Aplikacija za organizaciju događaja

Bašek, Marko

Sveučilište u Rijeci, Fakultet informatike i digitalnih tehnologija

Sveučilišni prijediplomski studij Informatika

Marko Bašek

Aplikacija za organizaciju događaja

Završni rad

Mentor: doc. dr. sc. Lucia Načinović Prskalo

Rijeka, 30.01.2024.

Sveučilište u Rijeci
FIDIT Fakultet informatike i digitalnih tehnologija
www.inf.uniri.hr

Rijeka, 28.4.2023.

Zadatak za završni rad

Pristupnik: Marko Bašek

Naziv završnog rada: Aplikacija za organizaciju događaja

Naziv završnog rada na engleskom jeziku: Application for event management

Sadržaj zadatka: Zadatak završnog rada je izraditi aplikaciju za organizaciju događaja koja omogućava pregledni prikaz događaja, stvaranje novih događaja i filtriranje događaja po zadanim kriterijima. U radu će se izraditi klijentska i poslužiteljska strana aplikacije te će se opisati tehnologije koje su se koristile prilikom izrade aplikacije i funkcionalnosti koje aplikacija nudi.

Mentor

Doc. dr. sc. Lucia Načinović Prskalo

Voditelj za završne radove

Doc. dr. sc. Miran Pobar

Zadatak preuzet: 23.02.2024.

(potpis pristupnika)

Adresa: Radmile Matejčić 2
51000 Rijeka, Hrvatska
Tel: +385(0)51 584 700
E-mail: ured@inf.uniri.hr
OIB: 64218323816
IBAN: HR1524020061400006966

Sadržaj

Sažetak

U ovom završnom radu u uvodnom se dijelu predstavlja ideja koja je poslužila kao temelj za izradu aplikacije. Zatim se fokus prebacuje na tehnologije koje su ključne za izgradnju aplikacije, Firebase i React. Prikazuje se njihova osnovna uloga i teorijski aspekti koji su odabrani kako bi se postigle željene funkcionalnosti. U završnom radu također se opisuju bitne značajke tih tehnologija zajedno sa snimkama zaslona koji prikazuju kako se koriste u stvarnom projektu. Slijedi opis dizajna aplikacije - odabir boja, interaktivnost aplikacije i korisnika, analiziraju se stilizacijske značajke povezane s dizajniranjem komponenata. Na samom kraju rada, analizira se kod kako bi se istaknule ključne funkcionalnosti na kojima se temelji završni rad. Fokus je stavljen na aspekte kao što su povezivanje s bazom podataka, CRUD[1] operacije, autentifikacija korisnika i sustav filtriranja.

Ključne riječi: aplikacija za organizaciju događaja, web aplikacija, Firebase, React, baza podataka, autentifikacija, Firebase Storage, dizajn korisničkog sučelja, CSS Module, styled-components

[1] Skraćenica za kreiranje, čitanje, ažuriranje, brisanje (eng. create, read, update, delete).

1. Uvod

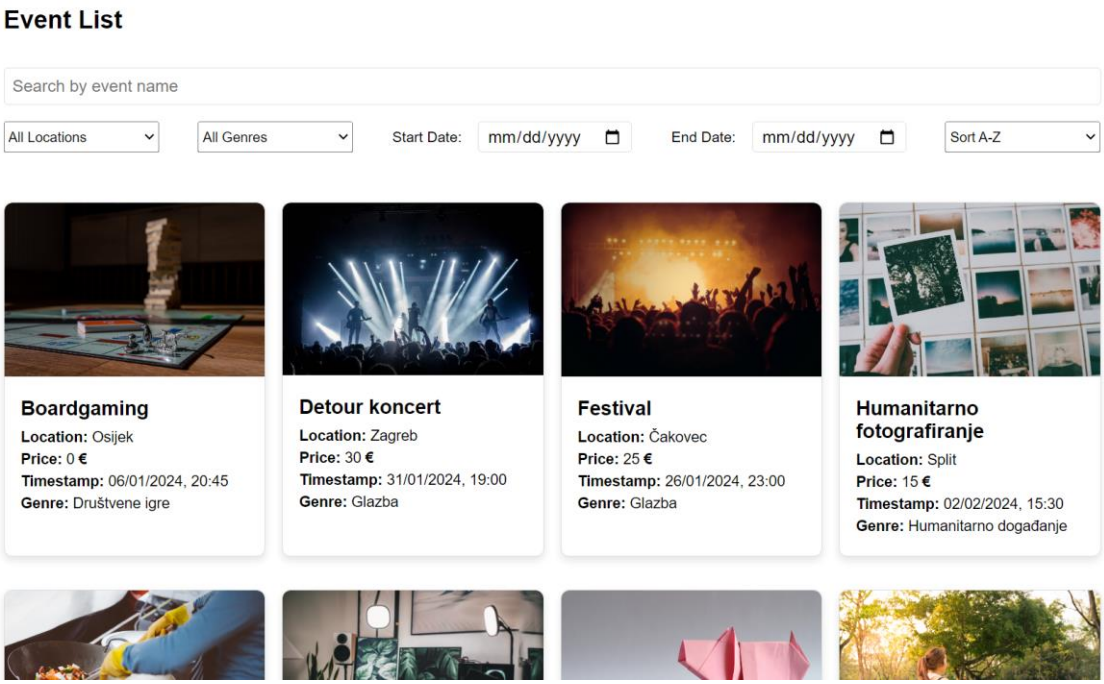
Završni rad opisuje aplikaciju za organizaciju događaja, opisuje razvoj i implementaciju. Fokus je usmjeren na poboljšanje dostupnosti aplikacije i lakše pretraživanje događaja. Danas vlada obilje događaja različitih žanrova koji se nalaze na različitim mjestima. Ideja same aplikacije je omogućavanje što jednostavnijeg pronalaženja, kreiranja, uređivanja i filtriranja specifičnih događaja u korisnikovoj blizini. Završni rad obuhvaća kompletan opis razvoja aplikacije, uključujući dizajn korisničkog sučelja, funkcionalnosti za stvaranje i upravljanje događajima te optimizirane mehanizme pretraživanja.

Kod izrade aplikacije korišten je React i Firebase platforma. React je popularan JavaScript okvir (eng. *framework*) koji je pružio temelj za dinamičko i učinkovito korisničko sučelje. Firebase platforma je s nizom alata kao što su Firebase Storage, Firebase Auth i Firestore pružila robusnu back-end[2] infrastrukturu. Firebase Storage omogućuje sigurno pohranjivanje datoteka, dok Firebase Auth upravlja autentifikacijom korisnika. Firestore, kao NoSQL[3] baza podataka, omogućuje učinkovito upravljanje podacima.

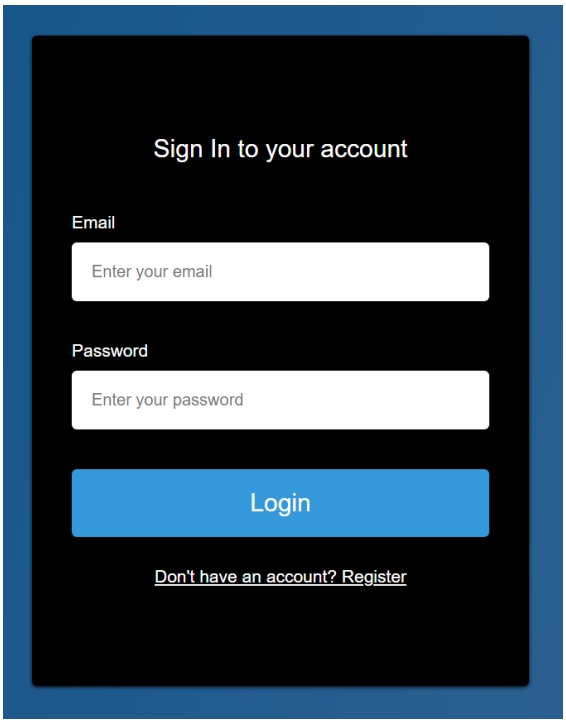
Aplikacija omogućuje korisnicima registraciju i prijavu (prikazano na slici 2) u aplikaciju na intuitivan način. Korisnici mogu pregledati sve trenutne događaje, uključujući njihove detalje, čak i ako nisu prijavljeni u aplikaciju. Također mogu pretraživati događaje pomoću tražilice ili postavljati filtere za olakšano pretraživanje (prikazano na slici 1). Prijava u aplikaciju omogućava korisnicima kreiranje vlastitih događaja (prikazano na slici 3), uređivanje i brisanje.

