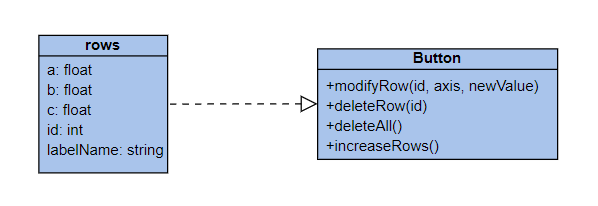


Implementacija

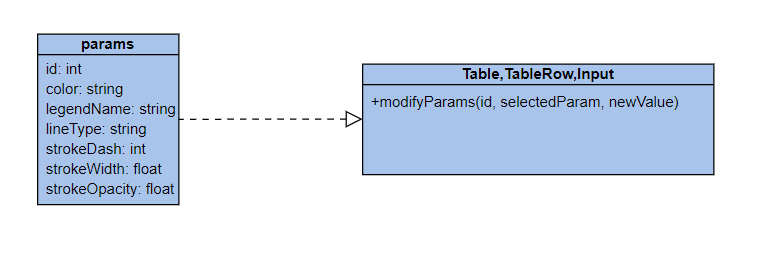
Realizacija grafova(uz pomoć React Hook-ova):



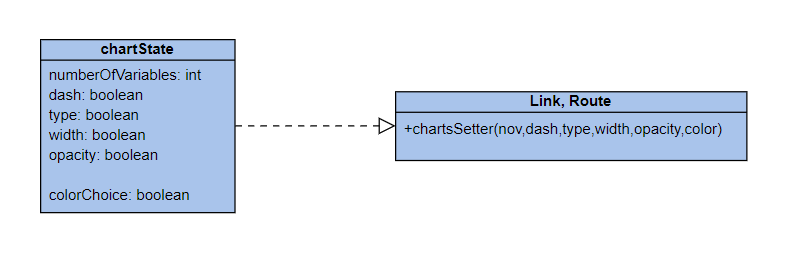
rows:



params:



chartState



Implementacija koda:

Cijela aplikacija izvodi se kao „Single page application“ kroz datoteku index.js.

Index.js:

Redovi(Vrijednosti):

  const [rows, setRows] = useState([{

      id: nanoid(),

      a: '',

      b: '',

      c: '',

      labelName: '',

  }]);

Inicijalizacija redova pomoću useState „Hook-a“. Jedan red predstavlja jedan podatak na grafikonu, koji ima 1,2 ili 3 vrijednosti + Ime. Id reda se dodjeljuje putem nanoId library-a.

const increaseRows = () => {

    const newRow = {

      id: nanoid(),

      a: '',

      b: '',

      c: '',

      labelName: '',

    }

    setRows(rows.concat(newRow));

  }

Funkcija koja dodaje nove redove.

<Col md={{ offset: 2 }}><ButtonComp variant={"primary"} text='Add new row' onClick={increaseRows}/></Col>

Poziv funkcije za dodavanje redova.

const ButtonComp = (props) => {

    return(

      <Button block {...props}>{props.text}</Button>

    )

  }

Gumb je destrukturiran unutar „ButtonComp“ komponente.

  const modifyRow = (id, axis, newValue) => {

    setRows(rows.map(row => {

      if (row.id === id) {

        row[axis] = newValue;

      }

      return row;

    }));

  }

  const deleteRow = (id) => {

    setRows(rows.filter((row) => {

      return row.id !== id;

    }))

  }

  const deleteAll = () => {

    setRows([]);

  }

Funkcije za izmjenu, i brisanje (pojedinog i svih redova)

<td className="table-icons" onClick={deleteRow}><BsFillTrashFill  className="react-icons"/></td>

Poziv funkcije za brisanje pojedinog reda.

<Col md={{ offset: 0 }}><ButtonComp variant={"danger"} text='Delete Table'  onClick={deleteAll} /></Col>

Poziv funkcije za brisanje svih redova.

