Izrada web aplikacije za internu komunikaciju i oglašavanje unutar poduzeća

Pavlaković, Mihael

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka / Sveučilište u Rijeci**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:495814

Sveučilište u Rijeci – Fakultet informatike i digitalnih tehnologija

Informacijski i komunikacijski sustavi

Mihael Pavlaković

Izrada web aplikacije za internu komunikaciju i oglašavanje unutar poduzeća

Diplomski rad

Mentor: doc. dr. sc. Lucia Načinović Prskalo

Rijeka, 11.06.2023

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

Rijeka, 7.5.2023.

Zadatak za diplomski rad

Pristupnik: Mihael Pavlaković

Naziv diplomskog rada: Izrada web aplikacije za internu komunikaciju i oglašavanje unutar poduzeća

Naziv diplomskog rada na eng. jeziku: Developing web application for internal communication and advertising within the company

Sadržaj zadatka: Zadatak diplomskog rada je izraditi aplikaciju za komunikaciju djelatnika i oglašavanje unutar poduzeća. Istražit će se i odabrati najprikladniji alati za razvoj web aplikacija. Opisat će se odabrani alati te sve funkcionalnosti i mogućnosti koje aplikacija nudi. Također će se pojasniti kôd kojim su implementirani elementi aplikacije.

Mentor:

Doc. dr. sc. Lucia Načinović Prskalo

Načinović Prskalo

Voditeljica za diplomske radove:

Prof. dr. sc. Ana Meštrović

Zadatak preuzet: 10.5.2023.

(potpis pristupnika) Pavlaković

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

Tablica sadržaja

1. Sažetak

Unutar diplomskog rada opisan je postupak i koncept izrade web aplikacije za internu komunikaciju i oglašavanje unutar poduzeća pod nazivom *Notice Board*. Cilj ove aplikacije je olakšati komunikaciju i oglašavanje unutar određenih poslovnih poduzeća koristeći potrebne funkcionalnosti aplikacije. U uvodnom dijelu objašnjeno je zašto su ovakve aplikacije potrebne te koje su njihove uobičajene funkcionalnosti. Potom su u poglavlju „Web tehnologije i alati“ opisane neke od postojećih tehnologija koje se koriste za izradu web aplikacija te one koje su izdvojene kao najbolje za izradu aplikacije *Notice Board*. Poglavlje „Funkcionalnost web aplikacije“ opisuje funkcionalnosti izrađene aplikacije i njene značajke u pozadini. Za kraj, u poglavlju „Budućnost web aplikacije“ opisane su mogućnosti dorade i poboljšanja određenih dijelova aplikacije u budućnosti, te su u poglavlju „Zaključak“ dane zaključne misli i mišljenja.

Ključne riječi: web aplikacija, web development, frontend, react.js, redux, state, funkcija, akcija, baza podataka, aplikacija za internu komunikaciju

2. Uvod

U današnjem digitalnom dobu, web tehnologije su postale neizostavan alat za komunikaciju, razmjenu informacija i poslovanje u svim sferama društva. Web aplikacije predstavljaju moćan način da se pristupi i koristi tehnologija putem internetskog preglednika, omogućujući korisnicima da interaktivno i učinkovito sudjeluju u različitim aspektima svakodnevnog života. Jedno od područja koje je doživjelo značajan napredak zahvaljujući web tehnologijama je interna komunikacija i oglašavanje unutar poduzeća.

Interni komunikacijski procesi igraju ključnu ulogu u funkcioniranju modernih organizacija. Učinkovita i pravovremena razmjena informacija među zaposlenicima potiče suradnju, timski rad i povećava produktivnost. Tradicionalni načini komunikacije, poput e-pošte, često nije dovoljno brz ili interaktivni način da zadovolji potrebe suvremenih organizacija koje se susreću s brzim promjenama i zahtjevima tržišta.

U tom kontekstu, izrada web aplikacija za internu komunikaciju i oglašavanje unutar poduzeća pruža inovativno rješenje za olakšavanje i poboljšanje komunikacije među zaposlenicima. Ove aplikacije omogućuju stvaranje centraliziranog digitalnog prostora gdje se mogu razmjenjivati poruke, obavijesti, dokumenti i drugi resursi. Uz to, web aplikacije za internu komunikaciju pružaju mogućnost interakcije putem različitih oblika, kao što su forumi, chatovi, komentari i ankete, potičući aktivno sudjelovanje zaposlenika.

Nadalje, web aplikacije za oglašavanje unutar poduzeća omogućuju jednostavno i ciljano dijeljenje informacija o internim događanjima, obavijestima o promjenama politika ili procedura te drugim relevantnim informacijama. Ove aplikacije pružaju mogućnost personalizacije sadržaja, prilagođavajući informacije specifičnim timovima ili pojedincima, čime se povećava njihova relevantnost i korisnost.

Neke od najboljih web aplikacija za internu komunikaciju i oglašavanje su: „Slack, Trello, Asana, Evernote, Basecamp, Zoom, Webex“ (Lider.hr, 2019) i slično.

U svjetlu navedenih činjenica, ovaj diplomski rad fokusira se na pregled izrađene web aplikacije za internu komunikaciju i oglašavanje unutar poduzeća. U cilju zadovoljavanja specifičnih potreba, ova web aplikacija pruža različite opcije za kreiranje i uređivanje objava kako bi omogućili interakciju i angažman korisnika odnosno čitatelja objave.

Kroz aplikaciju, korisnici imaju mogućnost kreiranja tekstualnih objava koje sadrže važne informacije, obavijesti ili rasprave vezano za internu komunikaciju. Osim teksta, korisnici također mogu priložiti dokumente, poput prezentacija, slika, dokumenata s informacijama ili drugih relevantnih materijala koji su dostupni za pregled i preuzimanje od strane ostalih korisnika.

Jedna od važnih funkcionalnosti aplikacije je mogućnost kreiranja glasanja ili anketa (eng. polls). Ona korisnicima omogućava postavljanje pitanja, ponuđenih opcija za odabir te prikupljanje povratnih informacija i mišljenja kolega. Ova funkcionalnost omogućava aktivno sudjelovanje zaposlenika u procesima donošenja odluka, istraživanju mišljenja ili dobivanju povratnih informacija o ključnim pitanjima unutar poduzeća.

Nadalje, aplikacija podržava interakciju korisnika s objavama putem komentara. Korisnici mogu ostavljati komentare na objave drugih korisnika kako bi izrazili svoje mišljenje, postavili pitanja ili pokrenuli diskusiju. Ova funkcionalnost potiče otvorenu komunikaciju i suradnju među zaposlenicima, olakšavajući razmjenu ideja i informacija.

Za postizanje svih navedenih funkcionalnosti, planiramo koristiti suvremene web tehnologije koje će osigurati efikasan i intuitivan rad aplikacije. Detaljnije o tome bit će rečeno u sljedećem poglavlju pod nazivom „Web tehnologije i alati“. Potom dolazimo do poglavlja „Funkcionalnost web aplikacije“ unutar kojeg su prikazane i opisane funkcionalnosti te sam izgled aplikacije i programskog koda koji je zaslužan za to. Za sam kraj ostavljamo poglavlja „Budućnost web aplikacije“ i „Zaključak“ unutar kojih sumiramo sve što je napravljeno i rečeno unutar završnog rada te navodimo u kojem bi smjeru mogao teći daljnji razvoj web aplikacije i dodavanje novih funkcionalnosti.

3. Web tehnologije i alati

U ovom poglavlju, istražit ćemo širok spektar alata i tehnika koje se koriste u razvoju modernih web aplikacija. Ovaj uvod će pružiti pregled glavnih aspekata web tehnologija, uključujući frontend development, backend development te neke od najpopularnijih tehnologija koje se koriste u ovim područjima.

Frontend development obuhvaća „sve što vidite na web stranici, poput gumba, poveznica, animacija i puno više“ (freeCodeCamp, 2021). „Posao front end programera je preuzeti viziju i koncept dizajna od klijenta i implementirati ih kroz kod“ (freeCodeCamp, 2021). Frontend developeri koriste različite tehnologije, kao što su „HTML (HyperText Markup Language), CSS (Cascading Style Sheets) i JavaScript“ (freeCodeCamp, 2021), kako bi stvorili interaktivne i atraktivne web stranice. HTML se koristi za strukturiranje sadržaja web stranica, CSS za stiliziranje elemenata i JavaScript za dinamične interakcije i manipulaciju sadržajem. Kako bi dodatno olakšali razvoj aplikacija postoje razvojni okviri (eng. framework) za CSS i JavaScript. Neki od CSS razvojnih okvira su Tailwind CSS, Bootstrap i Material UI. Dok su kod JavaScripta tri najpopularnija React.js, Vue.js i Angular.

S druge strane, „backend development znači rad na softveru na strani poslužitelja, koji se fokusira na sve što ne možete vidjeti na web stranici. Backend programeri osiguravaju ispravan rad web stranice, fokusirajući se na baze podataka, pozadinsku logiku, sučelje za programiranje aplikacija (API), arhitekturu i poslužitelje“ (Coursera, 2023). Backend developeri koriste različite tehnologije, poput programskih jezika kao što su Python, Ruby, Java ili PHP. Također, backend development isto uključuje upotrebu različitih okvira i biblioteka, kao što su Express.js, Django ili Ruby on Rails, kako bi se pojednostavio proces razvoja i omogućila brza izgradnja robusnih aplikacija.

Nakon provedenog istraživanja, otkriveno je da za ovaj projekt najbolji izbor tehnologija uključuje React.js i Redux u kombinaciji s Tailwind CSS-om za stiliziranje sučelja. React.js je popularan JavaScript okvir koji omogućuje izgradnju komponentnih korisničkih sučelja s ponovnom upotrebljivošću i efikasnim upravljanjem stanjem aplikacije. Redux se često koristi kao biblioteka za upravljanje stanjem aplikacije, olakšavajući praćenje i ažuriranje podataka između komponenti.

Uz to, za pohranu koda i verzioniranje, odabran je GitHub, popularna platforma za upravljanje izvornim kodom koja omogućuje timsku suradnju i kontrolu verzija. Za bazu podataka i posluživanje (eng. hosting) web stranice odabran je Firebase, koji pruža jednostavno i skalabilno rješenje za pohranu podataka u stvarnom vremenu i brz hosting web aplikacija.

3.1. Tailwind CSS

„Tailwind CSS je uslužni CSS razvojni okvir dizajniran da korisnicima omogući bržu i lakšu izradu aplikacija“ (Hubspot, 2022). Umjesto korištenja unaprijed definiranih stilova i klasa, Tailwind CSS koristi koncept korisnih (eng. utility) klasa koje pružaju direktne stilove i funkcionalnosti.

Jedna od ključnih karakteristika Tailwind CSS-a je veliki skup predefiniranih stilova koji „kontroliraju izgled, boju, razmake, tipografiju, sjene i slično“ (Hubspot, 2022). Umjesto pisanja CSS stilova ručno,

korisnici Tailwind CSS-a mogu koristiti prethodno spomenute korisne klase direktno u HTML-u kako bi primijenili željene stilove na elemente.

Primjerice, umjesto definiranja CSS stila za marginu, kao što je "margin: 10px;", Tailwind CSS omogućuje korištenje korisne klase "m-10" koja će automatski primijeniti željenu marginu na element. Ovakav pristup omogućuje brzo stvaranje i prilagodbu dizajna bez potrebe za pisanjem ili uređivanjem velike količine CSS koda.

Tailwind CSS također podržava reaktivno dizajniranje, omogućujući jednostavno prilagođavanje stilova za različite veličine ekrana i uređaje. Također, moguće je kombinirati korisne klase za stvaranje složenijih stilova, prilagođavanje izgleda i funkcioniranje elemenata prema potrebama projekta.

Kako bi se omogućilo korištenje Tailwind CSS-a u React.js aplikaciji, prvo je potrebno instalirati i konfigurirati framework. Konfiguracija se odvija unutar „tailwind.config.js“ datoteke unutar koje imamo različite izlaze. Ovi izlazi se odnose na tipove datoteka na koje će se Tailwind primijeniti. Stilovi se mogu primijeniti unutar datoteka tipa .js, .html i slično.

Tailwind CSS također omogućuje prilagodbu odnosno mogućnost konfiguriranja stilova, dodavanje vlastitih korisnih klasa ili prilagođavanje postojećih stilova prema potrebama projekta.

Korištenje Tailwind CSS-a u React.js aplikaciji nam je omogućilo brzo i konzistentno stiliziranje sučelja, pružajući bogat skup gotovih stilova i funkcionalnosti koje se mogu lako primijeniti na elemente.

3.2. React.js

„React.js je JavaScript razvojni okvir otvorenog koda (eng. open-source) i biblioteka koju je razvio Facebook. Koristi se za brzu i učinkovitu izgradnju interaktivnih korisničkih sučelja i web aplikacija sa znatno manje koda nego što biste to učinili sa samim JavaScriptom“ (Hubspot, 2022).

Jedna od ključnih karakteristika React.js-a je virtualni DOM (Document Object Model). Sigurno se pitate što je to DOM? Najjednostavnije rečeno to „predstavlja strukturne dijelove web dokumenta kao objekte kojima se može pristupiti i kojima se može manipulirati“ (freeCodeCamp, 2022). Možete ga zamisliti kao hijerarhijsko stablo koje opisuje sve elemente, sadržaje i atribute web stranice. Jedan od izazova s kojima se susrećemo kod manipulacije stvarnim DOM-om je da svaka promjena zahtijeva ponovno iscrtavanje cijelog stabla. Ako imate složenu web stranicu s puno elemenata, ovo može biti resursno zahtjevno i utjecati na performanse. Baš iz tog razloga React.js koristi virtualni DOM kako bi efikasno upravljao ažuriranjem korisničkog sučelja. Umjesto da direktno manipulira stvarnim DOM-om, React.js stvara virtualnu reprezentaciju DOM-a i uspoređuje je s prethodnim stanjem. Samo se promijenjeni dijelovi sučelja ažuriraju u stvarnom DOM-u, čime se postiže visoka učinkovitost i performanse.

React.js omogućuje izgradnju korisničkih sučelja u obliku komponenti. „Komponente su pojedinačni dijelovi konačnog sučelja, koji, kada se sastave, tvore cjelokupno korisničko sučelje aplikacije“ (Hubspot, 2022). Komponente mogu imati vlastito stanje i primati ulazne podatke putem svojih svojstava (eng. props). Ovaj modularni pristup olakšava razvoj, održavanje i testiranje aplikacija.

Kako bismo započeli rad u React.js-u, trebat će nam nekoliko temeljnih alata i koncepta:

1. Node.js: Node.js je potrebno instalirati na računalu kako bismo mogli koristiti npm (Node Package Manager) za upravljanje paketima i izgradnju React.js aplikacija.
2. Kreiranje novog projekta: Možemo koristiti naredbu "npx create-react-app ime-projekta" u naredbenom retku kako bi brzo inicijalizirali novi React.js projekt. Ovo će stvoriti osnovnu strukturu projekta s potrebnim konfiguracijama i datotekama.
3. JSX: JSX je sintaksa slična HTML-u koja se koristi u React.js-u za opisivanje strukture i prikaza korisničkog sučelja. JSX omogućuje ugrađivanje JavaScripta unutar HTML-a, olakšavajući rad s komponentama i njihovim podacima.
4. Komponente: Razumijevanje koncepta komponenata ključno je za razvoj u React.js-u. Potrebno je naučiti kako definirati komponente, kako ih koristiti unutar drugih komponenti, i kako upravljati stanjem i svojstvima komponenti.
5. Stanje (eng. state) i događaji (eng. events): React.js koristi stanje komponenti kako bi upravljao promjenama u korisničkom sučelju. Važno je naučiti kako koristiti stanje komponenti i kako reagirati na događaje kao što su klikovi, unos teksta ili druge interakcije korisnika. Upotreba metoda poput setState omogućuje ažuriranje stanja komponente i ponovno iscrtavanje sučelja kako bi se prikazale promjene.
6. Usmjeravanje (eng. routing): Za izgradnju višestraničnih aplikacija, korisno je koristiti routing. React.js ima nekoliko popularnih biblioteka za upravljanje rutama, poput React Router-a, koji omogućuje navigaciju između različitih stranica i prikazivanje komponenti na temelju trenutne rute.

Kada se upoznamo sa ovim osnovnim konceptima i alatima, možemo započeti izgradnju React.js aplikacija. Važno je pravilno organizirati komponente, pratiti smjernice najbolje prakse, koristiti komponentni pristup i razlučiti odgovornosti između frontend i backend dijela aplikacije.

Uz razumijevanje ovih ključnih aspekata razvoja u React.js-u, bit će lakše izgraditi skalabilne, brze i moderno vizualne web aplikacije.

3.2.1. Redux

U ovom projektu, korišten je Redux i Redux Toolkit kao sredstvo za upravljanje stanjem aplikacije. „Redux je biblioteka za pohranjivanje stanja varijabli u aplikaciji“ (Medium, 2021), a „Redux Toolkit je službeni preporučeni pristup za pisanje Redux logike“ (Redux, 2023). Isto tako „Redux Toolkit uključuje pomoćne funkcije koje pomažu pojednostaviti mnoge uobičajene slučajeve upotrebe, uključujući postavljanje state-a, stvaranje reducer-a i pisanje nepromjenjive logike ažuriranja" (Redux, 2023).