[2] Dio aplikacije koji se bavi obradom podataka i komunikacijom baze podataka, strana aplikacije koja nije vidljivija korisnicima.

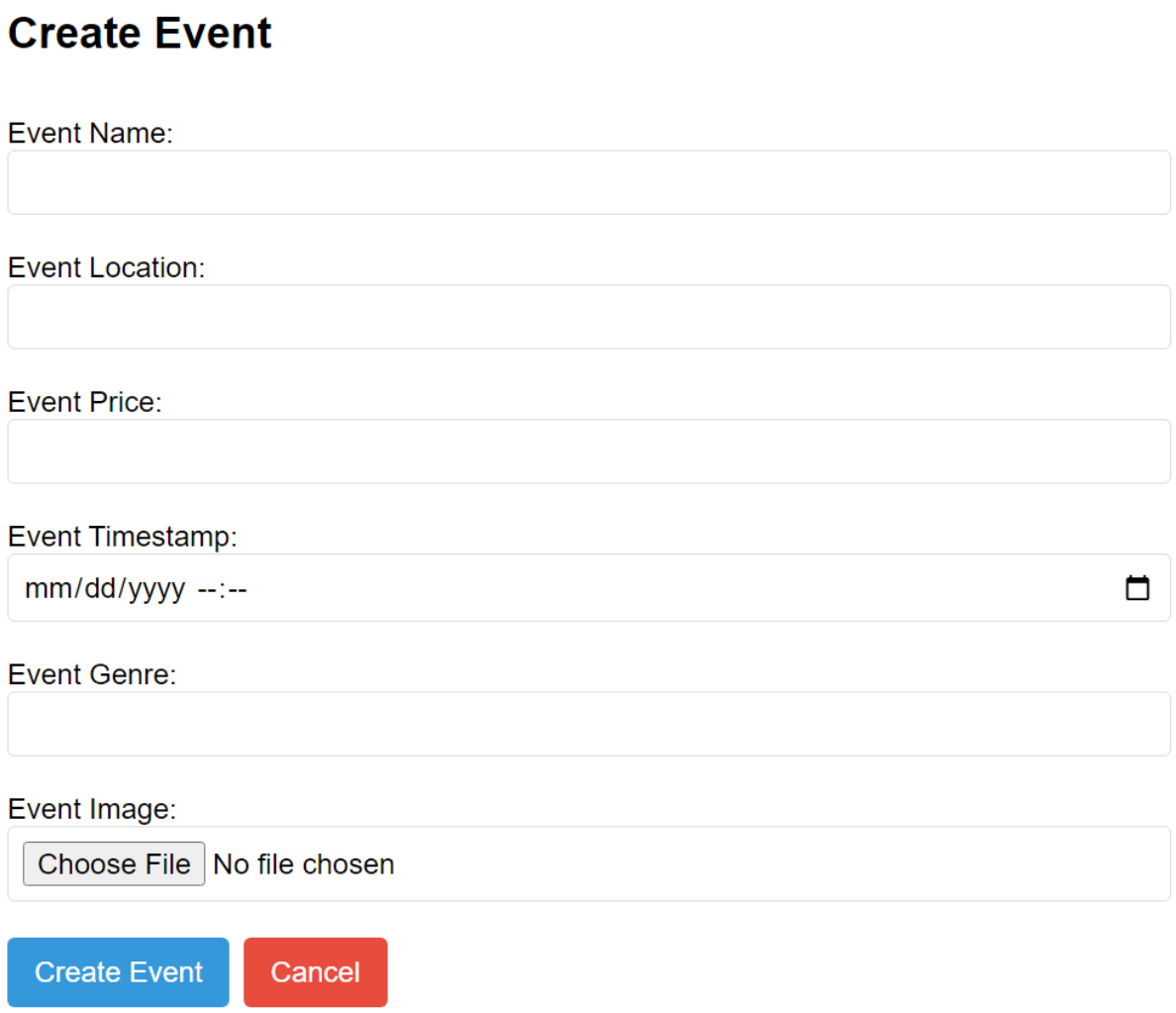
[3] Nerelacijska baza podataka koja koristi fleksibilan model sheme. Podržava nestrukturirane podatke i ne koristi tablični format za pohranjivanje podataka kao relacijska baza podataka.



Slika 1 Prikaz svih događaja s naprednim mogućnostima pretraživanja i filtriranja



Slika 2 Prijava u aplikaciju



Slika 3 Kreiranje događaja

2. Firebase

Firebase je Google-ova platforma koja olakšava razvoj i upravljanje mobilnih i web aplikacija (geeksforgeeks, 2024.). Skup je alata pozadinskih usluga u računalnom oblaku koja pomaže u izgradnji, implementaciji skaliranju aplikacija. Nudi razne mogućnosti koje omogućuju brže i sigurnije načine izrade aplikacije. Uključuje integraciju za iOS, Android, Java, JavaScript, Node.js, C++, PHP i Unity (Wikipedia, 2024.).

2.1 Povijest

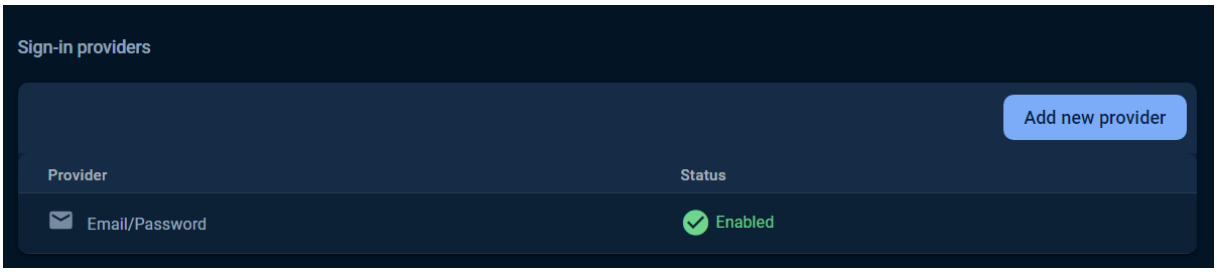
U svojim počecima tvrtka je nosila naziv Envolve, a osnovali su je James Tamplin i Andrew Lee. Njihova prvotna inicijativa bila je omogućiti programerima integraciju značajke online chata na njihove web stranice putem API-ja4. No, kako se razvijala, Envolve je doživio transformaciju koja je rezultirala stvaranjem Firebase platforme. Tamplin i Lee shvatili su da se njihova usluga chata ne koristi samo za razmjenu poruka, već i da je programeri igara koriste za prosljeđivanje stanja igara između igrača. Programeri su počeli koristiti Envolve za sinkronizaciju podataka aplikacija u stvarnom vremenu, omogućujući prijenos informacija o stanju igre među korisnicima u stvarnom vremenu. Osnivači Envolve-a donijeli su ključnu odluku o razdvajanju arhitekture u stvarnom vremenu, koja je upravljala chatom, od sustava koji je služio za chat. Tako je 2011. godine nastao Firebase kao samostalna tvrtka, a javnosti je predstavljen 2012. godine. Prva usluga koju je Firebase ponudio bila je Firebase Realtime Database, inovativna platforma za pohranu podataka u oblaku. Ova usluga omogućila je sinkronizaciju podataka aplikacija na Androidu, iOS-u i web uređajima, označavajući početak Firebase-ove posvećenosti pružanju učinkovitih rješenja za upravljanje podacima u stvarnom vremenu (Wikipedia, 2024.).