<Container className="fixed scroll">

   <TableComp rows={rows} deleteRow={deleteRow} modifyRow={modifyRow} chartState={chartState}/>

</Container>

Tablica redova.

const TableComp = ({rows, deleteRow, modifyRow, chartState}) => {

    if(rows < 1){

      return(

        <h2 className="center">Press "add new row" to start visualizing!</h2>

      )

    }

    switch(chartState.numberOfVariables) {

      case 1:

        return(

          <Table striped bordered hover size="sm">

            <tbody>

              <tr>

                <th>1st Value</th>

                <th>Name</th>

                <th className="table-icons"></th>

              </tr>{

                rows.map(row => <TableRow key={row.id} deleteRow={() => deleteRow(row.id)} chartState={chartState}

                modifyRow={(axis, newValue) => modifyRow(row.id, axis, newValue)} row={row}/>)

              }

            </tbody>

          </Table>

        )

„TableComp“ komponenta u kojem je fixirano zaglavlje tablice, te su redovi dodatno dekomponirani u komponentu „TableRow“, kako bi se mogli mapirati skupa sa točnim pozivima funkcija. Primjer je za red sa tri vrijednosti.

const TableRow = ({deleteRow, modifyRow, chartState, row}) => {

  switch (chartState.numberOfVariables){

    case 1:

      return(

        <tr>

          <td><input className="input-style" type="number" value={row.a} placeholder="enter 1st value" onChange={(e) => modifyRow('a', e.target.value)}/></td>

          <td><input className="input-style" type="text" value={row.labelName} placeholder="enter name" onChange={(e) => modifyRow('labelName', e.target.value)}/></td>

          <td className="table-icons" onClick={deleteRow}><BsFillTrashFill  className="react-icons"/></td>

        </tr>

      )

„TableRow“ komponenta, s pozivima funkcija.

Parametri grafikona:

  const [params, setParams] = useState(

    [

        {

          id: 1,

          color: "#111222",

          legendName: 'a',

          lineType: "monotone",

          strokeDash: 0,

          strokeWidth: 1,

          strokeOpacity: 1,

        },

        {

          id: 2,

          color: "#222555",

          legendName: 'b',

          lineType: "monotone",

          strokeDash: 0,

          strokeWidth: 1,

          strokeOpacity: 1,

        },

        {

          id: 3,

          color: "#333666",

          legendName: 'c',

          lineType: "monotone",

          strokeDash: 0,

          strokeWidth: 1,

          strokeOpacity: 1,

        }

    ]

  )

Inicijalizacija Parametara i postavljanje default stanja. Parametri služe kako bi se na grafikonima mogli promijeniti: Širina, Boja, Ime na legendi, Vrstu linije, Crtice, Širinu i prozirnost.

  const modifyParams = (id, selectedParam, newValue) => {

    setParams(params.map(param => {

      if (param.id === id) {

        param[selectedParam] = newValue;

      }

      return param;

    }));

  }

Funkcija za modifikaciju parametara grafikona

  const [chartState, setChartState] = useState({

    numberOfVariables: 3,

    dash: true,

    type: true,

    width: true,

    opacity: true,

    colorChoice: true

  })

S obzirom da svaki grafikon sadrži svoje parametre(nemaju svi iste), „chartState“ čuva parametre od odgovarajućih grafikona (npr. Bar Chart nema mogućnost mijenjanja izbora linije). Nakon klika na link željenog grafikona poziva se funkcija odgovorna za svaki grafikon i broj vrijednosti. Linkovi na grafikone se nalaze u komponentama „Cards.js“ i „SubNavBar.js“, te su im proslijeđene refence na funkcije koje postavljaju „chartState“.

<Nav.Link as={Link}  to="/three/threescatter" onClick={()=>chartsSetter({nov:3,dash:false,type:false,width:true,opacity:true,color:true})}><b>Scatter Chart</b></Nav.Link>

Primjer linka na „Scatter chart“ sa 3 varijable. Poziv funkcije koja postavlja „chartState“.

const chartsSetter = ({nov,dash,type,width,opacity,color}) =>

   setChartState({

     numberOfVariables: nov,

     dash: dash,

     type: type,

     width: width,

     opacity: opacity,

     colorChoice: color

   })

Funkcija koja postavlja „chartState“.