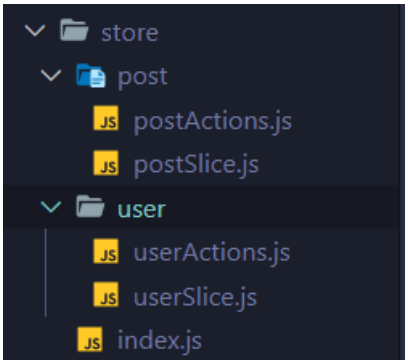
Redux omogućuje spremanje podataka iz aplikacije u jednoj zajedničkoj pohrani, nazvanoj "store". Stanje u Reduxu je nepromjenjivo, što znači da se ne mijenja izravno, već se stvaraju nove kopije stanja kada se izvrše određene akcije. Ove akcije se šalju putem objekata nazvanih "actions" i obrađuju se putem funkcija nazvanih "reducers".

Redux Toolkit pojednostavljuje razvoj Redux aplikacija tako što pruža ugrađene funkcionalnosti kao što su definiranje akcija i reducer-a uz pomoć koda visoke razine apstrakcije. Također pruža dodatne alate

poput "createSlice" za generiranje akcija i reducer-a s manje ponavljajućeg koda, te "createAsyncThunk" za upravljanje asinkronim operacijama. Asinkrone operacije u web aplikacijama su operacije koje se izvršavaju neovisno o ostatku koda, omogućujući paralelno izvršavanje više zadataka kao što su dohvaćanje podataka s udaljenog servera, slanje podataka na server ili izvođenje animacija, čime se postiže bolji odaziv i korisničko iskustvo.

Redux je koristan za složene aplikacije koje imaju puno komponenti koje dijele iste podatke ili kada treba pratiti i upravljati stanjem aplikacije na globalnoj razini. Omogućava lako praćenje promjena stanja, ažuriranje komponenti temeljem tih promjena i olakšava testiranje i održavanje koda.

U ovom projektu, Redux Toolkit je korišten za upravljanje stanjem aplikacije, skladištenje i ažuriranje podataka koji su korišteni za prikaz objava, komentara, glasanja i drugih relevantnih informacija. Redux Toolkit je olakšao organizaciju koda, smanjio količinu pisanja ponavljajućeg koda i pružio intuitivniji način za rukovanje akcijama i reducer-ima.



Slika 1 Prikaz Redux store-a za našu web aplikaciju

Na slici 1 vidimo kako je naš store organiziran. Imamo mapu pod nazivom „store“ unutar koje se nalaze dvije podmape („post“ i „user“) i glavna „index.js“ datoteka koja služi za povezivanje dvije polovice store-a. Svaka podmapa nosi drugačiji naziv jer se one odnose na cjeline koje kontroliraju. Mapa „user“ će po tome imate sve funkcionalnosti i stanja vezana za korisnika, dok će mapa „post“ biti vezana uz post akcije. Ovime postižemo separaciju koda koja je potrebna za korisničke dijelove od dijelova koji su potrebni za objave. Isto tako možemo vidjeti kako svaka od podmapa sadrži dvije datoteke - na primjer „userActions.js“ koja sadrži samo akcije i „userSlice.js“ koja uvodi navedene akcije u „slice“ kako bismo mogli određivati i modificirati stanja poslije izvršenja tih funkcija.

3.2.2. React Router

„React Router je je npm paket koji omogućuje implementaciju dinamičkog usmjeravanja u web aplikaciji“ (Geeksforgeeks, 2023). Ona omogućuje definiranje ruta (URL-ova) i povezivanje tih ruta s odgovarajućim komponentama koje će se prikazati na ekranu. React Router DOM je specifična implementacija React Routera za web aplikacije.

Korist React Routera u Single-Page Application (SPA) aplikacijama je u tome što omogućuje prijelaz između različitih "stranica" (ili dijelova aplikacije) bez da se ponovno učitava cijela stranica. Umjesto toga, samo se mijenja prikazana komponenta u postojećem DOM-u, čime se postiže brža i glatka

navigacija unutar aplikacije. React Router pruža mogućnosti kao što su definiranje ruta, prosljeđivanje parametara ruta, upravljanje povratnim vezama i animacija prijelaza.

„Single-Page Application (SPA) je web-mjesto ili web-aplikacija koja dinamički prepisuje trenutnu web stranicu s novim podacima s web poslužitelja, umjesto zadane metode web preglednika koji učitava cijele nove stranice" (bloomreach, 2018) . Drugim riječima, sve resurse, poput JavaScript datoteka, CSS-a i slika, preuzima se jednom prilikom učitavanja aplikacije, a kasnije se koristi JavaScript za dinamičko mijenjanje prikaza i dohvaćanje podataka bez potrebe za ponovnim učitavanjem cijele stranice. To omogućuje bržu navigaciju i bolje korisničko iskustvo.

S druge strane, „Multiple-Page Application (MPA) su web aplikacije koje se sastoje od velikog broja HTML stranica. Svaka stranica prikazuje drugačiji sadržaj koji se mora osvježiti svaki put kada korisnik s njom stupi u interakciju“ (CleanCommit, 2023). Svaki put kada se korisnik prebacuje na drugu "stranicu", zahtjev se šalje poslužitelju, a poslužitelj vraća novu HTML datoteku koja se prikazuje korisniku. MPA pristup često zahtijeva više vremena za učitavanje i osvježavanje stranica, što može rezultirati sporijim korisničkim iskustvom.

Ukratko, React Router i React Router DOM su alati koji omogućuju implementaciju dinamičke navigacije unutar SPA aplikacija, čime se postiže brža i glatka promjena prikaza bez potrebe za ponovnim učitavanjem cijele stranice.

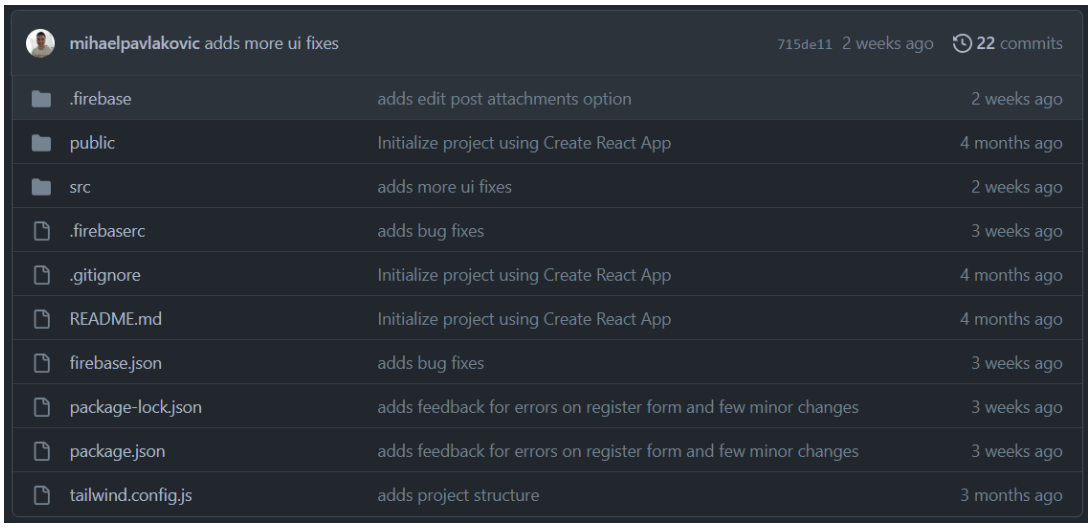
3.3. Github

„Git je besplatni distribuirani sustav za kontrolu verzija otvorenog koda koji je dizajniran za brzu i učinkovitu obradu svega, od malih do vrlo velikih projekata“ (Gitscm, 2023). On omogućuje programerima da prate, upravljaju i surađuju na projektima s drugim članovima tima. Git također omogućuje čuvanje povijesti svih napravljenih promjena u kodu, omogućuje povratak na prethodne verzije, usporedbu promjena i spajanje različitih grana koda.

„GitHub je web stranica i usluga temeljena na oblaku koja programerima pomaže u pohranjivanju i upravljanju kodom, kao i praćenju i kontroli promjena koda“ (Kinsta, 2022). Drugim riječima, omogućuje pohranu Git repozitorija na udaljeni poslužitelj, pristup s bilo kojeg mjesta i dijeljenje s drugim korisnicima. GitHub pruža dodatne značajke kao što su praćenje problema, upravljanje projektima, kolaborativno uređivanje koda i mogućnost pregleda i rasprave o kodu kroz funkcionalnost tzv. „pull“ zahtjeva.

Da biste koristili Git i GitHub za spremanje koda, prvo je potrebno instalirati Git na svoje računalo. Zatim možete stvoriti lokalni Git repozitorij u kojem ćete pratiti i upravljati kodom. Kada napravite promjene u kodu, koristit ćete Git naredbe poput "commit" za potvrđivanje promjena i "push" za slanje tih promjena na udaljeni repozitorij na GitHub-u.

Na slici 2 možemo vidjeti kako naš projekt izgleda kada se postavi na stranice GitHub-a pomoću spomenutih naredbi.



Slika 2 Prikaz izgleda projekta na GItHubu

3.4. Firebase

„Firebase je Backend-as-a-Service (Baas). Pruža razvojnim programerima razne alate i usluge koji im pomažu u razvoju kvalitetnih aplikacija, povećanju baze korisnika i ostvarivanju profita“ (Educative, 2023). Razvijena je od strane Googlea i omogućuje programerima da brzo razvijaju kvalitetne aplikacije bez potrebe za upravljanjem infrastrukturom poslužitelja.

Firebase pruža širok spektar funkcionalnosti koje pokrivaju različite aspekte razvoja aplikacija. Neke od ključnih usluga koje Firebase pruža su:

1. Baza podataka u stvarnom vremenu (Realtime Database): Ova usluga omogućuje sinkronizaciju podataka između klijentskih uređaja i poslužitelja u stvarnom vremenu. To je posebno korisno za aplikacije koje zahtijevaju trenutno ažuriranje podataka, kao što su chat aplikacije ili kolaborativni alati.
2. Autentifikacija korisnika (Authentication): Firebase pruža sigurno i jednostavno upravljanje korisničkim autentifikacijama. Omogućuje podršku za prijavu putem različitih pružatelja identiteta kao što su Google, Facebook, Twitter i e-pošta.
3. Pohrana datoteka (Storage): Ova usluga omogućuje pohranu i upravljanje datotekama u oblaku. Korisnici mogu spremiti slike, videozapise ili bilo koje druge datoteke na Firebase poslužiteljima i lako im pristupiti iz svoje aplikacije.
4. Hosting: Firebase omogućuje jednostavno posluživanje web stranica i statičkih datoteka. Programeri mogu jednostavno objaviti svoje web aplikacije na Firebase poslužitelju i imati ih dostupne na internetu.
5. Push obavijesti (Cloud Messaging): Ova usluga omogućuje slanje push obavijesti na mobilne uređaje. Programeri mogu koristiti Firebase za slanje obavijesti korisnicima svojih mobilnih aplikacija i zadržati ih angažiranima.

Firebase nudi i mnoge druge usluge kao što su analitika, testiranje aplikacija, funkcije u oblaku (Cloud Functions) i više.

3.4.1. Firebase Firestore

„Firestore je NoSQL baza podataka dokumenata izgrađena za automatsko skaliranje, visoku izvedbu i jednostavnost razvoja aplikacija“ (Firestore Docs, 2023).

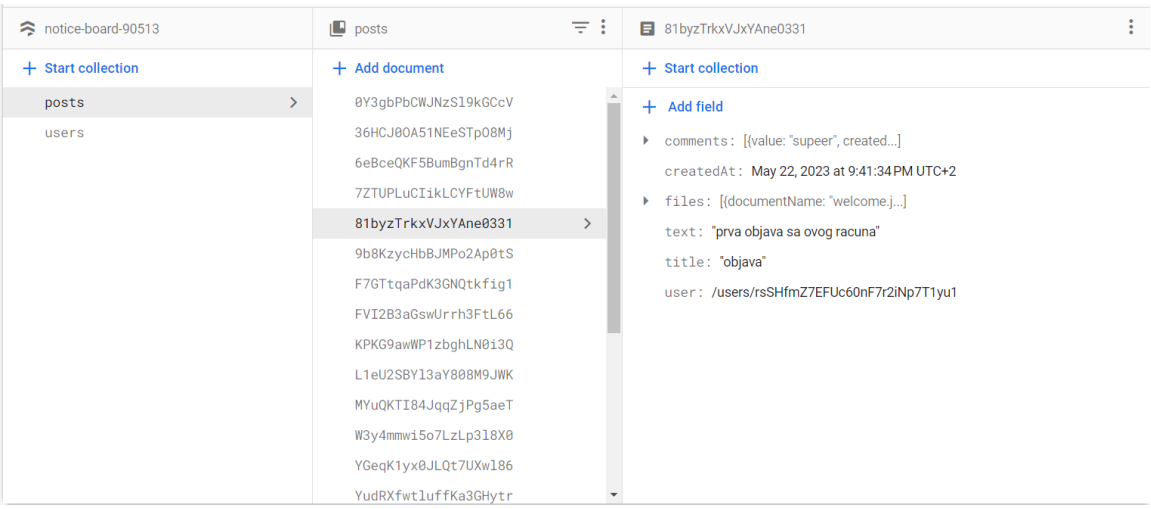
Firestore se temelji na konceptu kolekcija i dokumenata. Kolekcija je skup srodnih dokumenata, dok je dokument jedinstven zapis koji sadrži polja s vrijednostima. Svaki dokument ima jedinstven identifikator i pohranjuje se u kolekciji. Firestore podržava bogatu hijerarhijsku strukturu podataka, a kolekcije i dokumenti se mogu organizirati na način koji najbolje odgovara potrebama aplikacije.

Firestore pruža mogućnost sinkronizacije podataka između klijentskih uređaja i poslužitelja u stvarnom vremenu. To znači da svaka promjena podataka u bazi automatski propagira promjene na sve povezane uređaje. Ova značajka je korisna za aplikacije koje zahtijevaju trenutno ažuriranje podataka, kao što su chat aplikacije, aplikacije za praćenje u stvarnom vremenu ili aplikacije s više korisnika.

Firestore također pruža napredne mogućnosti pretraživanja podataka. Omogućuje postavljanje raznih upita i filtriranje podataka na temelju određenih kriterija. Ovo olakšava dobivanje preciznih rezultata iz velikih skupova podataka i poboljšava performanse aplikacije.

Firestore je integriran s drugim Firebase uslugama, poput autentifikacije korisnika i funkcija u oblaku, što omogućuje programerima da izgrade složene aplikacije s kompletnim backendom u oblaku. Također pruža bogat set biblioteka i SDK-ova za različite platforme i programski jezike, što olakšava razvoj aplikacija u skladu s njihovim potrebama.

Slika 3 prikazuje strukturu naše baze podataka za kreirani projekt. Bazu smo podijelili na dva dijela tako da smo kreirali kolekciju za objave (eng. posts) i kolekciju za korisnike (eng. users). Unutar svake od navedenih kolekcija nalaze se zapisi odnosno dokumenti koji su jedinstveni, a razlikujemo ih prvenstveno po njihovoj *id* vrijednosti. Za kraj, kao primjer smo prikazali kako izgleda struktura jedne objave na stranici. Ona sadrži vrijednosti kao što su *comments*, *createdAt*, *files*, *text*, *title* i referencu na korisnika koji ju je kreirao.



Slika 3 Prikaz Firebase Firestore baze podataka

3.4.2. Firebase Storage

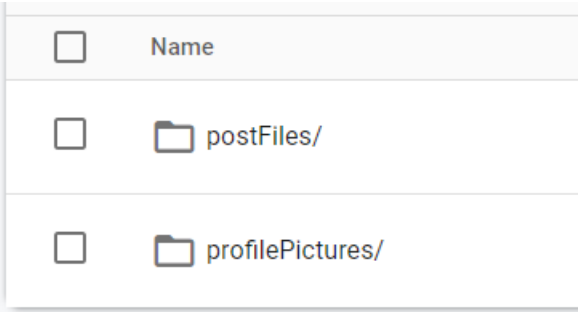
„Firebase Cloud Storage usluga je koju programeri mogu koristiti za pohranjivanje i preuzimanje datoteka koje izravno generiraju klijenti. Nije potreban kod na strani poslužitelja“ (Pullrequest, 2020). To je sigurno i skalabilno rješenje za spremanje različitih vrsta datoteka kao što su slike, videozapisi, dokumenti i drugi mediji.

Firebase Storage pruža jednostavno sučelje i set alata koji olakšavaju rad s datotekama. Programeri mogu jednostavno prenijeti datoteke izravno iz svoje aplikacije na Firebase poslužitelje koristeći Firebase SDK-ove i API-je. Spremljene datoteke dobivaju jedinstvene URL-ove koji se mogu koristiti za prikazivanje, preuzimanje ili dijeljenje datoteka s korisnicima aplikacije.