2.2 Firebase značajke i funkcionalnosti

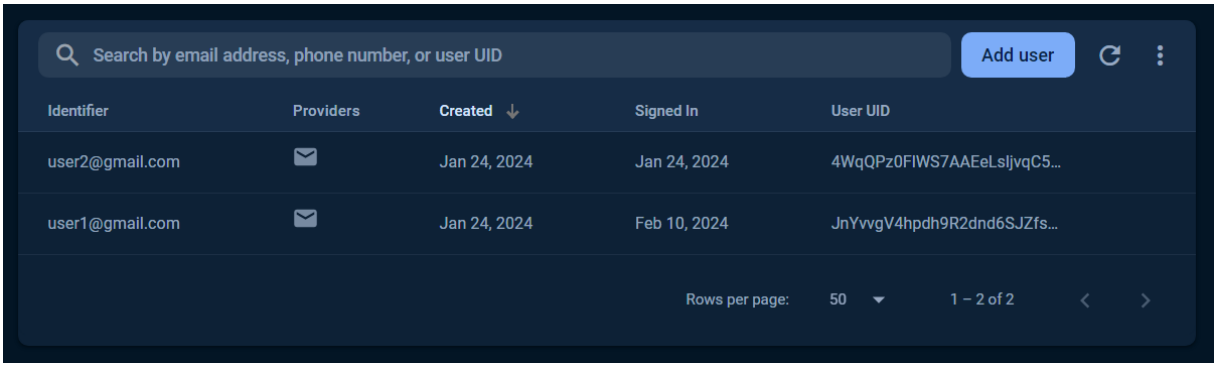
2.2.1 Autentifikacija

Zahvaljujući Firebase-ovoj usluzi autentifikacije, korisnici mogu s lakoćom i sigurnošću pristupiti svojim aplikacijama putem različitih metoda prijave. Ova usluga omogućava korisnicima da se prijavljuju pomoću svojih računa na popularnim platformama kao što su Google, Facebook, Apple, GitHub, Microsoft, Twitter… Na slici 4 prikazana je omogućena prijava email-om i lozinkom. Na slici 5 prikazani su kreirani računi korisnika na Firebase-u (TechTarget, 2024.).

[4] Aplikacijsko programsko sučelje, odnosno skup određenih pravila koje programeri slijede i koje koriste kod povezivanja s određenom aplikacijom



Slika 4 Metoda prijave Firebase autentifikacije



Slika 5Autentificirani korisnici

2.2.2 Baza podataka u stvarnom vremenu (eng. Realtime Database)

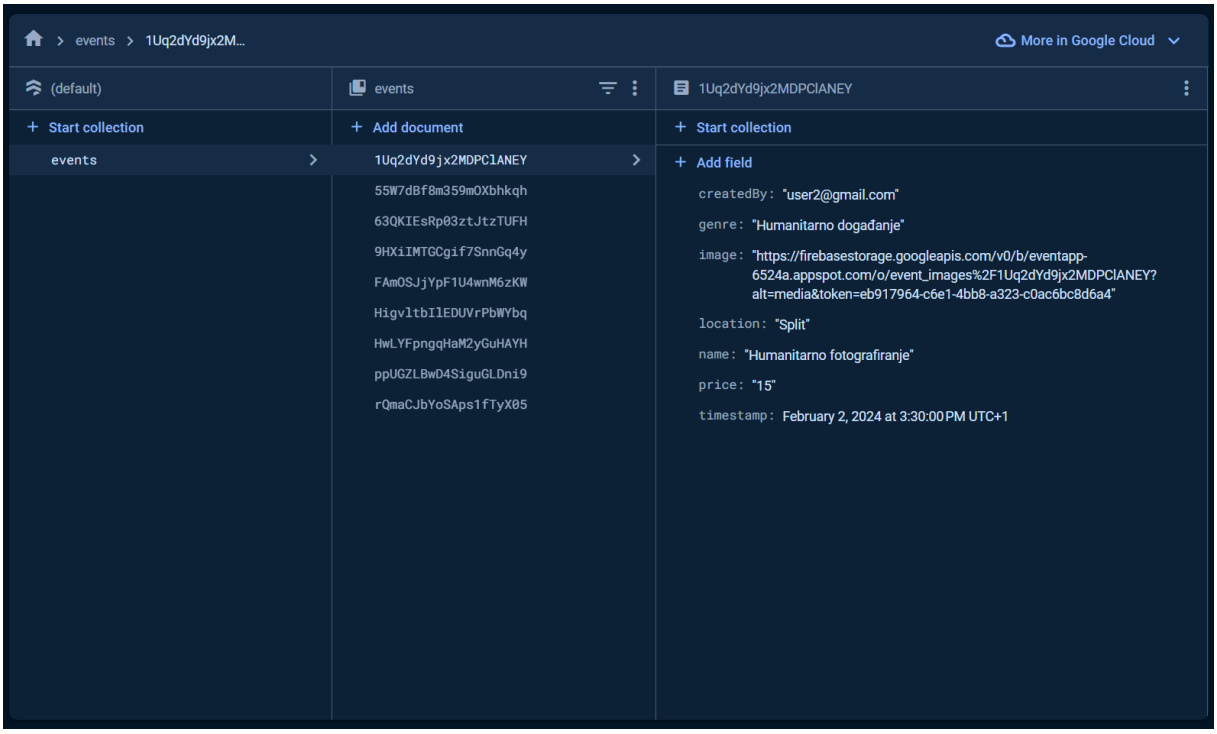
Firebase Realtime Database, kao NoSQL baza podataka u oblaku, pruža tvrtkama mogućnost pohrane i sinkronizacije podataka u stvarnom vremenu na svim uređajima njihovih korisnika. Ova napredna tehnologija značajno pojednostavljuje proces izrade aplikacija koje ostaju ažurne čak i kada korisnici nisu online (TechTarget, 2024.).

2.2.3 Cloud Firestore

Cloud Firestore je NoSQL baza podataka u oblaku za razvoj mobilnih, web i poslužiteljskih aplikacija koja se temelji na Google Cloud infrastrukturi i služi za sinkronizaciju i pohranu na strani klijenta i poslužitelja. U usporedbi s Firebase Realtime Database, Cloud Firestore održava sinkronizaciju podataka u klijentskim aplikacijama putem slušatelja u stvarnom vremenu. Nudi izvanmrežnu podršku za mobilne uređaje i web, omogućuje izradu responzivnih5 aplikacija

[5] Aplikacije koje se prilagođavaju različitim veličinama zaslona.

neovisno o latenciji mreže ili internetske povezanosti. Cloud Firestore pruža integraciju s drugim Firebase i Google Cloud proizvodima, uključujući Cloud Functions. Slika 6 prikazuje kolekciju događaja (eng. events) u Firestore-u (Firebase, 2024.).