<ParameterCustomization params={params} modifyParams={modifyParams} chartState={chartState}/>

Komponenta za modificiranje parametara.

const ParameterCustomization = ({params, modifyParams, chartState}) => {

    return(

      <Container fluid>

      <Row>

        <Col>

        {

          params.filter(param => param.id <= chartState.numberOfVariables).map(param => <InputComp key={param.id}

            modifyParams={modifyParams} param={param}/>)

        }

Komponenta „InputComp“ za mijenjanje boja, unutar komponente „ParameterCustomization“, dodatno su destrukturirane komponente za ostale parametre(WidthSelect,OpacitySelect,LegendName,DashSelect) na isti način kao i primjer „InputComp“.

const InputComp = ({modifyParams, param}) => {

    return(

      <>

        <input type="color" name="color"

        onChange={e => modifyParams(param.id, 'color', e.target.value)}/>

      </>

    )

}

Komponenta „InputComp“ omogućuje mijenjanje boja na grafikonu.

Download grafikona:

var handleSaveClick = (idOfChart) => {

    domtoimage.toBlob(document.getElementById(idOfChart))

       .then(function (blob) {

          fileDownload(blob, 'dom-to-image.png');

       });

   }

Funkcija za download grafikona. Ostvarena uz pomoć 2 dodatna library-a: domtoimage i fileDownload

<div id="blob1">

Za download potrebno je dodijeliti id grafikonu.

<Col md={{ offset: 0 }}><ButtonComp variant={"primary"} text='Download Chart' onClick={() => handleSaveClick("blob1")}/></Col>

Poziv funkcije za download „handleSaveClick“ i proslijeđivanje id-a grafikona. Bitno je da se umjesto poziva funkcije, pozove referenca, u suprotnom bi svaki re-render Button komponente pozvao funkciju, a u ovom slučaju to ne želimo.

Grafikoni:

<Container className="marginTop">

              <ResponsiveContainer className="justify-content-md-center">

                <Row>

                  <LineChart width={1400} height={350} data={rows}

                  margin={{ top: 5, right: 100, left: 5, bottom: 5 }}>

                  <CartesianGrid strokeDasharray="3 3" />

                  <XAxis dataKey="labelName"/>

                  <YAxis />

                  <Tooltip />

                  <Legend />

                  <Line strokeWidth={params[0].strokeWidth} name={params[0].legendName}

                   strokeDasharray={params[0].strokeDash} type={params[0].lineType} dataKey="a" stroke={params[0].color}/>

                  <Line strokeWidth={params[1].strokeWidth} strokeLinecap={params[1].strokeLinecap} name={params[1].legendName}

                  strokeDasharray={params[1].strokeDash} type={params[1].lineType} dataKey="b" stroke={params[1].color} />

                  <Line strokeWidth={params[2].strokeWidth} strokeLinecap={params[2].strokeLinecap} name={params[2].legendName}

                  strokeDasharray={params[2].strokeDash} type={params[2].lineType} dataKey="c" stroke={params[2].color} />

                  </LineChart>

                </Row>

              </ResponsiveContainer>

            </Container>

Komponenta „LineChart“ iz „Recharts“ library-a odgovorna je za renderiranje u ovom primjeru „Linijskog grafikona“, drugi grafikoni ostvareni su na sličan način. U „LineChart“ komponentu podaci se proslijeđuju kroz „props-e“ (data={rows}), koji se mijenjaju na prije definiran način.

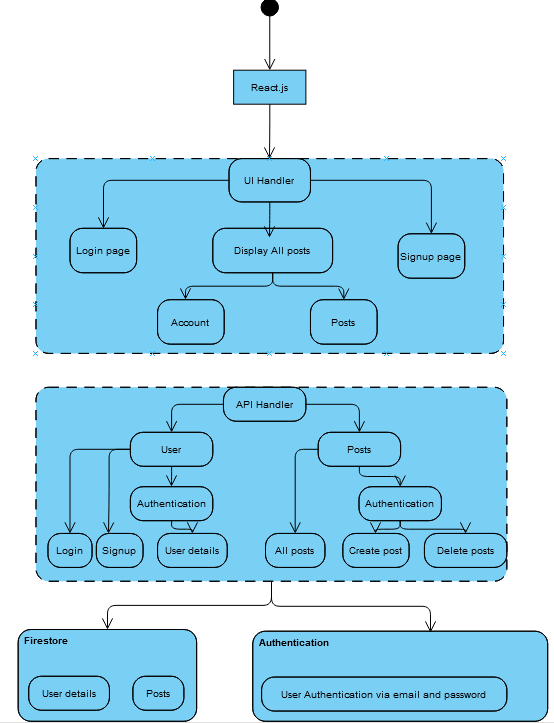
„CartesianGrid“ - komponenta za renderiranje pozadinske mreže na grafikonima.