Jedna od prednosti Firebase Storagea je skalabilnost. Bez obzira na to koliko datoteka korisnici prenose ili preuzimaju, Firebase se brine o skaliranju infrastrukture kako bi osigurao optimalne performanse i dostupnost. Također pruža ugrađene sigurnosne značajke i kontrole pristupa kako bi se osiguralo da samo autorizirani korisnici imaju pristup datotekama.

Korištenje Firebase Storagea olakšava programerima rukovanje datotekama u oblaku bez potrebe za održavanjem vlastite infrastrukture za pohranu. To čini Firebase Storage praktičnim i moćnim alatom za aplikacije koje zahtijevaju pohranu i upravljanje datotekama u oblaku.

Slika 4 prikazuje našu strukturu unutar Firebase Storagea. Imamo dvije odvojene mape gdje svaka ima svoju ulogu. Unutar „profilePictures“ mape pohranjene su profilne fotografije korisnika, na način da je generiran jedinstveni *id* i dodana ekstenzija fotografije tipa .jpg ili .jpeg. Ovime smo osigurali da svaki korisnik ima profilnu fotografiju koja će biti jedinstvenog naziva i neće doći do problema ukoliko 2 korisnika imaju istu fotografiju. Sljedeće, u mapi „postFiles“ pohranjuju se podmape s jedinstvenim id-om koji je vezan uz objavu. Na ovaj način osigurana je lakoća pohrane podataka za svaku objavu i smanjen rizik za pojavu grešaka prilikom same manipulacije istih.



Slika 4 Prikaz strukture Firebase Storege-a

3.4.3. Firebase Hosting

„Firebase Hosting potpuno je upravljana usluga hostinga za statički i dinamički sadržaj kao i mikroservise“ (Google, 2023). To je brzo, sigurno i pouzdano rješenje za objavljivanje web aplikacija i njihovo dostavljanje korisnicima diljem svijeta.

Firebase Hosting podržava posluživanje statičkih datoteka kao što su HTML, CSS, JavaScript, slike i ostali resursi. Firebase Hosting je integriran s Firebase alatima i platformom, omogućujući jednostavnu konfiguraciju i upravljanje web stranicama iz Firebase konzole ili kroz naredbeni redak.

Prednosti Firebase Hostinga uključuju:

1. Brzo učitavanje: Firebase Hosting koristi globalnu mrežu poslužitelja kako bi osigurao brzo učitavanje web stranica. Datoteke se distribuiraju na više geografskih lokacija, što omogućuje korisnicima brz pristup sadržaju bez obzira na njihovu lokaciju.
2. Sigurnost: Hosting koristi HTTPS protokol za osigurano šifriranje komunikacije između korisnika i web stranica. Firebase također pruža ugrađene sigurnosne značajke kao što su certifikati SSL i automatska obnova certifikata.
3. Prilagodljivost: Firebase Hosting omogućuje konfiguriranje prilagođenih domena i usmjeravanja, što omogućuje prilagodbu URL-ova web stranica i upravljanje preusmjeravanjem korisnika.
4. Jednostavno upravljanje: Firebase Hosting se lako integrira s drugim Firebase uslugama kao što su autentifikacija korisnika, Firestore baza podataka ili Firebase funkcije u oblaku. To olakšava integriranje web aplikacija s ostalim Firebase uslugama i upravljanje njima iz jednog mjesta.

Korištenje Firebase Hostinga je jednostavno i ne zahtijeva složenu konfiguraciju poslužitelja ili infrastrukture. Nakon objavljivanja web stranice, Firebase automatski preuzima brigu o distribuciji sadržaja i pružanju visoke dostupnosti.

3.5. Visual Studio Code

„Visual Studio Code (VS Code) besplatan je, lagan, ali moćan uređivač izvornog koda“ (InfoWorld, 2022), razvijen od strane Microsofta. Ovaj iznimno popularan alat postao je prva opcija za mnoge frontend developere zbog svoje izuzetne funkcionalnosti i jednostavnosti korištenja.

Jedna od ključnih značajki VS Codea je njegov moćan i prilagodljiv uređivač kôda. Uz podršku za sintaksno označavanje, automatsko dovršavanje koda, prelamanje teksta i mnoge druge korisne funkcionalnosti, VS Code olakšava pisanje i uređivanje HTML-a, CSS-a i JavaScripta. Ovaj uređivač kôda je intuitivan i prilagodljiv, omogućavajući programerima da budu produktivni i efikasni.

Uz snažan uređivač kôda, VS Code ima integriranu podršku za Git, najpopularniji sustav za kontrolu verzija. Ova integracija omogućuje programerima (eng. developers) da jednostavno prate promjene u kodu, upravljaju granama, provjere povijest promjena i surađuju s drugim članovima tima. VS Code olakšava timski rad i održavanje koda, čineći proces razvoja glatkim i organiziranim.

Jedna od ključnih prednosti VS Codea je njegova proširivost. VS Code je izuzetno fleksibilan i pruža podršku za proširenja koja korisnicima omogućuju prilagođavanje njihovog radnog okruženja. Postoji veliki broj dostupnih proširenja koja omogućuju dodavanje novih značajki, podršku za različite jezike i alate, integraciju s drugim servisima i još mnogo toga. Zahvaljujući ovom bogatom ekosustavu

proširenja, VS Code može se prilagoditi potrebama svakog programera i projekta, čineći radno okruženje jednostavnim i učinkovitim.

Osim toga, VS Code pruža izvrsnu podršku za ispitivanje i otkrivanje pogrešaka frontend aplikacija. Korisnicima omogućuje postavljanje točaka prekida, praćenje vrijednosti varijabli, korak po korak izvršavanje i još mnogo toga. Ova integrirana podrška za ispitivanje pomaže programerima da otklone pogreške i pronađu probleme u kodu, čime se smanjuje vrijeme potrebno za razvoj i poboljšava kvaliteta aplikacija.

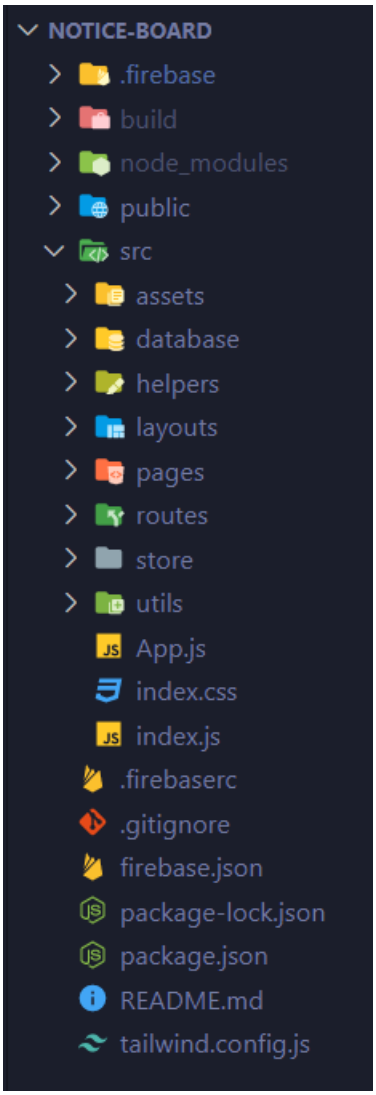
3.5.1. Struktura aplikacije

Nakon što smo objasnili što je to VS Code i koje su njegove značajke sada ćemo analizirati našu strukturu aplikacije. Drugim riječima vidjet ćemo kako su mape i datoteke organizirane da čine smislenu cjelinu unutar koje bi se svaki programer mogao snaći.

Unutar početnog direktorija što je u našem slučaju mapa pod nazivom „NOTICE-BOARD“ možemo pronaći mape poput „.firebase“ koja je konfiguracijska mapa od strane Google Firebase-a za postavljanje web aplikacije na poslužitelj. Potom dolazimo do „build“ mape koja se stvara tek nakon što izradimo ono što smatramo prvom verzijom aplikacije, a potom pokrećemo komandu „npm run build“. Sve unutar navedene mape se postavlja na poslužitelj kako bi se naša web aplikacija pravilno prikazivala. Zatim slijedi „node_modules“ mapa koja je sastavni dio svakog projekta jer unutar nje pronalazimo sve pakete (odnosno skripte/funkcije za pokretanje tih paketa) koji su potrebni za razvoj web aplikacija. Isto tako treba napomenuti kako svaki budući paket koji dodamo u projekt kreira svoju podmapu unutar „node_mpdules“ mape. „public“ mapa sadrži prvu ulaznu točku naše aplikacije, a riječ je o indeks.html datoteci. Uz nju nalaze se naravno i datoteke sa ekstenzijom .css koja sadrži sve naše stilove te ikone ili fotografije koje su potrebne za našu web aplikaciju. Nakon toga slijedi najbitnija mapa unutar koje ćemo provoditi većinu vremena, a to je „src“ mapa. Unutar nje proizvoljno, ali prateći neku strukturu kreiramo svoje datoteke i podmape.

Konkretno ovaj projekt je posložen na sljedeći način: imamo mape koje se odnose na glavne stranice i rasporede istih („layouts“, „pages“ i „utils“), mape koje sadrže funkcije koje se često koriste unutar same aplikacije („database“ i „helpers“), store mapa koju smo objasnili u poglavlju 2.2.1. za Redux store odnosno kreiranje stanja u aplikaciji, rute koje će navoditi korisnika kroz aplikaciju i „assets“ mapa sa fotografijama ili ikonama.

Kako to sve izgleda u VS Codu moguće je vidjeti na slici 5.



Slika 5 Prikaza strukture projekta unutar VS Coda

4. Funkcionalnosti web aplikacije

Unutar ovog poglavlja objasnit ćemo sve što se tiče funkcionalnosti web aplikacije. Počevši sa prijavom ili registracijom korisnika, kako to funkcionira u pozadini, izgled i ponuđene akcije na početnoj stranici (eng. dashboard) te koje su to ostale stranice naše web aplikacije.

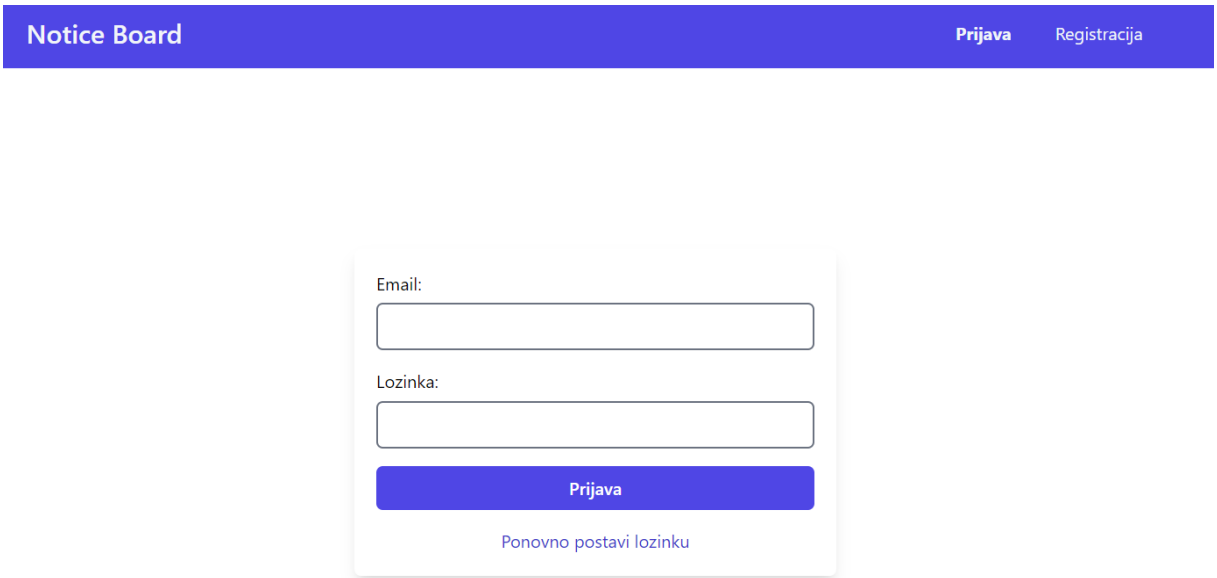
4.1. Prijava i registracija

Prilikom ulaska na web stranicu (prateći link: https://notice-board-90513.web.app/), korisniku se prikazuje sučelje za prijavu korisnika. Navigacijska traka ističe se primarnom bojom kako bi jasno označila dostupne opcije i omogućila korisniku jednostavno kretanje, uključujući prijelaz na stranicu za registraciju ako nema korisnički račun.

Forma za prijavu (slika 6) funkcionira na način da se od korisnika zahtijevaju podatci za prijavu poput emaila i lozinke koje je prethodno odabrao prilikom registracije. Ukoliko korisnik nema prethodno kreiran račun, prilikom pokušaja prijave pojavit će se prikladna greška koja jasno objašnjava da korisnički račun ne postoji te da treba pokušati ponovno ili kreirati novi račun. Slično tome, ukoliko korisnik unese pogrešnu lozinku za postojeći račun, također će mu se prikazati odgovarajuća greška.

Dizajn forme za prijavu slijedi smjernice dobrog dizajna, pri čemu je primarni CTA (Call to Action) gumb istaknut primarnom bojom, privlačeći pažnju korisnika i potičući ga na prijavu. Osim toga, na stranici postoji i opcija za korisnike koji su zaboravili svoju lozinku, a ona se prikazuje u sekundarnoj boji kako bi jasno istaknula manje bitnu radnju i izbjegla zbunjenost korisnika.

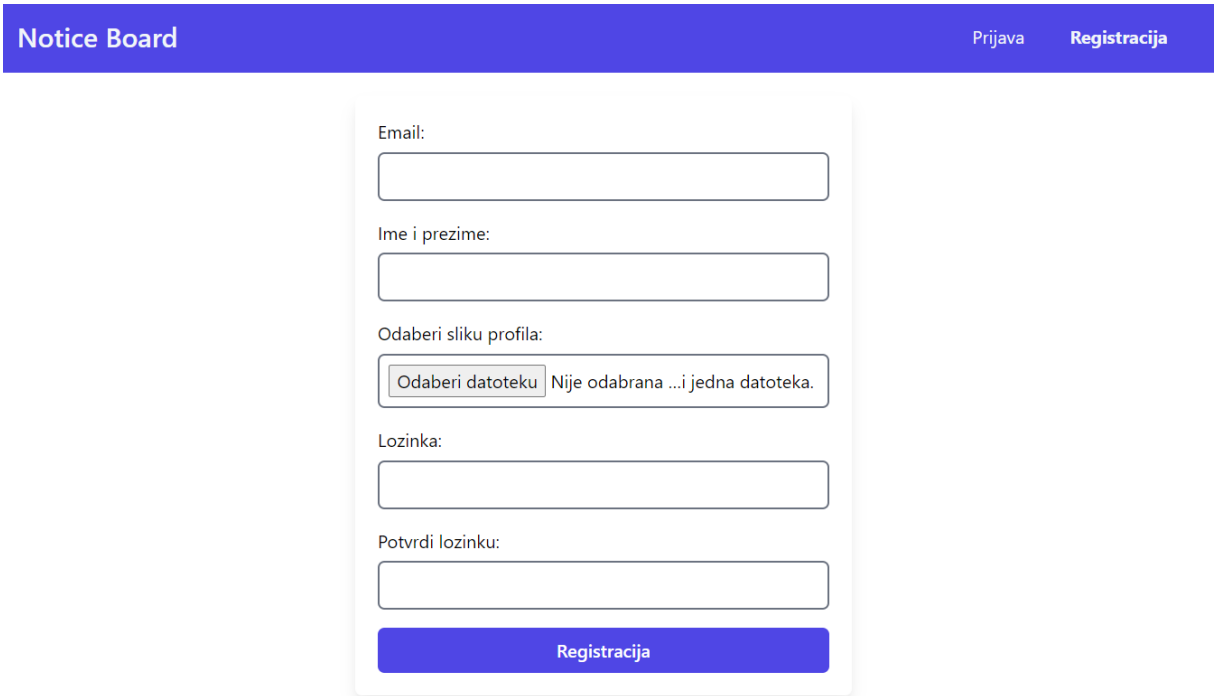
Sve navedene dizajnerske odluke vođene su ciljem stvaranja profesionalnog i dojmljivog korisničkog iskustva. Kombinacija minimalističkog dizajna, upotrebe boja za naglašavanje ključnih elemenata i pridržavanje smjernica dobrog dizajna doprinosi vizualno privlačnom sučelju koje olakšava korisniku prijavu na stranicu i intuitivno navođenje.