Slika 6 Kolekcija "events" u Cloud Firestore-u

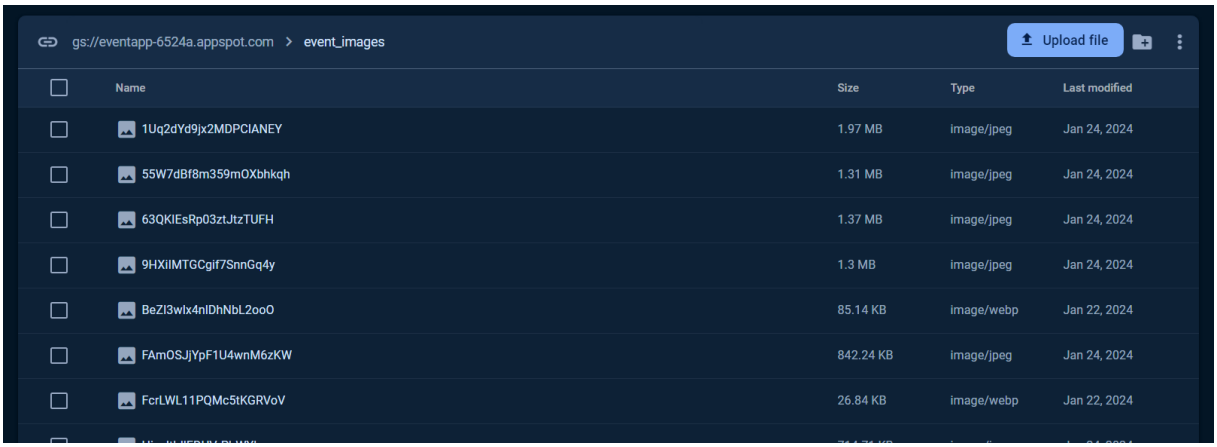
2.2.4 Cloud Messaging

Tvrtke mogu slati poruke na pametne telefone svojih korisnika uz Firebase Cloud Messaging (FCM), čak i kada ne koriste aplikaciju. Pomoću FCM-a programeri imaju mogućnost dinamičkog mijenjanja sadržaja aplikacije te izdavanja „push[6]“ obavijesti. Ovaj alat otvara vrata raznim interakcijama s korisnicima, omogućujući personalizirane obavijesti, ažuriranja ili promocije (TechTarget, 2024.).

[6] Poruke koje se šalju kako bi se korisnici obavijestili o određenim događajima, informacijama ili ažuriranjima iz aplikacija, web stranica ili sustava.

2.2.5 Storage

Pohrana u oblaku za Firebase temelji se na iznimno brzoj i sigurnoj infrastrukturi Google Clouda, pružajući programerima aplikacija pouzdanu platformu za pohranu i posluživanje korisnički generiranog sadržaja, uključujući fotografije i videozapise. Cloud Storage za Firebase predstavlja moćan, intuitivan i ekonomičan servis za pohranu objekata izgrađen uzimajući u obzir Googleovu vrhunsku skalabilnost. Firebase SDK-ovi[7] za pohranu u oblaku dodaju slojeve Googleove sigurnosti tijekom prijenosa i preuzimanja datoteka za Firebase aplikacije, neovisno o kvaliteti mreže. Slika 7 prikazuje spremljene slike u Firebase Storage-u (Firebase, 2024.).



Slika 7 Datoteke u Firebase Storage-u

2.2.6 Firebase sigurnosna pravila (eng. Firebase Security Rules)

Firebase sigurnosna pravila (eng. Firebase Security Rules) služe za zaštitu podataka u Cloud Firestore-u, Firebase Realtime Database-u i Cloud Storage-u. Također predstavljaju barijeru između vlastitih podataka i potencijalno zlonamjernih korisnika. Komponenta omogućuje definiranje pravila, jednostavnih ili složenih, prilagođenih specifičnostima aplikacije kako bi se osigurala zaštita podataka na odgovarajućoj razini propusnosti koju aplikacija zahtijeva. Za ostvarivanje pristupa bazi podataka u stvarnom vremenu, Cloud Firestore-u i Cloud Storage-u

[7] Skup alata, biblioteka i dokumentacija koje omogućuju programerima razvoj softverskih aplikacija za određenu platformu, operativni sustav ili programski jezik.

koriste se proširivi i fleksibilni jezici koji omogućuju precizno definiranje pravila pristupa podacima s korisničke strane. Kod pristupa bazi podataka u stvarnom vremenu, definicije pravila koriste JSON8, dok je za Cloud Firestore i Cloud Storage korišten jedinstveni jezik prilagođen složenijim strukturama specifičnih pravila.

Sigurnosna pravila Firebase Storage-a (prikazano na slici 8) omogućuju čitanje i pisanje samo autentificiranim korisnicima, dok pravila za bazu podataka Firestore (prikazano na slici 9) omogućuju svim korisnicima čitanje podataka, a zapis u bazu omogućuju samo autentificiranim korisnicima (Firebase, 2024.).

```
rules_version = '2';

// Craft rules based on data in your Firestore database
// allow write: if firestore.get(
//    /databases/(default)/documents/users/$(request.auth.uid)).data.isAdmin;
service firebase.storage {
  match /b/{bucket}/o {
    match /{allPaths=**} {
      allow read, write: if request.auth != null;
    }
  }
}
```

Slika 8 Firebase Storage sigurnosna pravila

```
rules_version = '2';

service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read: if true;
    }
    match /{document=**} {
      allow write: if request.auth != null;
    }
  }
}
```

Slika 9 Firestore Database sigurnosna pravila

[8] JavaScript Object Notation je jednostavan format za razmjenu podataka.

2.2.7 Crashlytics

Firebase Crashlytics predstavlja rješenje za tvrtke koje žele učinkovito pratiti i rješavati probleme s rušenjem njihovih aplikacija. Ovaj alat pruža detaljna izvješća o padu aplikacije, olakšavajući brzu analizu uzroka i promptno rješavanje problema (TechTarget, 2024.).

2.2.8 Performance Monitoring

Firebase Performance Monitoring predstavlja alat za organizacije koje žele dobivati precizne informacije o performansama svojih aplikacija. Alat omogućuje praćenje u stvarnom vremenu ključnih pokazatelja performansi, uključujući CPU, memoriju i mrežni promet, pružajući vrijedne uvide o tome koliko dobro aplikacija funkcionira (TechTarget, 2024.).

2.2.9 Test Lab

Firebase Test Lab, temeljen na oblaku, predstavlja alat za programere koji žele temeljito testirati svoje aplikacije na raznovrsnim uređajima i postavkama. Alat pruža programerima mogućnost testiranja njihovih aplikacija na nizu različitih uređaja i različitim postavkama, pomažući im osigurati ispravno funkcioniranje programa na različitim platformama i u različitim mrežnim scenarijima (TechTarget, 2024.).

2.3 Sigurnost

Platforma Firebase ozbiljno pristupa sigurnosti podataka, a to je evidentirano certifikatima ISO 27001 i SOC 2 tipa 2 za svaki od svojih podatkovnih centara. Kako bi osigurao potpunu zaštitu podataka, Firebase primjenjuje niz sigurnosnih postupaka, uključujući sljedeće ključne aspekte:

- **Šifriranje podataka**

Svaki bit podataka na Firebase-u prolazi kroz postupak šifriranja kako bi se osiguralo da su informacije sigurne tijekom prijenosa između korisnika i aplikacije, ali i dok miruju unutar sustava.

- **Kontrola pristupa na temelju uloga (eng. Role-based access control (RBAC))**

Kontrola pristupa temeljena na ulogama (RBAC) predstavlja značajku Firebase-a koja pruža preciznu kontrolu nad time tko ima pristup podacima aplikacije. Funkcionalnost omogućuje organizacijama postavljanje jasnih granica pristupa, čime se smanjuje rizik od neovlaštenog pristupa informacijama.

- **Zapisnici za revizije (eng. Audit logging)**

Firebase omogućuje tvrtkama praćenje svakog pristupa podacima aplikacije putem detaljnih zapisnika za reviziju. Zapisnici bilježe tko je pristupio podacima, kada je to učinjeno te pružaju transparentnost u korištenju podataka (TechTarget, 2024.).