„XAxis“ i „YAxis“ - komponente za renderiranje X i Y osi.

„Legend“ – komponenta za legendu, nalazi se ispod grafikona

„Line“ – komponenta koja renderira liniju na linijskom grafikonu, u ovom slučaju radi se o linijskom grafikonu s 3 linije. Putem „dataKey“ parametra proslijeđuju se vrijednosti podataka. Ostali parametri služe za prilagođavanje grafikona po želji.

Implementacija „Blog“ dijela aplikacije - Activity dijagram



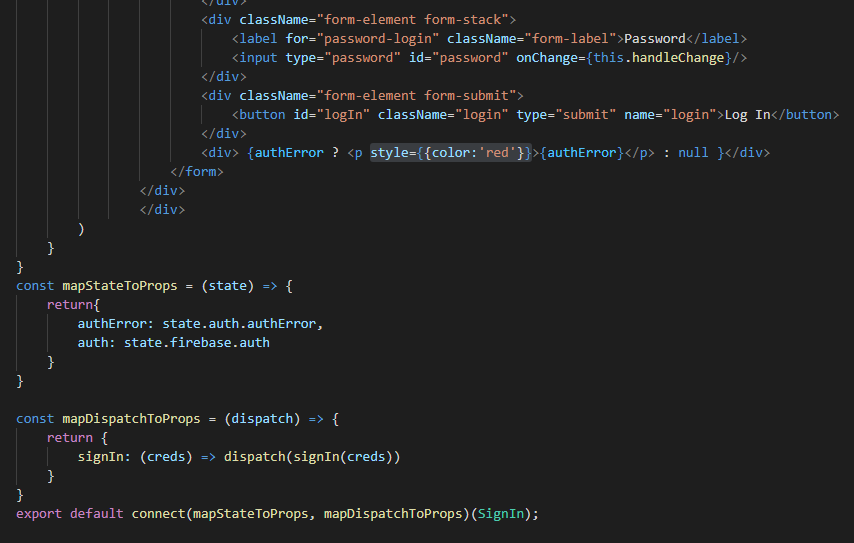
Implementacija koda:

„Blog“ dio aplikacije se može podijeliti u 3 dijela, dio za registraciju/logiranje korisnika, dio za prikazivanje svih članaka i dio za kreiranje novih članaka. Pohranu članaka i registracija korisnika se odvija na Firebase-u. Firebase je platforma koju je razvio Google za stvaranje mobilnih i web aplikacija, kategorizirana je kao NoSQL baza podataka koja pohranjuje podatke u dokumentima sličnima JSON-u.

Za povezivanje React-a s Firebase-om, potrebno je koristiti Redux. Redux je predvidivi kontejner za stanje JS aplikacija. To je jako mala biblioteka koja zahtjeva određen način pisanja koda koji je vezan za stanje aplikacije. State je ustvari JS objekat, koji je read only. Redux nameće način izmjene ovog objekta tako da sve informacije idu u jednom smijeru i ne izazivaju mutacije. Promjena stanja se inicira pomoću akcije, a reducer izvršava samu promjenu.