Slika 6 Prikaz stranice za prijavu

Na stranici registracije (slika 7) možemo primijetiti dosljedan dizajn koji odražava stil prethodno opisane stranice za prijavu. Međutim, forma za registraciju ima širi raspon polja koja korisnik treba ispuniti. Ova polja uključuju unos e-pošte, imena i prezimena korisnika, odabir slike profila, lozinke i potvrdu lozinke. Sva navedena polja su obavezna kako bi se uspješno izvršila registracija, a korisniku se pružaju povratne informacije ako polja nisu ispunjena ili ako dođe do određenih pogrešaka poput neusklađenosti lozinki, nedostajućih podataka ili odabira pogrešnog formata datoteke za profilnu sliku. Važno je napomenuti da korisniku prikazujemo vizualno odabranu sliku prilikom odabira slike profila, te mu pružamo mogućnost uklanjanja iste ukoliko to želi.

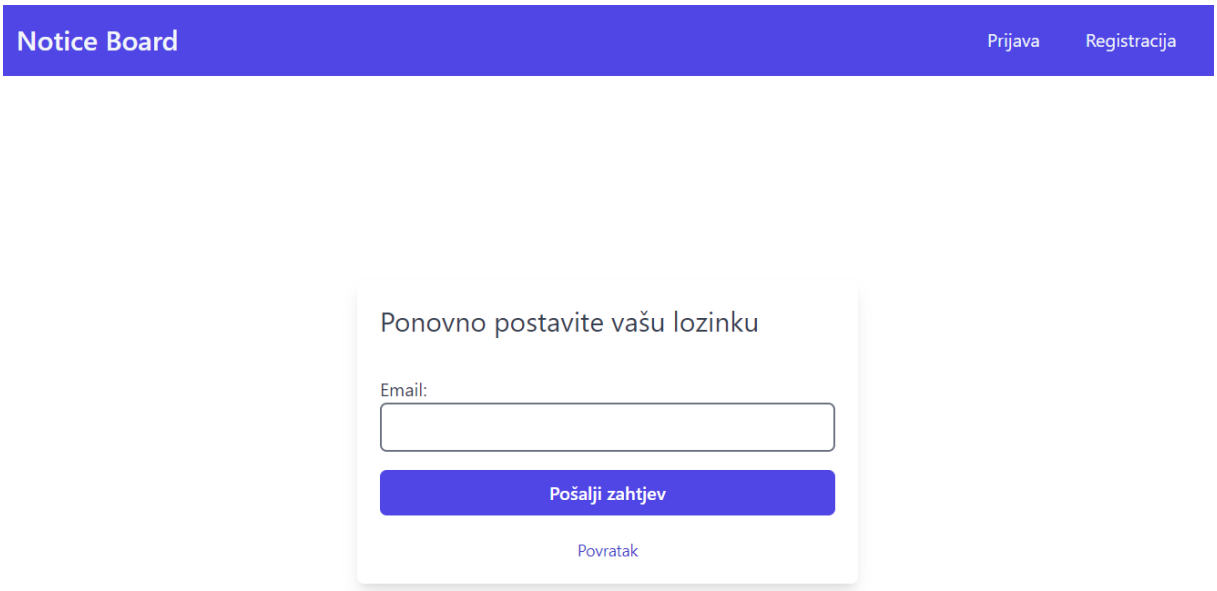
Nakon što korisnik ispuni formu i klikne na gumb „Registracija“, dolazi do promjene stanja gumba kako bi se prikazao napredak postupka. Tekst gumba se mijenja u „U tijeku“...", pružajući korisniku vizualnu indikaciju da se registracija provodi. Također, iznad gumba se prikazuje napredak u obliku progresivne trake (eng. progress bar) kako bi se korisniku pružila dodatna vizualna informacija o učitavanju i pohrani profilne slike u bazu podataka. Nakon uspješne registracije, korisnik se automatski preusmjerava na početnu stranicu aplikacije, koja je označena kao ruta "/".



Slika 7 Prikaz stranice za registraciju

Postupak ponovnog postavljanja lozinke (slika 8) omogućuje korisniku da vrati pristup svom postojećem računu u aplikaciji. Ovaj postupak zahtijeva da korisnik već ima registrirani račun i da je unio valjanu e-mail adresu prilikom registracije. Kako bi započeo postupak ponovnog postavljanja lozinke, korisnik unosi svoju e-mail adresu u odgovarajuće polje u obrascu i pritišće gumb "Pošalji zahtjev".

Nakon što korisnik pošalje zahtjev, Firebase šalje e-mail obavijest na unesenu adresu s uputama za ponovno postavljanje lozinke. Korisnik će primiti poruku koja sadrži relevantne informacije i poveznicu koju treba slijediti kako bi nastavio postupak. Kroz tu poveznicu, korisnik će biti preusmjeren natrag na našu aplikaciju kako bi mogao unijeti novoodabranu lozinku.



Slika 8 Prikaz stranice za ponovno postavljanje lozinke

4.1.1. Prijava i registracija u pozadini

Kako bismo objasnili sve važne dijelove komponenti za prijavu i registraciju prvo ćemo objasniti kako to sve funkcionira prilikom prijave. Što se to sve nalazi unutar navedene komponente i koje akcije i funkcije se pozivaju.

Na slici 9 prikazan je način na koji je organizirana naša komponenta naziva „Login.jsx“. Za početak uvodimo (eng. import) komponentu „Nav.jsx“ koja u sebi sadrži stilove i sadržaj bitan za navigaciju koja se mijenja ovisno o tome da li je korisnik prijavljen ili nije.

Potom slijedi uvod komponente „Card.jsx“ iz „utils“ mape. Ona na sebi ima stil za oblikovanje lijepog ruba oko prikazane forme. Kako bi se sve uredno prikazalo, potrebno je sav sadržaj koji planiramo imati unutar tog ruba staviti između otvorenog i zatvorenog naziva te komponente.

Samim time kao prva stvar u komponenti „Card.jsx“ bit će prikazana greška, ukoliko postoji, sa klasom „text-red-400 mb-2“ iz paketa Tailwind CSSa. Iza toga slijedi već prethodno viđena forma na slici 6, s poljima email i lozinka.

Kako bi se informacije koje je korisnik unio u predviđena polja pohranile, svaki od *input* oznaka na sebi ima vrijednost „onChange“. Ovaj događaj (eng. event) dolazi u sklopu React.js razvojnog okruženja kako bi programeri lakše dohvaćali korisničke informacije. Spomenuti događaj pohranjuje dohvaćene informacije u točno određene varijable koje kasnije prosljeđujemo na daljnju obradu.

```
<>
  <Nav />
  <div className="h-[90dvh] flex justify-center items-center">
    <Card>
      {hasError && (
        <div>
          <p className="text-red-400 mb-2">{error.errorMessage}</p>
        </div>
      )}
      <form
        onSubmit={handleSubmit}
        className="flex flex-col gap-4 w-[15rem] md:w-[25rem]"
      >
        <div className="flex flex-col gap-1.5">
          <label htmlFor="email">Email:</label>
          <input
            className="w-full rounded-md p-2 border-2 border-gray-500 focus:outline-none focus:border-in
          />
        </div>
        <div className="flex flex-col gap-1.5">···
        </div>
        <button
          type="submit"
          className="bg-indigo-600 hover:bg-indigo-700 text-white font-semibold py-2 px-4 rounded-md"
        >
          {status === "loading" ? "U tijeku..." : "Prijava"}
        </button>
        <Link
          to="/promjena-lozinke"
          className="text-indigo-700 hover:underline text-center"
        >
          Ponovno postavi lozinku
        </Link>
      </form>
    </Card>
  </div>
</>
```

Slika 9 Prikaz komponente Login.jsx

Sada dolazimo do logičkog dijela (slika 10) koji se nalazi unutar prethodno navedene „Login.jsx“ komponente. Iz razloga što projekt koristi Redux store, naše komponente bi u pravilu trebale biti dosta jednostavne u smislu da nije potrebno imati puno logike na „prednjem dijelu“ aplikacije. Kod koji gledate na slici 10 se nalazi neposredno iznad html strukture na slici 9. Ovim putem možemo primijetit kako imamo samo jednu pravu funkciju koja se pokreće tek onda kada korisnik na našoj web stranici pritisne gumb „Prijava“.

Ova akcija (pritisak gumba „Prijava“) rezultirat će pokretanjem funkcije „handleSubmit“ koja se nalazi u nastavku. Ova funkcija u sebi ima još jednu funkciju pod nazivom „preventDefault()“ jer želimo spriječiti ponovno učitavanje stranice. Ovo je izrazito bitno jer u SPA aplikacijama većinu takvih stvari je određena od strane programera - drugim riječima korisnika se navodi na točno određene dijelove stranice tako da nema prekinuto iskustvo ponovnog učitavanja stranice..

Sljedeća na redu je „dispatch()“ funkcija koja je ključna za pokretanje naše vlastito definirane „login()“ funkcije unutar Redux stora. Kao što možete primijetiti, ta funkcija prihvaća dva parametra koje smo dobili od korisnika, a to su *email* i *password*. Za kraj postavljamo vrijednosti emaila i lozinke na početno stanje odnosno na praznu vrijednost kako bi korisnik ukoliko se odjavi mogao ponovno ispuniti istu formu.

```
useEffect(() => {
  if (user) {
    navigate("/");
  }
}, [user, navigate]);

useEffect(() => {
  focusElement.current.focus();
}, []);

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

  dispatch(login({ email, password }));
  setEmail("");
  setPassword("");
};
```

Slika 10 Prikaz logičkog dijela komponente Login.jsx

Na slici 11 vidimo prethodno spomenutu „login()“ funkciju unutar Redux store-a. Ovo je asinkrona funkcija jer je potrebno pričekati odgovor od servera, u ovom slučaju Firebasea kako bi dobili željene podatke. Iz podataka koje smo dobili želimo odrediti korisnika koji se prijavio i vratiti njegove vrijednosti „van“ s *return*-om. Ova opisana radnja je omotana u *try catch* blok koji u slučaju da dođe do greške u dohvaćanju korisničkih podataka s danim emailom i lozinkom, hvata dobivenu grešku. S tom greškom možemo ići dalje na *return* ili možemo koristiti *throw* tako da se korisniku prikaže što je to točno pošlo po krivu. Bitno je napomenuti kako svaka funkcija unutar Redux store-a ima svoj *slice* kojemu pripada te njen naziv. Ovo se može vidjeti u liniji gdje piše „user/login“ gdje je *user* ime *slicea*, a *login* naziv funkcije. Taj naziv će nam biti od koristi u sljedećem koraku kada koristimo „extraReducers“ funkciju (slika 12).

```
export const login = createAsyncThunk(
  "user/login",
  async ({ email, password }) => {
    try {
      const { user } = await signInWithEmailAndPassword(auth, email, password);

      return JSON.stringify(user);
    } catch (error) {
      throw error;
    }
  }
);
```

Slika 11 Prikaz funkcije login unutar Redux store-a

Kako bi bolje shvatili što su to "extraReducers“, potrebno je prvo vidjeti koja je to njihova definicija. Reduceri su funkcije koje određuju kako će se promjene primijeniti na stanje aplikacije. Pa tako za konkretni primjer prijave odnosno „login()“ funkcije imamo tri stanja, a to su *pending* dok čekamo odgovor od servera, *fulfilled* kada smo uspješno dobili odgovor i *rejected* ukoliko dođe do greške u komunikaciji.

Pending stanje obično koristimo za prikazivanje *loadera* ili *spinnera* kako bi se korisniku dalo do znanja da se stvari učitavaju odnosno odvijaju. Kod *fulfilled* stanja dobivene podatke sa servera želimo pohraniti u state tako da „payload“ pridružimo varijabli *user*. *Payload* označava varijablu unutar koje se nalaze svi naši podatci koje smo vratili iz funkcije. Nije nužno da se uvijek navedeni *payload* pridružuje nekoj varijabli jer se unutar istog može nalaziti više stvari. I za kraj imamo *rejected* koji služi za hvatanje grešaka i njihovo sortiranje u određenu skupinu kojoj pripadaju. Trenutno su prikazane greške koje se odnose samo na prijavu korisnika.

```
extraReducers(builder) {
  builder
    .addCase(login.pending, state => {
      state.status = "loading";
      state.hasError = false;
      state.error.errorCode = "";
      state.error.errorMessage = "";
    })
    .addCase(login.fulfilled, (state, { payload }) => {
      state.status = "succeeded";
      state.hasError = false;
      state.user = payload;
      state.error.errorCode = "";
      state.error.errorMessage = "";
    })
    .addCase(login.rejected, (state, { error }) => {
      state.status = "failed";
      state.hasError = true;
      console.log(error);
      if (error.message.includes("wrong")) {
        state.error.errorCode = error.code;
        state.error.errorMessage = "Pogrešna lozinka. Pokušajte ponovo!";
      } else if (error.message.includes("user-not-found")) {
        state.error.errorCode = error.code;
        state.error.errorMessage =
          "Korisnik nije pronađen, provjerite ispravnost upisanog maila.";
      } else {
        state.error.errorCode = error.code;
        state.error.errorMessage = error.message;
      }
    })
```

Slika 12 Prikaz reducera za login funkciju

Što se registracije tiče, slijed događaja i izvršavanja koda je identičan uz par bitnih razlika. Unutar komponente pod nazivom „Register.jsx“ imamo puno više polja koja dohvaćaju korisnikove podatke, a samim time imamo i više varijabli koje šaljemo u funkciju „register()“. Isto tako prije samog poziva navedene funkcije vrše se provjere kao što su provjera upisane lozinke, podudaranje s ponovno upisanom lozinkom, sadrži li lozinka malo i veliko slovo te neki broj, jesu li sva polja ispunjena i tako dalje. Ukoliko su sve provjere uspješno prošle dolazimo do store funkcije „register()“ unutar koje kreiramo korisnikov objekt te ga pohranjujemo u bazu podataka (slika 13). Zatim slijedi kreiranje odnosno provjeravanje u kojoj se funkciji stanje nalazi kako bi se korisniku jasno moglo predočiti u kojoj je fazi njegov zahtjev.

```
export const register = createAsyncThunk(
  "user/register",
  async ({ email, password, name, profilePicture }, thunkAPI) => {
    try {
      const { user } = await createUserWithEmailAndPassword(
        auth,
        email,
        password
      );

      const downloadURL = await thunkAPI.dispatch(
        uploadImage({ profilePicture, userId: user.uid })
      );

      await updateProfile(user, {
        displayName: name,
        photoURL: downloadURL.payload,
      });

      const docRef = doc(db, "users", user.uid);
      const userObj = {
        uid: user.uid,
        displayName: name,
        email: email,
        isAdmin: false,
        photoURL: downloadURL.payload,
      };

      await setDoc(docRef, userObj);

      return userObj;
    } catch (error) {
      throw error;
    }
  }
);
```

Slika 13 Prikaz register funkcije unutar storea

4.2. Naslovna stranica

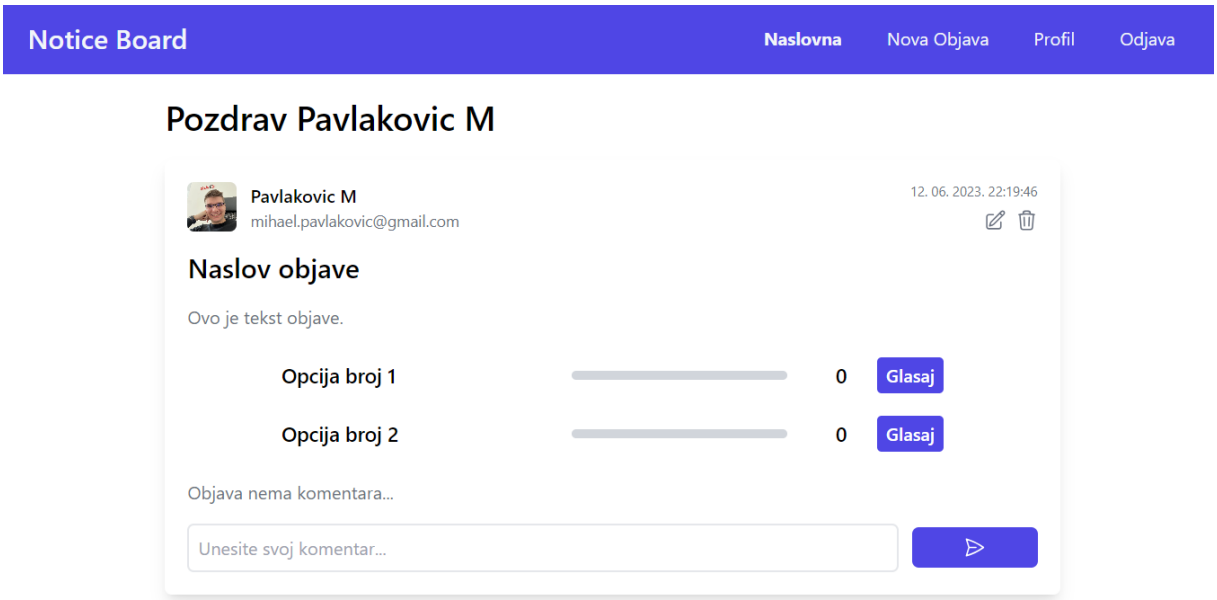
Sada dolazimo do glavne, odnosno naslovne stranice naše web aplikacije. Naslovna stranica se otvara svim korisnicima nakon uspješne prijave ili registracije.