3. React

React, poznat i kao React.js ili ReactJS, predstavlja moćnu besplatnu front-end9 JavaScript biblioteku otvorenog koda namijenjenu izgradnji korisničkih sučelja temeljenih na komponentama (React, 2024.). Ova biblioteka, održavana od strane Meta (ranije Facebooka) i široke zajednice pojedinačnih programera i tvrtki, pruža iznimnu fleksibilnost i brzinu u razvoju web i mobilnih aplikacija. Temeljen na konceptu komponenata, React omogućuje razdvajanje složenog koda na pojedinačne dijelove, olakšavajući organizaciju koda i poboljšavajući održivost projekta. React se ističe po sposobnosti rukovanja korisničkim sučeljem i renderiranja komponenata u DOM-u10. Omogućuje razvoj jednostranih, mobilnih ili poslužiteljskih aplikacija, a često se koristi u kombinaciji s okvirima poput Next.js radi optimalnog iskustva

[9] Dio aplikacije koji korisnici vide i koji služi za interakciju između korisnika i aplikacije.
[10] Document Object Model predstavlja strukturu objekata koja predstavlja dokument i omogućuje programima, posebno web preglednicima, da dinamički mijenjaju strukturu, stilove i sadržaj web stranica.

razvoja. Važno je naglasiti da se React aplikacije često oslanjaju na dodatne biblioteke za usmjeravanje i druge funkcionalnosti na strani klijenta (Wikipedia, 2024.).

3.1 Povijest

React je stvoren od strane Jordan Walke-a, softverskog inženjera kompanije Meta. Inicijalno, Walke je razvio prototip pod nazivom "F-Bolt" prije nego što mu je dao novo ime "FaxJS". Značajna prekretnica u povijesti Reacta dogodila se u svibnju 2013. godine tijekom JavaScript konferencije u Americi, kada je projekt službeno otvoren. Ovaj trenutak označio je važan korak u širenju Reacta, što ga je učinilo dostupnim i pristupačnim širokoj zajednici programera. Otvaranje projekta potvrdilo je njegov rastući značaj i prihvaćenost u svijetu razvoja web aplikacija (Wikipedia, 2024.).

3.2 React značajke

3.2.1 Deklarativna paradigma

React dosljedno slijedi paradigmu deklarativnog programiranja. U ovom pristupu, programeri oblikuju prikaze za svako stanje aplikacije, a React preuzima ulogu ažuriranja i renderiranja komponenata kad se podaci mijenjaju. Ova metodologija stoji u suprotnosti s imperativnim programiranjem, gdje programeri detaljno opisuju kako postići određeni rezultat (Wikipedia, 2024.).

3.2.2 Komponente

Središnji koncept u Reactu su komponente, entiteti koji čine srž React koda. Ove komponente su modularne i dizajnirane da budu ponovno iskoristive. Tipična React aplikacija sastoji se od mnogo slojeva komponenata, što omogućuje programerima jasnu organizaciju koda i održavanje skalabilnosti. React DOM biblioteka koristi se za prikazivanje komponenata u korijenskom elementu DOM-a. Na slici 10 prikazana je komponenta „NoMatch“.

```
const NoMatch = () => {
  return (
    <div style={{ padding: 20 }}>
      <h2>404: Page Not Found</h2>
    </div>
  );
};

export default NoMatch;
```

Slika 10 Komponenta NoMatch

Prilikom renderiranja komponenti, vrijednosti se prenose između komponenata pomoću objekta koji se naziva „props“, skraćenica od „svojstva“ (eng. properties) (prikazano na slici 11). Ove vrijednosti unutar komponenti nazivaju se njezinim stanjem. Ovaj pristup olakšava upravljanje podacima i omogućuje dinamičko ponašanje aplikacije (Wikipedia, 2024.).

```
const Navigation = ({user, logOut}) => {
  const [isOpen, setIsOpen] = useState(false);

  const toggle = () => {
    setIsOpen(!isOpen);
  };
  return (
```

Slika 11 Props "user" i "logOut" prosljeđuju se komponenti Navigation

3.2.3 Funkcijske komponente

Funkcijske komponente u Reactu deklariraju se pomoću funkcije, koristeći JavaScript sintaksu funkcije ili izraz funkcije strelice (eng. arrow function). Ove komponente prihvaćaju jedan argument po imenu "props" i vraćaju JSX[11], pružajući jasno strukturirane i jednostavne

[11] JavaScript proširenje koje omogućuje pisanje koda sličnog XML-u unutar JavaScript koda

načine definiranja dijelova korisničkog sučelja. Važno je napomenuti da od verzije Reacta 16.8 nadalje, funkcionalne komponente dobivaju dodatnu snagu s uvođenjem "useState" „hook-a“. Ovaj „hook“ omogućuje funkcionalnim komponentama da koriste stanje, čime se proširuje njihova sposobnost upravljanja dinamičkim promjenama unutar aplikacije

```
const Sidebar = ({ isOpen, toggle, user, logOut }) => {
  return (
    <SidebarContainer isOpen={isOpen} onClick={toggle}>
```

Slika 12 Funkcijska komponenta Sidebar

Funkcijska komponenta „Sidebar“ (prikazano na slici 12) prima četiri svojstva putem destrukturiranja „props“ objekta i zatim vraća JSX element „<SidebarContainer>“ (Wikipedia, 2024.).

3.2.4 React Hooks

React je svojom verzijom 16.8 lansirao revolucionarnu značajku „React Hooks“ koja predstavlja funkcije koje programerima omogućuju "povezivanje" stanja i životnog ciklusa Reacta iz funkcionalnih komponenti. Važno je napomenuti da se „hook-ovi“ koriste isključivo unutar funkcionalnih komponenti, nudeći programerima alat za upravljanje stanjem i ponašanjem komponenata bez potrebe za klasama. React nudi nekoliko ugrađenih „hook-a“ koje su postale ključni dio modernog razvoja React aplikacija. Među njima su „useState“, „useContext“, „useReducer“, „useMemo“ i „useEffect“. Dok su „useState“ i "useEffect“ među najčešće korištenima, svaki od ovih „hook-ova“ pruža specifične funkcionalnosti za kontrolu stanja, upravljanje kontekstom, upravljanje smanjenjem stanja, optimizaciju performansi te upravljanje nuspojavama. Primjerice, "useState" se često koristi za kontrolu stanja komponente, dok se "useEffect" koristi za upravljanje nuspojavama ili efektima koji se javljaju nakon svakog renderiranja komponente (prikazano na slici 13). Svi ostali „hook-ovi“ dokumentirani su u „Hooks API Reference“, pružajući programerima obilje mogućnosti za prilagodbu i optimizaciju njihovog React koda (Wikipedia, 2024.).

```
const Events = () => {
  const [events, setEvents] = useState([]);
  const [searchTerm, setSearchTerm] = useState("");
  const [selectedLocation, setSelectedLocation] = useState("");
  const [selectedGenre, setSelectedGenre] = useState("");
  const [sortOrder, setSortOrder] = useState("asc");
  const [startDate, setStartDate] = useState(null);
  const [endDate, setEndDate] = useState(null);
  const [allLocations, setAllLocations] = useState([]);
  const [allGenres, setAllGenres] = useState([]);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const eventsRef = collection(db, "events");
        const snapshot = await getDocs(eventsRef);
        const eventsData = snapshot.docs.map((doc) => ({
          id: doc.id,
          ...doc.data(),
        }));
        setEvents(eventsData);

        const locations = [
          ...new Set(eventsData.map((event) => event.location)),
        ];
        setAllLocations(locations);

        const genres = [...new Set(eventsData.map((event) => event.genre))];
        setAllGenres(genres);
      } catch (error) {
        console.error("Error fetching events: ", error);
      }
    };

    fetchData();
  }, []);
```

Slika 13 Korištenje hook-a "useState" i "useEffect"