Kod za registraciju i logiranje korisnika:

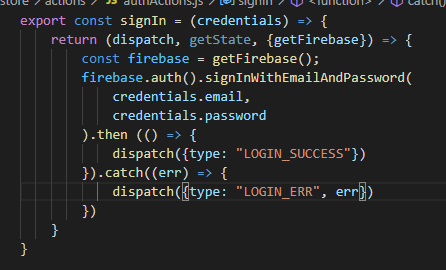




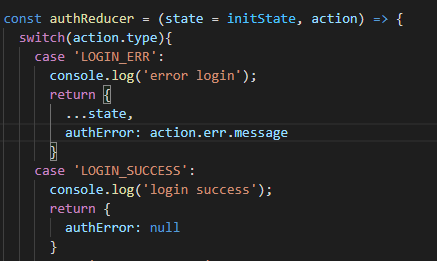
Klasa SignIn koja se koristi za logiranje korisnika, traži od korisnika njegov e-mail i lozinku. Pomoću funkcije handleChange, korisnikov upis se sprema u state (stanje) i pomoću funkcije handleSubmit, informacije iz state se „pošalju“ na funkciju signIn.

mapStateToProps () *je uslužni program koji pomaže vašoj komponenti da ažurira stanje (koje ažuriraju neke druge komponente).* U navedenom kodu, mapStateToProps() služi za provjeravanje ispravnosti mail adrese i lozinke.

mapDispatchToProps () *je uslužni program koji će vašoj komponenti pomoći da pokrene akcijski događaj (dispečerska radnja koja može uzrokovati promjenu stanja aplikacije).* U navedenom kodu, mapDispatchToProps() uzima korisnikove upisane podatke te ih šalje funkciji signIn.



Funkcija signIn uzima korisnikove podatke te ih šalje na firebase koja zajedno s funkcijom signInWithEmailAndPassword provjerava postojanje te iste mail adrese i logira korisnika na stranicu.

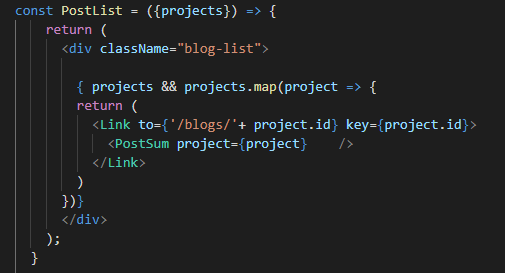


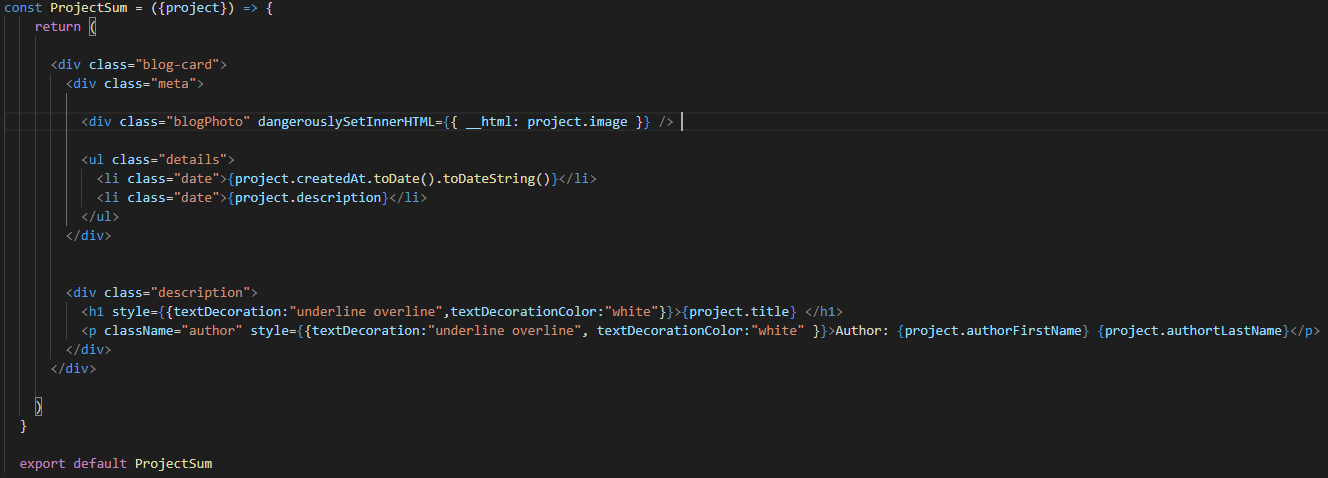
AuthReducer koja je povezana s funkcijom signIn preko types (LOGIN\_SUCCESS and LOGIN\_ERR) služi za provjeru točnosti mail adrese I lozinke. Poruke o neuspješnoj validaciji dolaze od Firebase-a koja pomoću action.err.message korisniku izbacuje određenu poruku greške prilikom logiranja na aplikaciju ( na primjer: lozinka mora sadržavati 6 slova, nije ispravna mail adresa…)