Naslovna stranica isto kao i prethodne sadrži minimalistički odnosno jednostavan dizajn unutar koje korisnik jasno može uočiti sve moguće radnje. Navigacija na vrhu mijenja svoje stanje jer je korisnik autoriziran i sada mu je omogućeno kretanje na ostale stranice tipa „Nova Objava“ i „Profil“. Odmah ispod navigacije, korisnika se dočekuje s porukom dobrodošlice u vidu pozdrava i prikaza njegovog korisničkog imena. U nastavku se prikazuju sve objave koje su napravljene unutar njegove tvrtke. Kao primjer (slika 14) je prikazana objava od prijavljenog korisnika s dodatkom za glasanja. Sama struktura objave je napravljena u tri dijela, a to su zaglavlje, tijelo i kraj koji se odnosi na komentare.

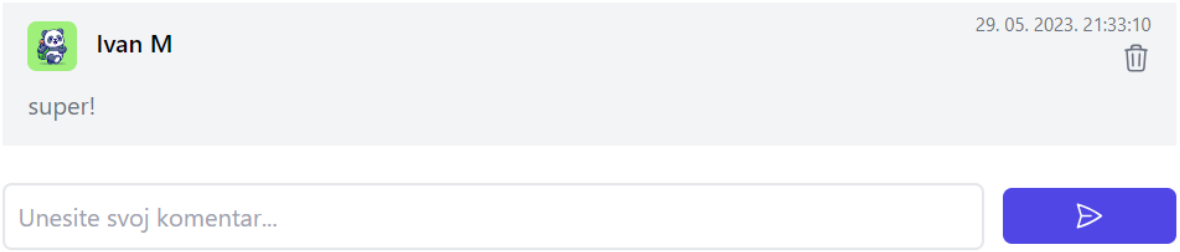
Unutar zaglavlja prikazane su informacije vezane za korisnika koji je kreirao objavu poput slike profila zbog lakšeg prepoznavanja, korisničkog imena i kontakt emaila koji je korišten prilikom registracije. S desne strane se nalazi grupa koja sadrži informacije o vremenu kreirane objave i akcije vezane uz tu objavu. Ponuđene akcije modificiranja objave (lijeva ikonica olovke i papira) i brisanja objave (desna ikonica kante za smeće) su dostupne samo vlasnicima objava. Admini unutar web aplikacije imaju prava na brisanje objava ili komentara ukoliko sadržaj smatraju neprimjerenima. U nastavku ove sekcije objasnit ćemo što se to događa klikom na navedene akcije i kako to funkcionira u pozadini.

Tijelo objave sastoji se od informacija korisnih za čitatelje objava, a to su naslov, tekst objave, dokumente ukoliko su priloženi i glasanje kao u našem slučaju (vidjeti sliku 14). Ukoliko je korisniku ponuđena opcija glasanja, prelaskom pokazivača preko ponuđenih opcija dobiva se jasni vizualni indikator kojoj opciji koji gumb pripada. Ovo je vrlo bitna stavka kako bi korisnici bili sigurni da glasaju za željenu opciju. Isto tako, nakon što se korisnik odluči za glasanje i pošalje svoj glas, gubi se mogućnost daljnje modifikacije odnosno promjene glasa ili ponovnog glasanja. Što se tiče vizualne promjene, korisniku se mijenja gumb „Glasaj“ iz aktivnog stanja u onemogućeno (eng. disabled) sa sivim tonovima primarne boje kako bi se jasno dalo do znanja da je mogućnost ponovnog glasanja nemoguća. Još jedna zanimljivost su trake za napredak koje su prikazane uz ponuđene opcije koje se ispunjavaju ovisno o tome koliko je ljudi glasovalo za pojedine opcije. Ovim vizualnim indikatorom lako možemo uočiti koja opcija vodi i kako će na kraju samo glasanje završiti. Kod priloženih datoteka na objavi možemo istaknuti sljedeće: ukoliko se priloži nekoliko fotografija one će se prikazati kao male ikonice koje je moguće povećati klikom na njih, dok se dokumenti tipa pdf mogu otvoriti u novoj kartici za brži pregled. Svi ostali tipovi datoteka otvaraju se tako da se preuzmu na korisnikovo računalo, a potom otvore u željenom programu.

Zadnji dio su komentari koje ova objava trenutno ne sadrži što je jasno izraženo zamjenskim tekstom „Objava nema komentara…“ i dodatnim input poljem s gumbom za slanje ukoliko korisnik želi ostaviti komentar. Ukoliko objava sadrži komentare, kao što je moguće vidjeti na slici 15, tada će komentar imati prilagođen izgled sličan samoj objavi. Karakteristike komentara su: zaglavlje s profilnom slikom i korisničkim imenom komentatora, informacijama kada je kreiran komentar, akcije za navedeni komentar te sam tekst komentara. Svaki sljedeće dodani komentar nakon prikazanog se dodaje jedan ispod drugoga. Prati se poredak po vremenu ostavljanja komentara isto kao i kod prikaza objava.



Slika 14 Prikaz naslovne stranice s objavom



Slika 15 Prikaz komentara na objavi

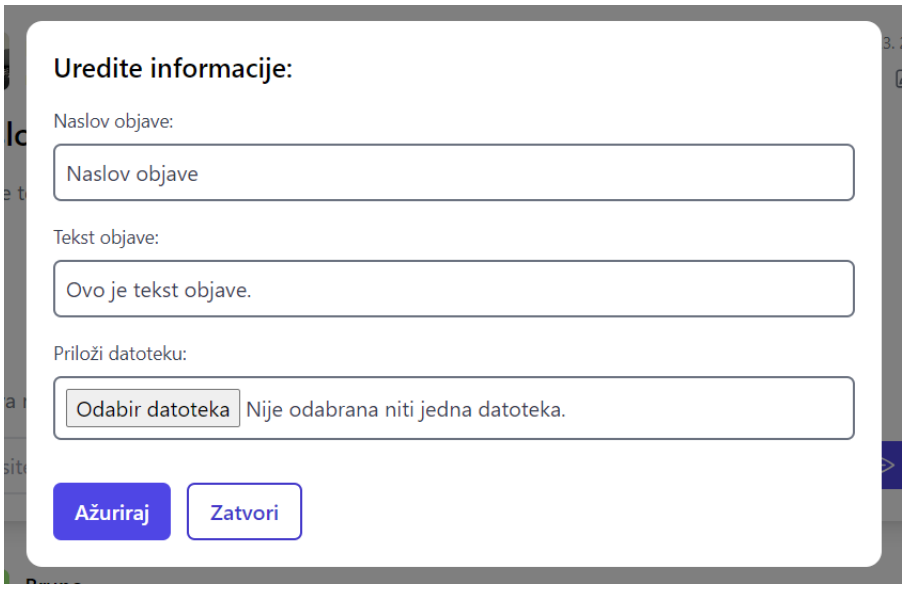
Sada se vraćamo na akcije koje smo spomenuli u prethodnom dijelu koje se nalaze u zaglavlju objave. Konkretno, pritiskom na ikonu olovke i papira što je oznaka za uređivanje postojeće objave, pojavit će se skočni prozor kao na slici 16.

Prozor se sastoji od naslova koji je u ovom slučaju „Uredite informacije:“, *input* polja koja omogućuju izmjenu vrijednosti objave te red akcija s gumbovima „Ažuriraj“ i „Zatvori“.

Input polja su napravljena na način da se prikazuju popunjena s vrijednostima koje je korisnik već prethodno ispunio prilikom kreiranja objave. Na taj je način lakše promijeniti ili restrukturirati objavu.

Ako objava koju uređujemo ne posjeduje priložene datoteke, ali smo se kasnije sjetili kako bismo ipak htjeli nešto priložiti, to je lako moguće napraviti bez kreiranja nove objave. Unutar prikazanog prozora nalazi se polje „Priloži datoteku“ koje je jednako onome koje se nalazi unutar stranice „Nova objava“. Kako to izgleda kada već imamo priložene datoteke moći ćemo vidjeti u sljedećem bloku.

Ukoliko je korisnik zadovoljan s unesenim izmjenama, moći će ih pohraniti klikom na gumb „Ažuriraj“ ili ukoliko je primijetio da ipak ne treba napraviti neke izmjene može lako odustati od toga pritiskom na „Zatvori“.



Slika 16 Prikaz skočnog prozora za uređivanje

U nastavku je prikazana situacija kada objava već ima priložene datoteke (slika 17). Za ovaj slučaj imamo prikazan jedan blok praznog mjesta ispod *input* polja za prilaganje datoteka i prije samih akcija za ažuriranje objave. Ovaj blok je napravljen na način da se provjerava posjeduje li objava priložene datoteke i ukoliko da, prvo prikaže sve fotografije u rešetkastom rasporedu (eng. grid layout) sa 4 stupca prelamajući ostatak u novi red, a potom datoteke tipa pdf, word i slično. Isto tako, ako sve priložene datoteke ne stanu unutar predviđenog mjesta, sa strane će se pojavit klizač koji će omogućiti listanje priloženih datoteka. Priložene datoteke moguće je i ukloniti pritiskom na ikonicu X koja se nalazi unutar crvenog kruga zbog boljeg vizualnog prikaza.

Slika 17 Prikaz skočnog prozora za uređivanje objave s priloženim datotekama

4.2.1. Naslovna stranica u pozadini

Naslovna stranica je dosta kompleksna što se tiče funkcionalnosti i samih akcija koje korisnik može poduzeti. Samim time, kako bismo dobili lakši pregled pozadinskih logika, smatramo da je najbolje podijeliti ih u par sekcija krenuvši od „Feed.jsx“ komponente, potom pregleda akcija za korisnika i admina na objavama, prikaza skočnog prozora, „handleSubmit()“ funkcija skočnog prozora, akcija koje se izvode unutar store-a za pojedine tipove skočnih prozora, glasanja i na samom kraju vidjeti ćemo kako radi ostavljanje i prikaz komentara.

Navedena „Feed.jsx“ komponenta je prva komponenta koja se prikazuje korisniku po registraciji. Ona sadrži komponentu navigacije i sam prikaz objava. Objave se prikazuju tek nakon što zahtjev koji smo poslali prođe kroz naredna tri stanja: *loading*, *failed* i *succeeded*. Za vrijeme *loading* faze prikazana je animirana ikonica učitavanja kako bi se korisniku jasno dalo do znanja u kojoj je fazi njegov zahtjev. *failed* status se pojavljuje ukoliko dođe do pogreške u zahtjevu i korisniku se prikazuje greška, te akcije koje je potrebno poduzeti kako bi se to otklonilo. I za kraj imamo *succeeded* stanje koje nam zapravo i vraća tražene podatke. Po dobivanju podataka potrebno je kreirati zasebne objave, a to ćemo najlakše napraviti korištenjem „map()“ funkcije koja vraća „Post.jsx“ komponentu omotanu „Card.jsx“ komponentom zbog ljepšeg prikaza. Sve opisano možemo vidjeti na slici 18.

```
useEffect(() => {
  dispatch(fetchPosts());
}, [dispatch]);

return (
  <>
    <Nav />
    <div className="mx-auto max-w-[65rem] px-5 sm:px-[8%]">
      <h1 className="text-2xl sm:text-3xl my-5 font-semibold">
        Pozdrav {user?.displayName}
      </h1>
      {status === "loading" && <Spinner />}
      {status === "failed" && <div>Greska: {error}</div>}
      {status === "succeeded" &&
        posts?.map(post => (
          <Card key={post.id}>
            <Post postData={post} />
          </Card>
        ))}
    </div>
  </>
);
```

Slika 18 Prikaz Feed.jsx komponente

Unutar same „Post.jsx“ komponente (slika 19) događa se strukturiranje objave, odnosno kako smo prethodno opisali - stvaranje zaglavlja, tijela i komentara. Kako ne bismo imali previše ponavljajućeg koda koji se odnosi na sam prikaz informacija i oblikovanje pomoću Tailwind CSS klasa, kao zanimljiv dio objasnit ćemo kako se to točno prikazuju ikonice za akcije na objavama.

Navedene ikonice se nalaze na desnoj strani unutar zaglavlja i vidljive su samo korisnicima koji su kreirali objave ili su postavljeni za admine unutar web aplikacije. Ovo je vidljivo na slici 19. Imamo dva uvjeta - prvi dio uvjeta provjerava je li korisnik koji je prijavljen u aplikaciju isti onaj koji je kreirao objavu i ako je to točno, prikazuju mu se dvije ikonice - jedna za uređivanje, a druga za brisanje objave. Drugi dio uvjeta provjerava je li korisnik koji je prijavljen različit od korisnika koji je kreirao objavu i ako ima polje „isAdmin“ – u tom slučaju ima pravo na brisanje svake objave i komentara. Drugim riječima, imat će ikonicu koša za smeće na svakoj objavi i komentaru. Ovaj uvjet se na prvu čini dosta nelogičan jer tražimo korisnika koji nije napravio objavu, ali razlog je vrlo jednostavan. Ako imamo admina koji želi kreirati objave, tada mu i dalje nudimo mogućnost da može urediti svoju objavu, a ne da samo može obrisati istu.

Nadalje vidimo kako svaka od spomenutih ikonica ima svoju „onClick“ akciju koja se izvršava kako bi se dobili željeni rezultati. Da budemo specifični, zanima nas ikona za uređenje koja postavlja *display* opciju na true pomoću „setDisplay“ funkcije.

```
<div className="text-xs text-gray-500">
  <div className="hidden sm:block">{postData.createdAt}</div>
  <div className="mt-2">
    {userData?.uid === postData.user?.uid && (
      <div className="flex gap-2 justify-end">
        <PencilSquareIcon
          className="h-6 sm:h-5 w-6 sm:w-5 hover:cursor-pointer hover:text-indigo-600"
          onClick={() => setDisplay(true)}
        />
        <TrashIcon
          className="h-6 sm:h-5 w-6 sm:w-5 hover:cursor-pointer hover:text-red-500"
          onClick={() => handleDelete(postData.id)}
        />
      </div>
    )}
    {userData?.uid !== postData.user?.uid && userData?.isAdmin && (
      <div className="flex justify-end">
        <TrashIcon
          className="h-6 sm:h-5 w-6 sm:w-5 hover:cursor-pointer hover:text-red-500"
          onClick={() => handleDelete(postData.id)}
        />
      </div>
    )}
  </div>
```

Slika 19 Prikaz strukture Post.jsx komponente

Kada promijenimo stanje *display* varijable aktiviramo blok koda koji se nalazi na slici 20. Ovaj blok će prikazati skočni prozor koji će biti tipa *post*. Ovaj tip je vrlo bitan iz jednog razloga, a to je da prilikom slanja forme koja se nalazi unutar tog prozora možemo lako razlikovati koji zahtjev šaljemo. Trenutačno unutar navedenog prozora mogu biti tri tipa forme, a to su: *post*, *comment* ili *user*. Unutar ove sekcije dotaknut ćemo se *post* i *comment* tipa, dok će *user* tip bit objašnjen u sekciji „Profil u pozadini“.

```
{display && (
  <Modal
    type={"post"}
    isOpen={display}
    index={""}
    itemId={postData.id}
    data={postData}
    onClose={() => setDisplay(false)}
  />
)}
```

Slika 20 Prikaz bloka koda koji prikazuje skočni prozor

Prilikom slanja forme za tip *post*, izvršava se blok koda koji je treći po redu u *if else* uvjetu (slika 21). Unutar nje pozivamo akciju pod nazivom „updatePost“ gdje pružamo tražene parametre poput *postId*, *title*, *text*, *files* i *documentId*. Navedeni parametri će nam pomoć pri pronalasku objave koju želimo modificirat te same izmjene podataka, ali više o tome ćemo u sljedećem dijelu. Na samom kraju nalazi se „.then()“ koji se brine o tome da nakon uspješno odrađenog ažuriranja objave zatvorimo skočni prozor i prikažemo korisniku njegove promjene.

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

  if (files.length > 0) {
    setDisplayProgress(true);
  }

  if (type === "user") {
    dispatch(updateProfileInfo(displayName)).then(() => onClose());
  } else if (type === "comment") {
    dispatch(
      updateComment({ postId: itemId, commentId: index, commentValue })
    ).then(() => onClose());
  } else if (type === "post") {
    dispatch(
      updatePost({
        postId: itemId,
        title: postTitleValue,
        text: postTextValue,
        files: fileObjects,
        documentId,
      })
    ).then(() => onClose());
  }
};
```

Slika 21 Prikaz logičkog dijela Modal komponente

Na slici 22 prikazana je akcija koju pozivamo na slici 21. Unutar nje imamo *try catch* blok za hvatanje grešaka prilikom izvođenja koda u *try* dijelu. Kako bismo uspješno ažurirali objavu potrebno je prvo pronaći objavu koju želimo ažurirati. Ovo radimo na način da koristimo „getDoc()“ funkciju iz paketa firebase, a potom iz dobivenih podataka izvučemo podatke o objavi.

Ako smo ažurirali priložene datoteke unutar objave morat ćemo modificirat našu već definiranu varijablu *files* tako da dodamo nove vrijednosti. Potom kreiramo objekt *updates* koji će sve promijenjene podatke oblikovat u jednu cjelinu koju kasnije funkcijom „updateDoc()“ prenosimo u bazu podataka. Kao *return* vrijednost vraćamo sve podatke koje smo i primili unutar funkcije kako bismo dalje unutar state-a mogli modificirati vrijednosti. Ovime korisnik neće morati ponovno učitavati stranicu da vidi napravljene promjene, ali ako dođe do ponovnog učitavanja baza će dostaviti nove ažurirane vrijednosti.