3.2.5. Usmjeravanje (eng. Routing)

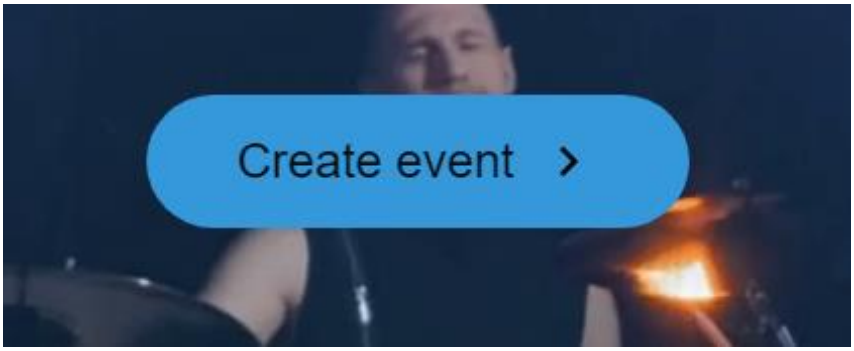
React ne uključuje ugrađenu podršku za usmjeravanje, stoga je razvojni proces za rukovanje rutama u React aplikacijama olakšan uz pomoć popularnih biblioteka trećih strana. Jedna od najčešće korištenih biblioteka za upravljanje usmjeravanjem je „react-router“. React-router pruža sveobuhvatno rješenje za definiranje ruta, navigaciju i upravljanje promjenama URL-a na „React-friendly“ način. Glavna funkcionalnost „react-router-dom-a“ leži u implementaciji dinamičkog usmjeravanja u web aplikacijama. Biblioteka pruža rješenje za upravljanje rutama koje se temelje na komponentama, što je idealno za usmjeravanje u React aplikacijama koje se izvode u pregledniku. Dinamičko usmjeravanje znači da se navigacija kroz aplikaciju prilagođava dinamički, ovisno o trenutačnom stanju aplikacije, korisničkim interakcijama ili drugim uvjetima. „React-router-dom“ omogućava definiranje ruta koje su povezane s određenim komponentama, što olakšava kontrolu nad time koje se komponente renderiraju ovisno o trenutnom URL-u ili korisničkoj interakciji (prikazano na slici 14). Glavni paket za implementaciju usmjeravanja u React aplikacijama je react-router, koji uključuje uobičajene komponente i značajke usmjeravanja u React aplikacijama. React-router-dom je specijalizirani paket koji je specificiran za izradu web aplikacija. Koristi ovisnosti „react-router-a“ sa dodatnim DOM (document object model) komponentama kao što su „<BrowserRouter>“ i “<Link>“ (Syncfusion, 2024.).

```
<BrowserRouter>
  {shouldRenderNavigation && <Navigation user={user} logOut={logOut} />}
  <Routes>
    <Route path="/" element={<Hero />} />
    <Route path="/events" element={<Events />} />
    <Route path="/login" element={<Login onLogin={setUser} />} />
    <Route path="/register" element={<Register />} />
    <Route path="/createevent/*" element={<CreateEvent user={user} />} />
    <Route path="*" element={<NoMatch />} />
  </Routes>
</BrowserRouter>
```

Slika 14 Usmjeravanje korištenjem objekata iz „react-router-dom“ biblioteke

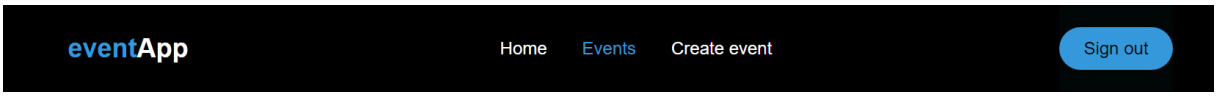
4. Dizajn

Dizajn aplikacije ima zadaću korisnicima povećati privlačnost i olakšati bolju snalažljivost kroz navigaciju aplikacije. Za postizanje oku privlačnog i intuitivnog dizajna korištena je određena paleta boja. Za navigaciju (prikazano na slici 17) kroz aplikaciju korištena je crna boja (#010606) koja je zaslužna za moderan stil današnjih aplikacija kojima prevladava crna boja. Bijela boja (#fff) koristi se za kontrast, odnosno za lakšu čitljivost kod forme unosa i uređivanja podataka. Plava boja (#3498db) koja se koristi dodaje osvježavajuću boju za razbijanje monotonosti. Boja je korištena za gumbove i za označavanje trenutnog stanja kroz navigaciju kako bi korisnik mogao bez naprezanja oka odrediti svoju trenutnu poziciju u navigaciji kroz samu aplikaciju. Implementacija samog stila u programski kod odrađena je pomoću stiliziranih komponenti (eng. style-components) i kroz „module.css“. Korištene su ikone iz FontAwesom biblioteke koja omogućuje implementaciju svih njihovih ikona. Ikone su prepoznatljive, poboljšavaju estetski dojam i korisničko iskustvo. Dodana je animacija na gumbove koja mijenja boju i ikonu prilikom prelaska pokazivača da bi se naglasila dinamičnost aplikacije i interaktivnost između korisnika i aplikacije (prikazano na slici 15 i 16).



Slika 15 Animacija gumba prije prelaska pokazivača

Slika 16 Animacija gumba nakon prelaska pokazivača



Slika 17 Navigacijska traka

4.1 Styled-components

React style-components omogućuje pisanje CSS-a[12] unutar samog JavaScript koda na modularan način koji je spreman za ponovno korištenje u Reactu. Styled-components modul u React-u omogućuje korištenje samih React komponenata kao osnova za definiranje stilova na niskom nivou. Mapiranje između komponenata i njihovih stilova u potpunosti se eliminira, čime se pojednostavljuje proces stiliziranja i održavanja koda. Na slici 18 prikazano je stiliziranje komponente „HeroButton“ korištenjem styled-components modula (styled components, 2024.).

[12] Jezik za opis stila, oblikovanje izgleda i određivanja pravila izgleda elemenata.

```
const HeroButton = styled(Link)`
  border-radius: 50px;
  background: ${({ primary }) => (primary ? "#3498db" : "#010606")};
  white-space: nowrap;
  padding: ${({ big }) => (big ? "14px 48px" : "12px 30px")};
  color: ${({ dark }) => (dark ? "#010606" : "#fff")};
  font-size: ${({ fontBig }) => (fontBig ? "20px" : "16px")};
  outline: none;
  border: none;
  cursor: pointer;
  display: flex;
  justify-conten: center;
  align-items: center;
  transiton: all 0.2s ease-in-out;
  text-decoration: none;

  &:hover {
    transiton: all 0.2s ease-in-out;
    background: ${({ primary }) => (primary ? "#fff" : "#3498db")};
  }
`;
```

Slika 18 Stilizirana komponenta "HeroButton"

4.2. Module.css

CSS Module je CSS datoteka sa svim nazivima klasa i animacijskim nazivima prema zadanim postavkama lokalno (prikazano na slici 19). Također je komponentno ciljan (eng. component-scoped) CSS koji je portabilan, tradicionalan i omogućuje minimalne nuspojave oko međusobnih sudaranja naziva ili utjecaja stila na druge komponente. Pravila stila su primijenjena na određenu komponentu ili modul. CSS Moduli pridonose boljoj organizaciji i čitljivosti koda (Gatsby, 2024.).

```css
.hero-content {
  z-index: 3;
  max-width: 1200px;
  position: absolute;
  padding: 8px 24px;
  display: flex;
  flex-direction: column;
  align-items: center;
}

.hero-h1 {
  color: #fff;
  font-size: 48px;
  text-align: center;

  @media screen and (max-width: 768px) {
    font-size: 40px;
  }

  @media screen and (max-width: 480x) {
    font-size: 32px;
  }
}
```

Slika 19 Module.css komponente "Hero"

4.3 FontAwesome

Fontawesome nudi službenu komponentu koja je jednostavna za implementaciju u React aplikacijama. Omogućuje jednostavno korištenje fontawesome ikona na individualan ili globalan način.