Sličan kod s dodatnim parametrima za korisnikov upis ( ime I prezime korisnika) se nalazi na funkciji signUp. Logika koda s povezivanjem na Firebase I provjeravanjem ispravnosti upisane mail adrese I lozinke je ista kao što je opisano.

Drugi dio “Blog” dijela aplikacije je prikazivanje svih članaka.

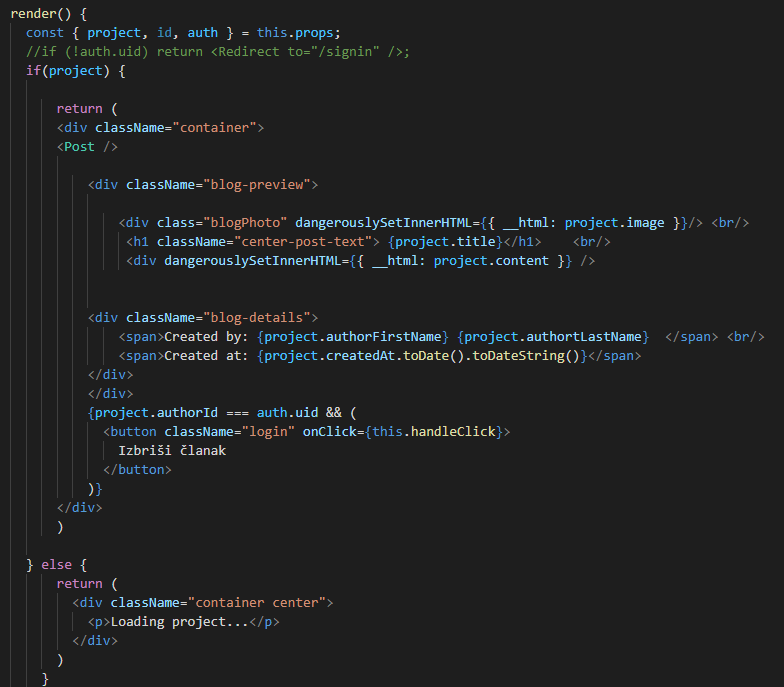
U klasi PostList se prenosi props “projects” koja s map funkcijom prolazi kroz listu podataka te pridružuje svaki project.id uz točan članak koji se ispisuju na dio bloga gdje korisnik može vidjeti sve članke.



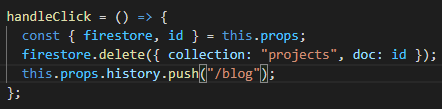


U const ProjectSum, preko props project, I prethodne map funkcije koja je svaki članak povezala s odgovarajućim id s Firebase-a, određujemo što će se sve prikazivati od podataka spremljenih za određeni članak. Određeno je da će se u card stilu prikazivati naslov članka, slika, autor, datum I kratki sažetak o tom članku.

Svaka slika na Firebase je spremljena kao URL, te za pretvaranje iz URL u sliku koja se prikazuje na stranici, korištena je funkcija dangerouslySetInnerHTML={{ \_\_html: project.image }}

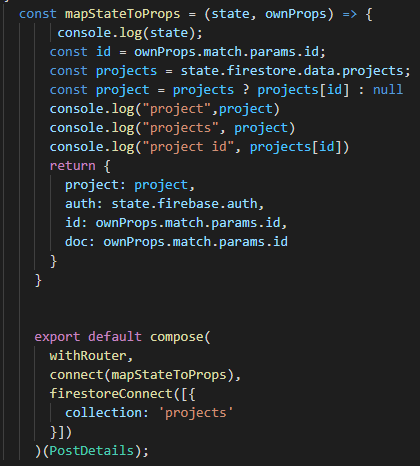


Klasa PostDetails služi kako bi se svaki članak iz liste svih članak otvorio I pročitao. Pomoću props projects, dohvaća se određeni podatak iz baze podataka na određeno mjesto s određenim css atributom. .