```
export const updatePost = createAsyncThunk(
  "post/updatePost",
  async ({ postId, title, text, files, documentId }, thunkAPI) => {
    try {
      const postRef = doc(db, "posts", postId);

      const postSnapshot = await getDoc(postRef);
      const docData = postSnapshot.data();
      const currentFiles = docData.files || [];
      const existingDocumentId = docData.documentId;

      const updatedFiles = [...currentFiles, ...files];

      const updates = {
        title: title,
        text: text,
        files: updatedFiles,
        documentId: existingDocumentId ? existingDocumentId : documentId,
      };

      await updateDoc(postRef, updates);

      return {
        postId: postId,
        title: title,
        text: text,
        files: updatedFiles,
        documentId,
      };
    } catch (error) {
      throw error;
    }
  }
);
```

Slika 22 Prikaz updatePost funkcije unutar store-a

Kao što vidimo na slici 21, možemo za tip skočnog prozora postavit *comment* vrijednost koja će se odnositi na izmjenu ostavljenog komentara. Ovime dolazimo do drugog dijela programskog koda koji se nalazi u *if else* uvjetu, a to je pozivanje akcije „updateComment“. Navedena akcija odnosno funkcija koju možemo vidjeti na slici 23, prihvaća podatke *postId*, *commentId* te *commentValue*. Ovime ćemo lagano naći komentar na specifičnoj objavi tako da prvo tražimo objavu koja odgovara vrijednosti *postId*. Korištenjem funkcije „doc()“ iz firebase paketa dobivamo referencu na dokument, a „getDoc()“ funkcijom dohvaćamo vrijednosti dokumenta.

Svi komentari koji se nalaze unutar tražene objave pohranit će se u novu varijablu naziva *updatedComments*. Ovime ćemo dobiti polje koje možemo modificirati tako da prvo pronađemo komentar sa specifičnom *commentId* vrijednosti, a potom pristupimo „value“ vrijednosti i zamijenimo ju novom. Ovom radnjom smo uspješno zamijenili komentar s novom vrijednosti i kasnije to ažurirali u bazi pozivom funkcije „updateDoc()“.

Kao i kod prethodne funkcije za ažuriranje objave i ovdje vraćamo sve dobivene vrijednosti kako bismo mogli ažurirati stanje u state-u.

```
export const updateComment = createAsyncThunk(
  "post/updateComment",
  async ({ postId, commentId, commentValue }, thunkAPI) => {
    try {
      const docRef = doc(db, "posts", postId);
      const commentForChange = await getDoc(docRef).then(
        doc => doc.data().comments
      );
      let updatedComments = [];

      commentForChange.forEach(comment => {
        updatedComments.push({
          ...comment,
        });
      });
      updatedComments[commentId].value = commentValue;

      await updateDoc(docRef, {
        comments: updatedComments,
      });

      return { postId, commentId, commentValue };
    } catch (error) {
      throw error;
    }
  }
);
```

Slika 23 Prikaz updateComment funkcije unutar state-a

Sljedeće je objašnjen dio programskog koda zadužen za omogućavanje glasanja (slika 24). Ovaj blok se prikazuje samo ukoliko je *postData* varijabla postojeća unutar objave. Prikazujemo ju na način da s „map()“ funkcijom vratimo *li* tag za svaku od opcija glasanja. Navedena opcija sadrži elemente kao što je sam naziv opcije, traku napretka koja korisnicima daje do znanja koja opcija vodi i za kraj gumb „Glasaj“ koji ima povezanu „onClick“ akciju na sebi.

Zanimljivo za primijetit je to što gumb „Glasaj“ oblikujemo tako da se razlikuju dva stanja. Prvo je kada korisnik još nije ostavio svoj glas - onda ima primarnu boju, a drugi je kada korisnik glasa i boja se mijenja na „bg-opacity-80“.

```
{postData?.pollOptions && (
  <ul className="w-full md:w-4/5 mx-auto space-y-1">
    {postData.pollOptions.map((option, index) => (
      <li
        key={index}
        className="flex flex-col sm:flex-row items-start sm:items-center hover:bg-indigo-100 p-2"
      >
        <div className="flex-grow">
          <span className="text-lg font-semibold">{option.name}</span>
        </div>
        <div className="flex flex-row items-center w-full sm:w-auto">
          <div className="w-48 h-2 bg-gray-300 rounded-full sm:mx-4">
            <div
              className="h-full bg-indigo-600 rounded-full"
              style={{ width: `${(option.votes / 10) * 100}%` }}
            ></div>
          </div>
          <div className="w-16 text-center">
            <span className="text-lg font-semibold">{option.votes}</span>
          </div>
          <button
            onClick={() => handleVoteAction(postData.id, index)}
            disabled={postData?.totalVotedUsers?.includes(userData?.uid)}
            className={`py-1 px-2 rounded ${
              !postData?.totalVotedUsers?.includes(userData.uid)
                ? "bg-indigo-600 hover:bg-indigo-700 text-white font-semibold"
                : "bg-indigo-600 bg-opacity-80 text-white font-semibold"
            }`}
          >
            Glasaj
          </button>
        </div>
      </li>
    ))}
  </ul>
)}
```

Slika 24 Prikaz strukture za glasanje unutar Post komponente

Prethodno spomenuta „onClick“ akcija kada je pozvana izvodi kod prikazan na slici ispod (slika 25). Ovo je poprilično jednostavna funkcija koja služi za pozivanje akcije unutar store-a pod nazivom „handleVote“. Kako bi navedena akcija radila potrebna su dva parametra - *postId* i *optionIndex*.

```
const handleVoteAction = (postId, optionIndex) => {
  dispatch(handleVote({ postId, optionIndex }));
};
```

Slika 25 Prikaz onClick akcije pod nazivom handleVoteAction

Slika 26, prikazuje akciju koja se poziva na slici 25. Kako bismo smanjili repetitivnost opisivanja/prikaza koda odabran je blok koda koji se nalazi unutar *try* dijela.

```
const userId = auth.currentUser.uid;
const { posts } = thunkAPI.getState().post;

const updatedPosts = posts.map(post => {
  if (post.id === postId) {
    if (!post.totalVotedUsers.includes(userId)) {
      const pollOptions = post.pollOptions.map((option, index) => {
        if (index === optionIndex) {···
        }
        return option;
      });
      const totalVotedUsers = [...post.totalVotedUsers, userId];

      return {
        ...post,
        pollOptions,
        totalVotedUsers,
      };
    }
  }
  return post;
});

const updatedPost = updatedPosts.find(post => post.id === postId);

const postRef = doc(db, "posts", postId);
await updateDoc(postRef, {
  pollOptions: updatedPost.pollOptions,
  totalVotedUsers: updatedPost.totalVotedUsers,
});

return {
  postId,
  pollOptions: updatedPost.pollOptions,
  totalVotedUsers: updatedPost.totalVotedUsers,
};
```

Slika 26 Prikaz funkcije handleVote unutar store-a

Unutar ovog bloka malo smo promijenili način kako pristupamo promjeni vrijednosti prilikom glasanja. Prvo iz state-a dohvaćamo sve objave koje se nalaze u *posts* varijabli. Potom pomoću funkcije „.map()“ prolazimo kroz svaki post dok ne zadovoljimo određene uvjete, a to su da je post koji tražimo jednakog *id*-a kao i *post* kojem želimo ažurirat glasanje. Nakon što smo ga pronašli potrebno je vidjeti da li je korisnik već prethodno glasao na objavi te ukoliko nije idemo dalje u *if* uvjet koji pretražuje sve opcije kod glasanja. Unutar ove „map()“ funkcije tražimo opciju sa specifičnim indeksom koje smo dobili prilikom pozivanja akcije na slici 25. Tek nakon što su zadovoljeni svi uvjeti, modificiramo opciju za glasanje tako da joj pribrojimo jedan glas i u varijablu *votedUsers* pohranimo id korisnika koji je glasao. Ovime ćemo dobiti traku napretka koja se pomaknula za određeni dio u odnosu na prethodno stanje. Zatim dolazimo do toga da moramo ažurirati vrijednost varijable *totalVotedUsers* kako bismo spriječili korisnika da glasa ponovo na istu objavu. Za sam kraj navedenu objavu vraćamo i pohranjujemo u varijablu *updatedPosts*.

Kako bismo mogli ažurirati sve promjene na strani baze odnosno Firebase-a, potrebno je filtrirati *updatedPosts* pomoću „.filter()“ funkcije kako bismo dobili točno specifičan post koji ima traženo glasanje. Povratnu vrijednost ove funkcije pohranjujemo u *updatedPost* u jednini iz razloga što će samo jedna objava odgovarati danom uvjetu. Potom radimo klasično ažuriranje na strani baze uz „updateDoc()“ funkciju kojoj dajemo referencu na objavu koju želimo ažurirati i objekt koji mijenja vrijednosti u bazi.

Iz ove funkcije je također potrebno vratiti određene podatke kojima ćemo moći ažurirati stanje u state-u tako da korisnik istog trenutka vidi promjene.

Sada ćemo opisati kako funkcioniraju komentari i njihov prikaz. Na slici 27 vidimo kako su prikazani komentari. Prvo provjeravamo postoje li uopće neki komentari za prikazati i ako je to netočno prikazuje se zadana vrijednost „Objava nema komentara…“, dok u slučaju da je točno idemo u „.map()“ funkciju. Ovime prolazimo kroz svaki komentar i vraćamo komponentu koja je zaslužna sa strukturiranje i oblikovanje komentara pod nazivom „PostComment.jsx“.

Navedena komponenta sadrži i funkciju „handleCommentDelete“ koju je moguće pozvati iz same komponente, a izvršava se akcija koja se nalazi unutar store-a. Ova akcija pod nazivom „deleteComment“ prihvaća podatke poput *postId* i *commentId*, te služi za uklanjanje željenog komentara. Naravno taj komentar mora biti uklonjen od osobe koja ga je i kreirala.

```
<div>
  {postData?.comments.length === 0 ? (
    <p className="text-gray-500">Objava nema komentara...</p>
  ) : (
    postData.comments.map((item, index) => {
      return (
        <PostComment
          key={index}
          index={index}
          itemId={postData.id}
          comment={item}
          handleCommentDelete={() =>
            dispatch(
              deleteComment({ postId: postData.id, commentId: index })
            )
          }
        />
      );
    })
  )}
</div>
```

Slika 27 Prikaz generiranja komentara

Detaljni prikaz spomenute akcije dan je na slici 28.

Za početak tražimo referencu na dokument, odnosno objavu unutar koje se komentar nalazi. Potom dohvaćamo podatke te objave i pozivamo funkciju „updateDoc()“. Ona ažurira točno traženu objavu tako da ukloni iz polja vrijednost koja odgovara pruženoj *commnetId* vrijednosti. Na kraju je potrebno ažurirati state tako da vratimo vrijednosti koje su i ušle u funkciju.

```
export const deleteComment = createAsyncThunk(
  "post/deleteComment",
  async ({ postId, commentId }, thunkAPI) => {
    try {
      const docRef = doc(db, "posts", postId);
      const comForDelete = await getDoc(docRef).then(doc => doc.data());

      await updateDoc(doc(db, "posts", postId), {
        comments: arrayRemove(comForDelete.comments[commentId]),
      });

      return { postId, commentId };
    } catch (error) {
      throw error;
    }
  }
);
```

Slika 28 Prikaz funkcije deleteComment unutar store-a

Kod kojim je prikazan prozor i akcije kada korisnik ostavlja novi komentar prikazan je na slikama 29, 30 i 31 Prvo imamo obrazac *form* (slika 29) koji prikazujemo korisniku u obliku jednog *input* polja i *button* element koji jasno označuje poziv na akciju.

```
<form
  onSubmit={handleSubmit}
  className="flex flex-row items-center gap-3"
>
  <textarea
    ref={textareaRef}···
    value={comment}
  ></textarea>
  <button
    type="submit"···
  >
    <PaperAirplaneIcon className="h-5 w-5 hover:cursor-pointer mx-auto" />
  </button>
</form>
```

Slika 29 Prikaz forme za ostavljanje komentara

Akcija odnosno funkcija (slika 30) koja će se izvršiti nakon slanja forme je „handleSubmit“. Unutar nje sprječavamo zadane postavke preglednika da osvježi stranicu i nastavljamo na *if* uvjet.

Ovaj uvjet je postavljen tako da korisnik ne može ostaviti prazan komentar na objavi već samo vrijednosti koje imaju neku „težinu“. Ovime smo mislili na vrijednosti koje imaju minimalno jedno slovo ili znak. Kada je uvjet zadovoljen pozivamo akciju „createComment“ koja prihvaća vrijednosti *postId* i *commentValue*.

Za kraj po izvršavanju ove akcije postavljamo *comment* vrijednost na prazan string odnosno „“, kako korisnik ne bi vidio svoj komentar i input polje s tom vrijednosti.

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

  if (comment.trim() !== "") {
    dispatch(createComment({ postId: postData.id, commentValue: comment }));
  }

  setComment("");
};
```

Slika 30 Prikaz akcije koja se izvodi po slanju forme za ostavljanje komentara

Na slici 31 prikazano je kako navedena akcija izgleda unutar store-a. Najprije kreiramo novu vrijednost odnosno objekt *comment* koji popunjavamo s vrijednostima koje smo dali funkciji plus *createdAt* kako bi se moglo prikazati vrijeme kreiranja komentara. Potom koristimo funkciju „updateDoc()“ s kojom ažuriramo specifičnu objavu na kojoj je komentar ostavljen tako da koristimo „arrayUnion()“ funkciju.

Za kraj vraćamo sve vrijednosti koje je naša funkcija primila kako bismo dalje u state-u mogli ažurirati vrijednosti komentara.

```
export const createComment = createAsyncThunk(
  "post/createComment",
  async (commentData, thunkAPI) => {
    try {
      const { postId, commentValue } = commentData;
      const userId = auth.currentUser.uid;
      const userRef = doc(db, "users", userId);

      const comment = {
        value: commentValue,
        user: userRef,
        createdAt: Timestamp.fromDate(new Date()).toDate(),
      };

      await updateDoc(doc(db, "posts", postId), {
        comments: arrayUnion(comment),
      });

      return {
        postId,
        comment,
        user: thunkAPI.getState().user.userData,
      };
    } catch (error) {
      throw error;
    }
  }
);
```

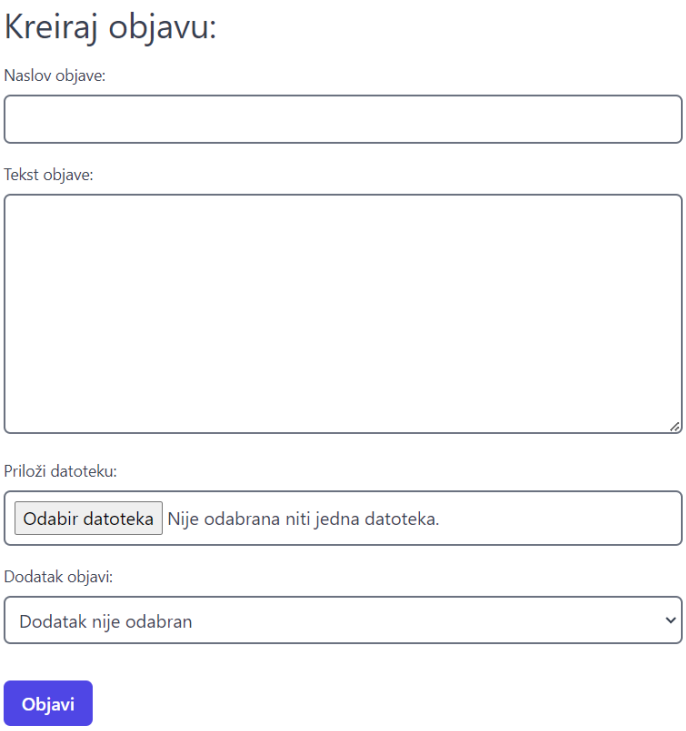
Slika 31 Prikaz funkcije createComment unutar store-a

4.3. Nova objava

Stranica „Nova objava“ (slika 32), sadrži glavnu formu koja je smještena na sredini stranice radi lakšeg vizualnog pregleda i ispunjavanja. Forma nosi naziv "Kreiraj objavu" i prati *input* polja potrebno za stvaranje nove objave.