5. Analiza programskog koda

5.1 Povezivanje Firebase-a s Aplikacijom

Datoteka "firebase.js" (prikazano na slici 20) služi za konfiguraciju i inicijalizaciju Firebase SDK-ova za usluge Autentifikacije, Firestore-a i pohrane podataka u web aplikaciji. Uvoze se potrebne funkcije iz Firebase biblioteke. „InitializeApp“ se koristi za inicijalizaciju Firebase

aplikacije, dok se "getAuth“, „getFirestore“ i „getStorage“ koriste za autentifikaciju, bazu podataka i spremište podataka. Definiraju se potrebni ključevi koje dobivamo putem Firebase konzole, a zatim se inicijaliziraju Firebase Authentication, Firestore i Storage kako bismo dobili njihove instance.

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getAuth } from "firebase/auth";
import { getFirestore } from "firebase/firestore";
import { getStorage } from "firebase/storage";

// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "[redacted]",
  authDomain: "[redacted]",
  projectId: "[redacted]",
  storageBucket: "[redacted]",
  messagingSenderId: "[redacted]",
  appId: "[redacted]",
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);

// Initialize Firebase Authentication and get a reference to the service
export const auth = getAuth(app);
export const db = getFirestore(app);
export const storage = getStorage(app);
```

Slika 20 firebase.js datoteka

5.2 Autentifikacija

5.2.1 Registracija

Funkcija „signUp“ koristi se za obradu registracije korisnika (prikazano na slici 21). Funkcija se poziva kada korisnik pošalje obrazac za registraciju. Koristi se „(e.preventDefault())“ funkcija koja sprječava automatsko ponovno učitavanje stranice koje se

obično dešava kada se obrazac pošalje. Zatim se koristi funkcija iz Firebase auth servisa „createUserWithEmailAndPassword“ kako bi se stvorio korisnički račun korisnika sa unesenim e-mailom i lozinkom. Novostvorene informacije o korisnikovom računu, čija je registracija uspješna, spremaju se u „userCredential“ nakon čega se korisnik preusmjerava na stranicu "login“. Kod neuspjele registracije obavještava se korisnik da je došlo do pogreške i traži se od njega da se ponovo registrira.

```
const signUp = (e) => {
  e.preventDefault();
  createUserWithEmailAndPassword(auth, email, password)
    .then((userCredential) => {
      console.log(userCredential);
      alert("Register Success!");
      navigate("/login");
    })
    .catch((error) => {
      console.log(error);
      alert("Error register! Handle properly!");
    });
};
```

Slika 21 Funkcija "signUp"

5.2.2. Prijava

Funkcija „handleLogin“ poziva funkciju „signInWithEmailAndPassword“ iz Firebase Authentication servisa koja provjerava unesene vjerodajnice i prijavljuje korisnika u aplikaciju ukoliko su ispravne (prikazano na slici 22). Nakon uspješne prijave sprema se „userCredential“ koji sadržava informacije o korisničkom računu koji je uspješno prijavljen. Korisnički podaci spremaju se u lokalno spremište kako bi se održala sesija korisnika između ponovnih posjeta stranici. Korisnik se preusmjeruje na stranicu koja služi za kreiranje vlastitih događaja i koja omogućava kontrolu nad vlastitim događajima, pristup stranici je moguć samo ako je korisnik prijavljen u aplikaciju. Ukoliko prijava u aplikaciju nije uspješna ispisuje se poruka o grešci prilikom prijave i traži se od korisnika ponovna prijava u aplikaciju.

```
const handleLogin = () => {
  signInWithEmailAndPassword(auth, creds.email, creds.password)
    .then((userCredential) => {
      const user = userCredential.user;
      onLogin && onLogin({ email: creds.email });
      localStorage.setItem("user", JSON.stringify(user));
      navigate("/createevent");
      console.log("Login Success!");
    })
    .catch((error) => {
      // const errorCode = error.code;
      // const errorMessage = error.message;
      console.log("Error login! Handle properly!");
      alert("Wrong email or password, try again!");
    });
};
```

Slika 22 Funkcija "handleLogin"

5.2.3 Odjava

Funckija „logOut“ poziva funckiju „signOut“ iz Firebase Authentication servisa kako bi se izvršila odjava trenutnog korisnika koji se dobiva pomoću prethodno incijalizirane instance „getAuth()“ (prikazano na slici 23). Ukoliko je odjava uspješna brišu se podaci o trenutnom korisniku iz lokalnog spremišta kako bi se završila sesija. Ispisuje se poruka u uspješnoj odjavi korisnika, vraća se stanje varijable „setUser“ na vrijednost „null“ i korisnik se vraća na početnu stranicu. Dođe li do greške prilikom odjave ispisuje se poruka o pogrešci.

```
function logOut() {
  signOut(getAuth())
    .then(() => {
      localStorage.removeItem("user");
      console.log("Logout Success");
      setUser(null);
      navigate("/");
    })
    .catch((error) => {
      console.log("Logout Error");
    });
}
```

Slika 23 Funkcija "logOut"

5.3 Dohvaćanje podataka iz baze podataka

Koristi se React Hook „useEffect“ kako bi se izvršio određeni kod nakon što se komponenta inicijalno prikaže na ekranu. Poziva se funkcija „fetchData“ koja se koriste za dohvaćanje podataka iz Firestore baze podataka. Dohvaća se referenca na Firestore kolekciju „events“ pomoću „collection“ funkcije. Pomoću funkcije „getDocs“ izvršava se upit za dobivanje svih dokumenata iz kolekcije. Podatke o događajima postavljamo kao stanje pomoću „setEvents“. Izdvajamo jedinstvene lokacije i žanrove iz podataka o događajima i postavljamo ih u stanje komponenti koje čuvaju sve lokacije i svi žanrovi. Slika 24 prikazuje dohvaćanje podataka iz baze podataka.

```
useEffect(() => {
  const fetchData = async () => {
    try {
      const eventsRef = collection(db, "events");
      const snapshot = await getDocs(eventsRef);
      const eventsData = snapshot.docs.map((doc) => ({
        id: doc.id,
        ...doc.data(),
      }));
      setEvents(eventsData);

      const locations = [
        ...new Set(eventsData.map((event) => event.location)),
      ];
      setAllLocations(locations);

      const genres = [...new Set(eventsData.map((event) => event.genre))];
      setAllGenres(genres);
    } catch (error) {
      console.error("Error fetching events: ", error);
    }
  };

  fetchData();
}, []);
```

Slika 24 Funkcija "fetchData"

5.4 Kreiranje novog objekta

Prvo se provjerava jesu li ispunjena sva obavezna polja, zatim se koristi „collection“ kako bi se dohvatila referenca na kolekciju „events“. Inicijalizira se novi objekt s podacima u novom događaju. Objekt se dodaje u Firestore kolekciju „events“ pomoću funckije „addDoc“. Dohvaća se ID dodanog događaja koji se kasnije koristi za stvaranje reference u Storage-u. Ukoliko je dodana slika („eventImage“) na događaj, slika se prenosi na odabrano mjesto u Storage-u, URL slike se dodaje u novokreirani objekt. Ažurira se dokument s novim podaci u Firestore-u i korisnik dobiva poruka da je novi događaj uspješno kreiran. Na kraju se resetiraju polja za unos podataka. Na slici 25 prikazano je kreiranje novog objekta.