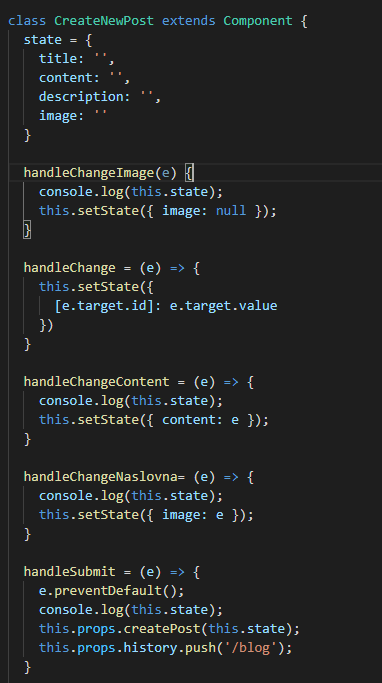
Dodatno je napravljena mogućnost da korisnik koji je napisao članak može isti I izbrisati.

U ovoj klasi je korišten samo mapStateToProps koji nam omogućava da se spojimo na Firebase-ov Firestore (realtime baza podataka) I određenu “collection” koju smo nazvali “projects” kako bi mogli dohvatiti točne podatke s baze.



Treći dio je kreiranje članaka.

U klasi CreateNewPost imamo 4 state (title, content, description, image) koje korisnik u određenoj form unosi te se svaki taj unos zabilježi pomoću funkcija handleChange() koja ažurira navedene states. handleSubmit je zaslužan da se svako to ažuriran states pošalje na bazu preko funkcije createPost.





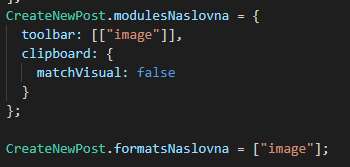
Ukoliko korisnik koji nije logiran pristupi izradi članaka, stranica će ga preusmjeriti na stranicu za logiranje jer samo korisnik koji je ulogiran može kreirati članke.

Već navedeno, korisnik upisuje tekst u formi, gdje navodi naslov članka, kratak opis članka koji se prikazuje na popisu svih članaka, cijeli tekst članka te naslovnu sliku.

Kako bi korisnik mogao urediti tekst ( staviti bold font, promijeniti font, veličinu slova…) korištena je React-ova komponenta ReactQuill što je rich-text editor. Pomoću ovog editora, korisnik može ubacivati slike I mijenjati tekst po svojoj želji.



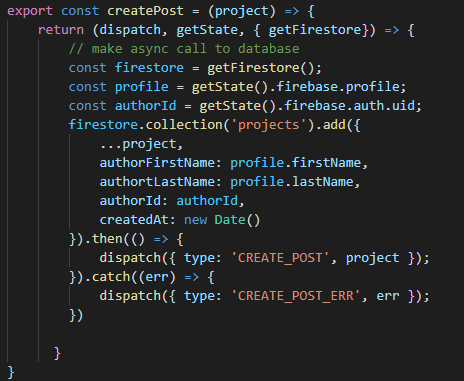
Preko određenih ReactQuill modules I formats određujemo što će se u svakom editor prikazivati te možemo dodavati dodatne modules kako bi korisnik imao više opcija za uređivanje.



Za ubacivanje naslovne slike, također je korišten ReactQuill editor ali je ograničen da korisnik može ubaciti samo sliku.



Ponovno su korišteni mapStateToProps I mapDispatchToProps koji povezuju ulogiranog korisnika s njegovim profilom ( upisanim podacima, ime I prezime koje se kasnije automatski sprema I prikazuje kao autor članka) te se funkcijom createPost.



Const createPost uzima props project iz ažuriranih states iz klase CreateNewPost te zajedno s profilom korisnika I njegovim ID I datumom kreiranja se sprema na bazu pod collection “projects”.

Korisničke upute – GitHub ( <https://github.com/JosipTadic/IIP2021> ) readme!