Da bi objava bila uspješno kreirana, korisnik mora ispuniti obavezno polje – „Naslov objave“. Ovo je polje od velike važnosti pa je stoga obavezno. Ako korisnik želi istaknuti neki kratak tekst koji je bitan za sve, naslov će biti prikazan podebljano na naslovnoj stranici. Također, korisnik ima mogućnost kombiniranja naslova s tekstom objave ili priloženim dokumentima, ili kombiniranja naslova s dodatkom objave poput glasanja.

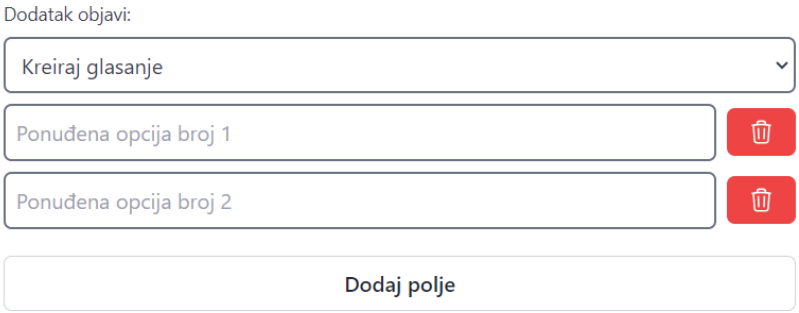
U slučaju da korisnik zanemari obavezno polje naslova, a već je odabrao opciju "Objavi", pojavit će se prikladna greška, a polje u kojem je greška nastala bit će označeno crvenim obrubom.



Slika 32 Prikaz stranice za kreiranje nove objave

Ukoliko korisnik želi dodati glasanje (slika 33), mora prvo otići u sekciju "Dodatak objavi" i odabrati "Kreiraj glasanje". Nakon odabira te opcije, pojavit će se gumb "Dodaj polje" koji će generirati dva prazna polja klikom na njega. Korisnik ima zadatak ispuniti ta polja, a ako ne želi dodavati više polja, jednostavno može kreirati novu objavu s unesenim podacima.

Važno je napomenuti da se prilikom svakog sljedećeg dodavanja polja pojavljuje samo jedno *input* polje za tekst glasanja. Razlog tome je ograničenje da se glasanje može kreirati samo ako postoje barem dvije opcije i isto tako ako su te opcije ispunjene. Ako korisnik pokuša objaviti glasanje, a navedeni zahtjevi nisu ispunjeni, dobit će detaljno obavijest o tome što je pošlo po krivu i kako to riješiti.



Slika 33 Prikaz kreiranja glasanja u novoj objavi

4.3.1. Nova objava u pozadini

Kako bismo bolje shvatili što se događa u pozadini prilikom kreiranja objave, krenut ćemo od objašnjavanja akcije koja proizlazi iz pritiska gumba „Objavi“.

Gumb „Objavi“ tipa *submit* aktivira funkciju u *form* tagu pod nazivom „handleSubmit“. Ova funkcija će se izvršiti samo ukoliko dođe do podnošenja zahtjeva uz klik na prethodno navedeni gumb.

Kod funkcije „handleSubmit“ prikazan je na slici 34.. Prvo moramo spriječiti zadane postavke preglednika da osvježi stranicu prilikom podnošenja zahtjeva. To developeri također trebaju imati na umu. Drugo, na redu su razne provjere kako bi se utvrdilo je li korisnik ispunio sve uvjete kako bi se željena objava mogla kreirati. U te uvjete ulazi provjera je li naslov objave koji je obavezan ostao prazan ili je korisnik možda unio samo razmak kao naslov, je li korisnik odabrao glasanje, ali je zaboravio da mora imati minimalno dvije ponuđene opcije ili situacija kada je korisnik odabrao dvije opcije, ali nije unio naslov jedne od opcija. Svaki od navedenih uvjeta ukoliko nije ispunjen će rezultirati time da se varijabla *hasError* postavlja na vrijednost *true*.

Za posljednju provjeru imamo uvjet koji provjerava ukoliko navedena varijabla *hasError* sadrži grešku. Drugim riječima, varijabla mora biti negativna ukoliko želimo nastaviti dalje na akciju.

Ova akcija je definirana unutar store-a pod *post* dijelom. Pozivamo ju tako da uvezemo funkciju sa točno danim imenom i kao objekt joj pružimo sve prikupljene podatke iz korisnikove forme. Podatci koje smo prikupljali su: *postTitle*, *postText*, *files* i *pollOptions*, odnosno naslov objave, tekst objave, dokumenti koji su priloženi i opcije glasanja.

„.then()“ dio koji se izvršava tek onda kada se ispune sve radnje unutar „createPost“ funkcije, služi za navigaciju korisnika na početnu stranicu (odnosno „/“ rutu) i resetiranje svih varijabli vezano za formu. S ovim resetiranjem osiguravamo da korisnik ima čista polja koja će ispunjavati s novim podatcima ukoliko želi kreirati još jednu objavu.

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

  let hasError = false;

  if (files.length > 0) {
    setDisplayProgress(true);
  }

  if (postTitle.trim() === "") {
    setTitleError("Naslov ne smije ostati prazan.");
    hasError = true;
  }

  if (selectedValue === "option1") {
    if (pollOptions.length <= 1) {
      setOptionError("Morate imati minimalno 2 opcije.");
      hasError = true;
    } else if (!pollOptions.every(option => option.name.trim() !== "")) {
      setOptionError("Morate popuniti sva polja za opcije glasanja.");
      hasError = true;
    }
  }

  if (!hasError) {
    dispatch(createPost({ postTitle, postText, files, pollOptions })).then(
      () => {
        navigate("/");
        dispatch(resetUploadProgress());
        setPostTitle("");
        setPostText("");
        setFiles([]);
      }
    );
  }
};
```

Slika 34 Prikaz handleSubmit funkcije unutar komponente

Na slici 35 možemo vidjeti kod koji se izvodi kada se pozove funkcija „createPost()“ iz „CreatePost.jsx“ komponente.

Prvo je definiran naziv funkcije i kojem *slice-u* ona pripada (radi se o *post sliceu*). Potom imamo klasičan *try catch* blok za hvatanje grešaka prilikom izvođenja potrebnih radnji.

Uvjet koji slijedi provjerava ima li *files* ključ koji se nalazi u objektu neku dužinu, odnosno sadrži li neke datoteke koje bi se trebale prenijeti u bazu podataka. Naravno, ukoliko navedeni *files* ključ nema dužinu onda se smatra kako korisnik nije priložio datoteku te se preskaču naredne radnje. Za korisnike koji su priložili datoteke moramo prvo napraviti određenu vrstu transformacije prije nego pokrenemo akciju „uploadFiles“. Isto tako, uz *files* varijablu unutar koje smo pohranili navedene promjene, akciji pridružujemo i *documentId* varijablu koja služi za imenovanje posebne mape specifične za tu objavu. *documentId* dobijemo tako da pozovemo funkciju „generateId()“ koja u stvarnosti zapravo vraća samo vrijednost oblika 17ebd960-3f89-4491-a2e6-0ca005097e50. Za ovo smo koristili popularni paket „uuid4“ koji prethodno moramo instalirati pokretanjem naredbe „npm install uuid4“. Ukoliko kreiranje mape s navedenim imenom i prijenos datoteke u istu prođe bez grešaka, navedenu radnju spremamo u *uploadTask* varijablu. Kako bismo bili sigurni da ova varijabla sadrži željene vrijednosti (povratne

informacije Firebase Storage-a), koristimo *await* opciju. . Spomenuta opcija se koristi u slučaju asinkrone komunikacije kada je bitno da pričekamo odgovor kako bismo mogli izvršiti daljnje radnje. Potom informacije koje smo pričekali spremamo unutar *uploadResult* varijable koju potom filtriramo tako da dobijemo *downloadURL* i *documentName* za svaki od priloženih datoteka.

```
export const createPost = createAsyncThunk(
  "post/createPost",
  async (data, thunkAPI) => {
    try {
      const userId = auth.currentUser.uid;
      const userRef = doc(db, "users", userId);
      const documentId = generateId();
      let downloadURLs = [];
      let postData = {};

      if (data.files.length > 0) {
        const files = Array.isArray(data.files)
          ? data.files
          : Object.values(data.files);

        const uploadTask = thunkAPI.dispatch(
          uploadFiles({ files, documentId })
        );

        const uploadResult = await uploadTask;

        if (uploadResult.payload) {
          downloadURLs = uploadResult.payload.map(file => {
            const { downloadURL, documentName } = file;
            return { downloadURL, documentName };
          });
        }
      }
```

Slika 35 Prikaz createPost funkcije unutar store-a

Sada dolazimo do drugog dijela „createPost()“ funkcije (slika 36) unutar koje imamo još jedan *if else* uvjet koji se odnosi na to je li korisnik kreirao glasanje. Ukoliko je odgovor pozitivan odnosno *pollOptions* je duži od nula, tada kreiramo uz sve potrebne varijable i varijablu *pollOptions* i *totalVotedUsers*. Unutar varijable *pollOptions* pohranjujemo podatke kao što su naslovi opcija glasanja i jednu praznu varijablu *votedUsers* koja će se modificirati tako da se korisnik koji glasa zapisuje unutar nje. Ovime omogućujemo prikaz drugim čitateljima objave da vide trenutno stanje glasanja pored svake od opcija. Sljedeće je na redu *totalVotedUsers* varijabla koja će isto biti modificirana tako da se spremaju korisnici koji glasaju na objavi, ali konkretno će služiti za provjeru je li korisnik već glasao i ukoliko je, neće mu bit dozvoljeno mijenjanje odabira ili ponovno glasanje.

Na samom kraju, objekte koje smo kreirali unutar *if else* uvjeta dodajemo u kolekciju *posts* unutar naše Firestore baze podataka.

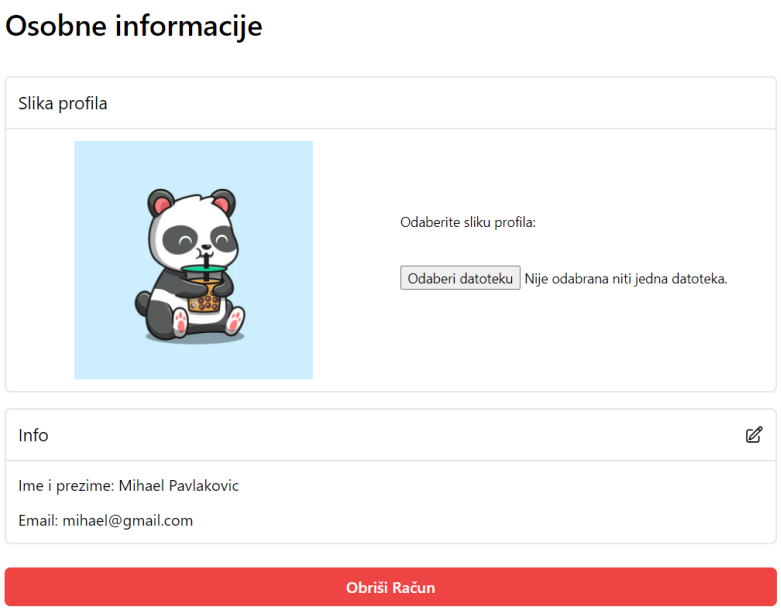
```
      if (data.pollOptions.length > 0) {
        postData = {
          title: data.postTitle,
          text: data.postText,
          createdAt: Timestamp.fromDate(new Date()).toDate(),
          documentId: data.files.length > 0 ? documentId : "",
          files: downloadURLs,
          user: userRef,
          pollOptions: data.pollOptions,
          totalVotedUsers: [],
        };
      } else {
        postData = {
          title: data.postTitle,
          text: data.postText,
          createdAt: Timestamp.fromDate(new Date()).toDate(),
          documentId: data.files.length > 0 ? documentId : "",
          files: downloadURLs,
          user: userRef,
        };
      }

      await addDoc(collection(db, "posts"), postData);
    } catch (error) {
      throw error;
    }
  }
);
```

Slika 36 Prikaz createPost funkcije unutar store-a 2. dio

4.4. Profil

Sada ćemo opisati posljednju stranicu, a to je „Profil.jsx“ (slika 37). Ova stranica korisnicima nudi mogućnosti uređivanja profilne fotografije, uređivanja korisničkih informacija te brisanja računa ukoliko korisnik više ne želi imati otvoren račun na našoj internoj aplikaciji.

Ukoliko korisnik želi obrisati svoj korisnički račun, može odabrati opciju "Obriši Račun". Prije nego što se račun obriše, korisniku se prikazuje dodatna provjera kako bi bilo sigurno da zaista želi izbrisati račun. Nakon potvrde brisanja, korisnika se preusmjerava na prijavnu stranicu, ali ukoliko koristi stare podatke za prijavu, prikazuje se greška o nepostojećem računu.

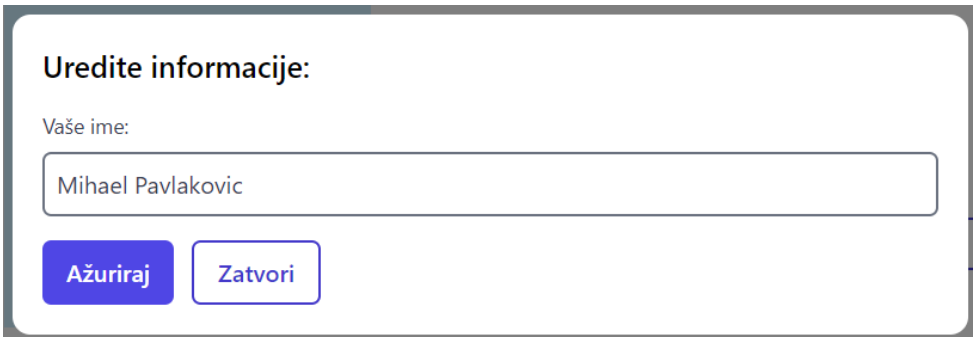


Slika 37 Prikaz profilne stranice

Ukoliko želimo promijeniti sliku profila, korisnik odabire gumb "Odaberi datoteku" (možemo vidjeti na slici 38) koji će otvoriti prozor s fotografijama na njihovom računalu. Nakon odabira fotografije, stanje aplikacije se mijenja te korisniku prikazuje naziv odabrane fotografije i dva gumba - jedan za primjenu promjena i drugi za odustajanje i odbacivanje fotografije. Po pritisku na gumb "Primijeni", korisniku se prikazuje traka za napredak i nakon što doseže 100%, profilna fotografija se mijenja, a stanje se resetira na prvobitnu vrijednost kao kada smo prvi put pristupili stranici profila.

Slika 38 Prikaz ažuriranja slike profila

Za uređivanje osobnih podataka, korisnik može kliknuti na ikonu olovke i papira u bloku „Info“, koja otvara novi prozor s jednim poljem za izmjenu - polje za ime i prezime (slika 39). Korisnik može unijeti željene promjene te ih spremiti pritiskom na gumb "Ažuriraj".



Slika 39 Prikaz skočnog prozora za izmjenu informacija profila

Navedene funkcionalnosti stranice Profil pružaju korisniku potpunu kontrolu nad svojim profilom, omogućujući im personalizaciju i upravljanje svojim podacima u skladu s njihovim potrebama.

4.4.1. Profil u pozadini

Što se tiče programskog koda koji radi u pozadini ove profilne stranice imamo nekoliko zanimljivosti na koje ćemo se osvrnuti.

Na slici 40, možemo vidjeti varijablu *types* unutar koje smo definirali koje sve tipove profilnih fotografija prihvaćamo, a to su *.png*, *.jpeg* i *.jpg*. Ovu varijablu pozivamo unutar funkcije „handleSelect()“ koja provjerava ima li odabrana datoteka dopušteni nastavak i ukoliko ima postavljamo ju u lokalnu varijablu „image“. Ako navedeni kriterij nije zadovoljen, izvađa se drugi dio *if* uvjeta koji postavlja grešku koju kasnije prikazujemo korisniku.

Kako smo odabrali željenu fotografiju za sliku profila potrebno ju je i prenijeti na server, a to se odvija tek kada korisnik odabere opciju „Primjeni“. Navedeni gumb je povezan sa izvođenjem funkcije „uploadImageHandler()“ koji kao prvi korak prikazuje traku napretka, a potom i poziva funkciju unutar store-a pod nazivom „updateProfilePicture()“. Navedena funkcija kao parametar zahtjeva fotografiju koju možemo dobiti iz lokalne varijable *image*.