```
const handleCreateEvent = async () => {
  try {
    // Provjeri jesu li sva polja ispunjena
    if (
      !eventName ||
      !eventLocation ||
      !eventPrice ||
      !eventTimestamp ||
      !eventGenre
    ) {
      window.alert("Please fill all fields before creating an event.");
      return;
    }

    const eventsRef = collection(db, "events");

    // Deklariraj addedEventId prije korištenja
    let addedEventId;

    const newEvent = {
      name: eventName,
      location: eventLocation,
      price: eventPrice,
      timestamp: eventTimestamp ? new Date(eventTimestamp) : null,
      genre: eventGenre,
      createdBy: user.email,
    };

    const addedEventRef = await addDoc(eventsRef, newEvent);

    // Dodijeli vrijednost addedEventId
    addedEventId = addedEventRef.id;

    const storageRef = ref(storage, `event_images/${addedEventId}`);
    if (eventImage) {
      const imageSnapshot = await uploadBytes(storageRef, eventImage);
      const imageUrl = await getDownloadURL(imageSnapshot.ref);
      newEvent.image = imageUrl;
    }

    const updatedEventRef = doc(db, "events", addedEventId);
    await updateDoc(updatedEventRef, newEvent);

    window.alert("Event created successfully!");

    // Ažuriraj lokalno stanje dodavanjem novog događaja
    setUserEvents((prevUserEvents) => [
      ...prevUserEvents,
      { id: addedEventId, ...newEvent },
    ]);

    // Resetiraj polja za unos
    setEventName("");
    setEventLocation("");
    setEventPrice("");
    setEventTimestamp("");
    setEventGenre("");
    setEventImage(null);
    setIsCreatingEvent(false);
  } catch (error) {
    console.error("Error creating event: ", error);
  }
};
```

Slika 25 Funkcija "handleCreateEvent"

5.5 Ažuriranje objekta

Dobiva se referenca na odabrani događaj u Firestore-u pomoću funkcije „doc“. Ažurirani podaci spremaju se u „updateEvent“. Provjerava se postoji li nova slika, ako postoji ažurira se slika u Firestore Storage-u. Funkcija „updateDoc“ ažurira podatke o događaju u bazi. Korisnik dobiva poruku o uspješnom ažuriranju događaja i nakon toga se resetiraju polja za unos podataka. Na slici 26 prikazana je funkcija za ažuriranje objekta iz baze podataka.

```
const handleEditEvent = async () => {
  try {
    const eventRef = doc(db, "events", editingEvent.id);
    const updatedEvent = {
      name: eventName,
      location: eventLocation,
      price: eventPrice,
      ...(eventTimestamp ? { timestamp: new Date(eventTimestamp) } : {}),
      genre: eventGenre,
    };

    // Ako postoji odabrana nova slika, ažuriraj je u Firebase Storage
    if (eventImage) {
      const storageRef = ref(storage, `event_images/${editingEvent.id}`);
      await uploadBytes(storageRef, eventImage);
      const imageUrl = await getDownloadURL(storageRef);
      updatedEvent.image = imageUrl;
    }

    console.log("Updated Event Data:", updatedEvent);

    await updateDoc(eventRef, updatedEvent);

    // Ažuriraj lokalno stanje kako biste odmah vidjeli promjene
    setUserEvents((prevUserEvents) =>
      prevUserEvents.map((e) =>
        e.id === editingEvent.id ? { ...e, ...updatedEvent } : e
      )
    );

    window.alert("Event updated successfully!");

    setEventName("");
    setEventLocation("");
    setEventPrice("");
    setEventTimestamp("");
    setEventGenre("");
    setEditingEvent(null);
    setIsCreatingEvent(false);
  } catch (error) {
    console.error("Pogreška prilikom ažuriranja događaja: ", error);
  }
};
```

Slika 26 Funkcija "handleEditEvent"

5.6 Brisanje objekta

Pomoću funkcije „eventRef“ dobiva se referenca na događaj, odnosno objekt koji se želi obrisati iz baze. Funkcija „deleteDoc“ briše odabrani objekt iz baze podataka. Zatim se ažurira lokalno stanje korisnikovih događaja u bazi kako bi se događaj odmah uklonio sa liste i dobiva se poruka u uspješnom brisanju događaja iz baze podataka. Na slici 27 prikazano je brisanje objekta iz baze podataka.

```
const handleDeleteEvent = async (event) => {
  try {
    const eventRef = doc(db, "events", event.id);
    await deleteDoc(eventRef);

    // Ažuriraj lokalno stanje kako bi se događaj odmah uklonio
    setUserEvents((prevUserEvents) =>
      prevUserEvents.filter((e) => e.id !== event.id)
    );

    window.alert("Event deleted successfully!");

    setEditingEvent(null);
  } catch (error) {
    console.error("Error deleting event: ", error);
  }
};
```

Slika 27 Funkcija "handleDeleteEvent"

5.7 Filtriranje

Filtriranje se vrši po nazivu događaja, lokaciji, žanru i datumu. Varijabla „filteredEvents“ je rezultat filtriranih događajima po atributima. Filtriranje se primjenjuje na prethodni rezultat pomoću funkcije „filter()“ kako bi se moglo primijeniti više filtera odjednom. Uspoređuje se da

li odabrani naziv, lokacija, žanr ili datum u rasponu odgovaraju događajima iz baze podataka. Na slici 28 prikazano je filtriranje događaja.

```
const filteredEvents = events
  .filter((event) =>
    event.name.toLowerCase().includes(searchTerm.toLowerCase())
  )
  .filter((event) =>
    selectedLocation
      ? event.location.toLowerCase() === selectedLocation.toLowerCase()
      : true
  )
  .filter((event) =>
    selectedGenre
      ? event.genre.toLowerCase() === selectedGenre.toLowerCase()
      : true
  )
  .filter((event) => {
    if (startDate && endDate) {
      return (
        event.timestamp?.toDate() >= startDate &&
        event.timestamp?.toDate() <= endDate
      );
    }
    return true;
  });
```

Slika 28 Filtriranje događaja

6. Zaključak

Završni rad predstavlja inovativnu aplikaciju za organizaciju događaja koja se ističe suvremenim pristupom i korištenjem naprednih tehnologija. Front-end dijelom aplikacije upravlja React, dok je back-end implementiran putem Firebase platforme. Ova kombinacija pruža visoku razinu performansi, prilagodljivost i jednostavnost u korištenju. Jedan od ključnih aspekata aplikacije je njezin visoki potencijal koji je poduprt dobro osmišljenim korisničkim sučeljem. Dizajn je jednostavan i intuitivan, čime se korisnicima omogućuje lako snalaženje i interakcija s aplikacijom. Firebase Authentication osigurava siguran pristup aplikaciji, čime se štite korisnički podaci. Aplikacija ima pozitivan utjecaj na korisnike pružajući im mogućnost jednostavnog pronalaženja i kreiranja događaja, odnosno upravljanje događajima. Praktična funkcionalnost čini je korisnom za širok spektar korisnika, bez obzira na uzrast. Otvorenost prema različitim interesima i mogućnost pronalaska zanimljivosti u blizini čine je odličnim alatom za organizaciju događaja i zajedničkih aktivnosti. Sveukupno, aplikacija predstavlja korak naprijed u optimizaciji organizacije događaja pomoću naprednih tehnologija.

Reference

Firebase. (14. Veljača 2024.). Dohvaćeno iz Documentation: https://firebase.google.com/docs

Gatsby. (14. Veljača 2024.). Dohvaćeno iz Documentation: https://www.gatsbyjs.com/docs/how-to/styling/css-modules/

geeksforgeeks. (14. Veljača 2024.). Dohvaćeno iz Firebase – Introduction: https://www.geeksforgeeks.org/firebase-introduction/

React. (14. Veljača 2024.). Dohvaćeno iz Documentation: https://legacy.reactjs.org/docs/getting-started.html

styled components. (14. Veljača 2024.). Dohvaćeno iz Documentation: https://styled-components.com/docs

Syncfusion. (14. Veljača 2024.). Dohvaćeno iz React Router vs. React Router DOM: https://www.syncfusion.com/blogs/post/react-router-vs-react-router-dom.aspx

TechTarget. (14. Veljača 2024.). Dohvaćeno iz Google Firebase: https://www.techtarget.com/searchmobilecomputing/definition/Google-Firebase

Wikipedia. (14. Veljača 2024.). Dohvaćeno iz Firebase: https://en.wikipedia.org/wiki/Firebase

Wikipedia. (14. Veljača 2024.). Dohvaćeno iz React (software): https://en.wikipedia.org/wiki/React_(software)

Popis slika

Popis priloga

Repozitorij aplikacije „eventapp“:

https://github.com/mbasek/eventapp