```
const types = ["image/png", "image/jpeg", "image/jpg"];

const reset = () => {
  setDisplayProgress(false);
  refValue.current.value = "";
  setImage(null);
  dispatch(resetUploadProgress());
};

const handleSelect = e => {
  let selected = e.target.files[0];

  if (selected && types.includes(selected.type)) {
    setImage(selected);
    setError("");
  } else {
    setError("Odaberite sliku formata .png, .jpeg ili .jpg");
    reset();
  }
};

const uploadImageHandler = () => {
  setDisplayProgress(true);
  dispatch(updateProfilePicture(image)).then(() => {
    if (status === "succeeded") {
      reset();
    }
  });
};
```

Slika 40 Prikaz logičkog dijela komponente Profil

Prateći funkciju "updateProfilePictur“()" (slika 41) koja se nalazi unutar datoteke "userActions.js", imamo cjelovit pregled svih koraka koje treba poduzeti kako bi korisnik uspješno ažurirao svoju profilnu fotografiju.

Prvo, koristimo klasični *try catch* blok koji hvata sve eventualne greške koje se mogu pojaviti tijekom izvođenja radnji.

Prvi korak je prijenos fotografije na Firebase Storage kako bismo dobili *downloadURL*. To postižemo pozivanjem funkcije "uploadImage()" unutar iste datoteke "userActions“js". Ova funkcija prima parametar *profilePicture*, koji dobijemo iz datoteke "Profile.jsx" kada korisnik klikne na gumb "Primijeni", i *userId* koji dobijemo iz trenutno aktivne sesije korisnika. Kada se prijenos fotografije uspješno završi, dobivamo povratnu informaciju u obliku spomenutog *downloadURL-a*, koji kasnije postavljamo u korisnikov profil koristeći funkciju "updateProfile()", te ga također postavljamo u Firestore koristeći funkciju "updateDoc()".

Na kraju, vraćamo *downloadURL* kako bismo ga mogli postaviti u sam *user* state unutar reducera. Ovom radnjom postižemo vizualnu promjenu kojom korisnik odmah vidi ažuriranu fotografiju bez potrebe za ponovnim učitavanjem stranice.

Kombinacijom svih ovih koraka, korisnik će moći jednostavno ažurirati svoju profilnu fotografiju, a sustav će se brinuti o prenošenju slike, ažuriranju relevantnih podataka i osiguravanju trenutnih promjena na korisničkom sučelju.

```
export const updateProfilePicture = createAsyncThunk(
  "user/updateProfilePicture",
  async (profilePicture, thunkAPI) => {
    try {
      const user = auth.currentUser;

      // Upload the photo to Firebase Storage
      const downloadURL = await thunkAPI.dispatch(
        uploadImage({ profilePicture, userId: user.uid })
      );

      await updateProfile(user, {
        photoURL: downloadURL.payload,
      });

      // Update the profile picture URL in the users collection
      const userRef = doc(db, "users", user.uid);
      await updateDoc(userRef, {
        photoURL: downloadURL.payload,
      });

      return downloadURL.payload;
    } catch (error) {
      throw error;
    }
  }
);
```

Slika 41 Prikaz updateProfilePicture funkcije unutar store-a

Sada dolazimo do dijela ažuriranja korisnikovih osobnih informacija kao što je primjerice korisničko ime. U prethodnoj sekciji vidjeli smo kako se klikom na ikonu olovke i papira odnosno za uređivanje informacija o korisniku otvara novi skočni prozor. Navedeni prozor dio je „Modal.jsx“ komponente koje smo već prethodno spomenuli, ali u ovom slučaju imamo tip za korisnika odnosno kako smo mi to nazvali *user*. Unutar komponente, kada korisnik pritisne gumb „Ažuriraj“ izvađa se „handleSubmit()“ funkcija unutar koje pozivamo akciju iz „userAction.js“ datoteke. Specifično, radi se o „updateProfileInfo()“ funkciji koja je zaslužna za promjenu podataka korisničkog profila.

Navedena funkcija prima podatak *displayName* koji koristimo unutar *try catch* bloka. „updateProfile()“ je funkcija iz Firebase paketa koja služi za ažuriranje korisničkih podataka tako da joj

pružimo referencu na trenutno prijavljenog korisnika i podatke koje želimo promijeniti. Potom je potrebno te iste podatke odnosno podatak promijeniti i u bazi podataka u kolekciji *users*. Kao *return* vrijednost vraćamo *displayName* kako bi se mogla modificirati vrijednost unutar store-a. Ovime dobijemo promjenu koju korisnik odmah može vidjeti bez da prethodno mora ponovno učitati stranicu. Sve opisane radnje moguće je vidjeti na slici 42.

```
export const updateProfileInfo = createAsyncThunk(
  "user/updateProfileInfo",
  async (displayName, thunkAPI) => {
    try {
      const user = auth.currentUser;

      await updateProfile(user, {
        displayName: displayName,
      });

      const userRef = doc(db, "users", user.uid);
      await updateDoc(userRef, {
        displayName: displayName,
      });

      return displayName;
    } catch (error) {
      throw error;
    }
  }
);
```

Slika 42 Prikaz updateProfileInfo funkcije unutar store-a

5. Budućnost web aplikacije

Kroz daljnje razvojne korake i optimizaciju, možemo unaprijediti performanse i funkcionalnosti aplikacije te je prilagoditi zahtjevima suvremenog web okruženja. U ovom poglavlju ćemo analizirati potrebne korake kako bismo postigli te ciljeve, uključujući optimizaciju performansi, doradu funkcionalnosti i mogućnost migracije na poseban hosting s drugim tipovima baza podataka.

Jedan od ključnih aspekata daljnjeg razvoja aplikacije je optimizacija performansi. S obzirom na stalni napredak tehnologija i rastuće očekivanje korisnika u pogledu brzine i odziva, potrebno je provesti dublju analizu i optimizaciju kako bi aplikacija funkcionirala na najvišoj razini. To može uključivati primjenu tehnika kao što su caching podataka, optimizacija upita prema bazi podataka, kompresija i minimizacija resursa te upotreba CDN-a (Content Delivery Network) za brže učitavanje statičkih

datoteka. Kao još jedan ključan korak se može navesti provođenje redovitih ažuriranja i održavanja. Ubrzan tehnološki napredak zahtijeva kontinuirano praćenje novih verzija i poboljšanja korištenih tehnologija, kao i popravke eventualnih sigurnosnih propusta. Redovito provođenje ažuriranja omogućuje nam da ostanemo u koraku s najnovijim trendovima i pružimo korisnicima najbolje moguće iskustvo.

Osim optimizacije performansi, postoji mogućnost dorade i unapređenja funkcionalnosti aplikacije. Na temelju povratnih informacija korisnika i daljnjeg istraživanja tržišta, možemo identificirati nove značajke koje bi korisnicima pružile dodatnu vrijednost i poboljšale njihovo iskustvo korištenja. Primjerice, možemo razmotriti dodavanje mogućnosti privatnih poruka između korisnika, dodatne opcije filtriranja i sortiranja objava, integraciju s drugim platformama ili uslugama. Od postojećih funkcionalnosti možemo dodatno razraditi opcije objava tako da dodamo grafikone za razne vizualizacije. Isto tako možemo doraditi profil korisnika, tako da se prikazuju sve korisnikove objave i razne interakcije s drugim objavama. Ovime omogućujemo korisnicima odlazak na profil željene osobe te pretraživanje ili poduzimanje raznih akcija.

Također, važno je razmotriti migraciju aplikacije na poseban hosting s drugim tipovima baza podataka. Ovisno o potrebama aplikacije i rastućem broju korisnika, možemo istražiti različite opcije kao što su skalabilni hosting pružatelji ili upotreba cloud usluga za upravljanje bazama podataka. Time bismo osigurali skalabilnost, pouzdanost i efikasnost u upravljanju podacima te oslobodili resurse za fokusiranje na razvoj novih značajki.

Važno je uložiti napore u poboljšanje korisničke podrške i interakcije. Slušanje korisničkih povratnih informacija i pružanje brzih i kvalitetnih odgovora na njihove upite ključno je za izgradnju dugoročnih odnosa s korisnicima. Razmatranje implementacije sustava za podršku korisnicima, kao što su chatbotovi ili ticketing sustavi, može znatno poboljšati korisničko iskustvo i povećati zadovoljstvo korisnika.

U budućnosti, također treba razmotriti mogućnost proširenja na mobilne platforme. Mobilni uređaji postaju sve popularniji u pristupu web sadržaju, stoga bi razvoj mobilne verzije naše aplikacije omogućio korisnicima da pristupaju i koriste aplikaciju na bilo kojem uređaju. Ovo bi zahtijevalo prilagodbu sučelja i funkcionalnosti za mobilno okruženje te pažljivo testiranje kako bi se osigurao dosljedan i intuitivan korisnički doživljaj na svim platformama.

Konačno, treba istaknuti da je budućnost aplikacije uvelike povezana s brzim razvojem tehnologija poput umjetne inteligencije, strojnog učenja i analitike podataka. Integracija ovih tehnologija može donijeti brojne prednosti, poput personaliziranog sadržaja i preporuka, prediktivne analitike i boljeg razumijevanja korisničkih potreba. Stoga, istraživanje i implementacija ovih tehnologija u skladu s potrebama aplikacije mogu otvoriti nova područja rasta i unaprijediti korisničko iskustvo.

6. Zaključak

Tijekom izrade ovog završnog rada, prošli smo kroz različite faze razvoja web aplikacije, počevši od temeljite analize tržišnih potreba prilikom izrade aplikacija za internu komunikaciju i oglašavanje. Svakom koraku smo posvetili pažnju kako bismo osigurali da naša aplikacija zadovoljava sve zahtjeve korisnika. Proveli smo istraživanje različitih tehnologija na frontendu kako bismo odabrali one koje su najprikladnije za našu svrhu, uzimajući u obzir performanse, fleksibilnost i podršku zajednice. Kao rezultat toga, poboljšali smo naše znanje i vještine u tim tehnologijama, a istovremeno smo stekli iskustvo u kreiranju sofisticiranih komponenti s mnogo funkcionalnosti.

Detaljno smo razradili sve funkcionalnosti naše aplikacije, stvarajući intuitivan i korisnički prijateljski dizajn koji omogućuje korisnicima da se lako poistovjete s aplikacijom. Bez obzira na to koji web preglednik korisnici koriste, naša aplikacija pruža konzistentno i pouzdano iskustvo, prilagođavajući se različitim ekranima i rezolucijama. Unutar aplikacije smo implementirali mehanizme za internu komunikaciju i oglašavanje kako bi korisnici mogli lako dijeliti informacije i surađivati s kolegama.

Posebnu pažnju smo posvetili postavljanju i upravljanju bazom podataka. Pohranjivanje objava, priloženih datoteka i korisničkih profila, vršimo na siguran način kako bismo osigurali integritet podataka i zaštitu privatnosti korisnika. Tijekom razvojnog procesa, suočili smo se s raznim izazovima i naučili kako rješavati probleme koji se javljaju prilikom rada s podacima na ovakvoj razini.

Kroz završetak ovog projekta, primijetili smo značajan napredak u znanju i vještinama. Naučeno je prilagođavati se kompleksnim zahtjevima projekta, uključujući izazove u izradi sofisticiranih komponenti i stiliziranju koristeći Tailwind CSS. Kroz provedbu CRUD (Create, Read, Update, Delete) funkcionalnosti, razumjeli smo važnost osnovnih operacija nad podacima i njihovu primjenu u praksi. Sve ovo predstavlja vrijedno iskustvo koje nadograđuje našu stručnost i doprinosi našem profesionalnom portfelju. Kroz ovaj rad smo uspješno zaokružili i objedinili svoje znanje koje smo stekli tijekom diplomskog studija na Fakultetu informatike i digitalnih tehnologija.

Važno je naglasiti da je ova web aplikacija samo početak putovanja kao web developer-a. Sada smo opremljeni znanjem, iskustvom i vještinama koje možemo primijeniti na budućim projektima. U svijetu brzog tehnološkog napretka, kontinuirano učenje i praćenje najnovijih trendova bitno je za uspjeh u ovom području.

U zaključku, ovaj diplomski rad nam je pružio priliku za primjenu teorije u praksi i stjecanje praktičnih vještina. Kroz analizu, dizajn, implementaciju i testiranje ove web aplikacije, razvili smo se kao stručnjaci u području web razvoja. Ponosni smo na postignute rezultate i vjerujemo da će ova aplikacija biti korisna i pružiti olakšanje u internoj komunikaciji i oglašavanju korisnicima.

Za kraj još želimo istaknuti zahvalnost svima koji su nam pružili podršku i pomogli tijekom izrade ovog diplomskog rada, uključujući našeg mentora, kolege, prijatelje i različite online resurse. Rad na ovom projektu je bio izazovan, ali i iznimno ispunjavajući. Spremni smo nastaviti učiti, rasti i primjenjivati svoje znanje kako bismo doprinijeli razvoju web tehnologija i stvaranju inovativnih rješenja u budućnosti.

7. Literatura

1. freeCodeCamp, How to Set Up VSCode for Your React Projects
   URL: https://www.freecodecamp.org/news/vscode-react-setup/ (Pristupljeno: 12.6.2023)
2. Firebase, Cloud Firestore
   URL: https://firebase.google.com/docs/firestore (Pristupljeno: 15.6.2023)
3. Firebase, Get started with Firebase Hosting
   URL: https://firebase.google.com/docs/hosting/quickstart (Pristupljeno: 14.6.2023)
4. React, Quick Start
   URL: https://react.dev/learn (Pristupljeno: 16.6.2023)
5. Tailwind CSS, Get started with Tailwind CSS
   URL: https://tailwindcss.com/docs/installation (Pristupljeno: 15.6.2023)
6. Bitbucket Support, Basic Git commands
   URL: https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html (Pristupljeno: 15.6.2023)
7. Laptop.hr, Najbolje aplikacije za internu komunikaciju,
   URL: https://www.laptop.hr/office/najbolje-aplikacije-za-internu-komunikaciju (Pristupljeno: 25.6.2023)
8. Hr-gadget-info.com, Slack alternative: 10 najboljih alata za komunikaciju u timu
   URL: https://hr.gadget-info.com/78236-slack-alternatives-10-best-tools-for-team-communication (Pristupljeno: 25.6.2023)
9. Lider, 11 ultimativnih aplikacija za male poduzetnike,
   URL: https://lidermedia.hr/aktualno/11-ultimativnih-aplikacija-za-male-poduzetnike-26032 (Pristupljeno: 25.6.2023)
10. Education-wiki, Što je Front End developer? - Vodič za uloge, vještine i rast karijere,
    URL: https://hr.education-wiki.com/7570843-what-is-front-end-developer (Pristupljeno: 29.6.2023)
11. Beekeeper, 5 Benefits of Using Internal Communications Software
    URL: https://www.beekeeper.io/blog/5-benefits-of-using-internal-communications-software/ (Pristupljeno: 29.6.2023)
12. freeCodeCamp, Front End Developer – What is Front End Development, Explained in Plain English,
    URL: https://www.freecodecamp.org/news/front-end-developer-what-is-front-end-development-explained-in-plain-english/ (Pristupljeno: 29.6.2023)
13. Coursera, What Does a Back-End Developer Do?,
    URL: https://www.coursera.org/articles/back-end-developer (Pristupljeno: 30.6.2023)
14. Hubspot, Tailwind CSS: What It Is, Why Use It & Examples,
    URL: https://blog.hubspot.com/website/what-is-tailwind-css (Pristupljeno: 2.7.2023)
15. Hubspot, What is React.js? (Uses, Examples, & More),
    URL: https://blog.hubspot.com/website/react-js (Pristupljeno: 2.7.2023)
16. freeCodeCamp, What is the DOM? The Document Object Model Explained in Plain English,
    URL: https://www.freecodecamp.org/news/what-is-the-dom-explained-in-plain-english/ (Pristupljeno: 4.7.2023)

17. Medium, What Is Redux?,
URL: https://medium.com/swlh/what-is-redux-b16b42b33820 (Pristupljeno: 6.7.2023)
18. Redux, Why Redux Toolkit is How To Use Redux Today,
URL: https://redux.js.org/introduction/why-rtk-is-redux-today (Pristupljeno: 6.7.2023)
19. Geeksforgeeks, What is react-router-dom ?,
URL: https://www.geeksforgeeks.org/what-is-react-router-dom/ (Pristupljeno: 5.7.2023)
20. Bloomreach, What Are Single Page Applications and Why Do People Like Them So Much?,
URL: https://www.bloomreach.com/en/blog/2018/what-is-a-single-page-application
(Pristupljeno: 3.7.2023)
21. CleanCommit, Single Page Application (SPA) vs Multi Page Application (MPA): Which Is The Best?,
URL: https://cleancommit.io/blog/spa-vs-mpa-which-is-the-king/ (Pristupljeno: 3.7.2023)
22. Kinsta, What Is GitHub? A Beginner's Introduction to GitHub,
URL: https://kinsta.com/knowledgebase/what-is-github/ (Pristupljeno: 5.7.2023)
23. InfoWorld, What is Visual Studio Code? Microsoft's extensible code editor,
URL: https://www.infoworld.com/article/3666488/what-is-visual-studio-code-microsofts-extensible-code-editor.html (Pristupljeno: 2.7.2023)

8. Popis priloga

Uz ovaj rad, prilažem web adresu aplikacije Notice Bord i repozitorij na kojemu je pohranjen kod aplikacije. Web adresa aplikacije je: https://notice-board-90513.web.app/. Repozitorij je pohranjen na adresi: https://github.com/mihaelpavlakovic/notice-board. Ova aplikacija predstavlja rezultat razvoja i pruža mogućnost pristupa različitim funkcionalnostima aplikacije za internu komunikaciju i oglašavanje.

9. Popis slika