Aplikacija za stvaranje i vođenje kvizova

Kaučić, Matija

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni diplomski studij Računarstvo

APLIKACIJA ZA STVARANJE I VOĐENJE KVIZOVA

Diplomski rad

Matija Kaučić

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Matija Kaučić
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D-1130R, 13.10.2020.
JMBAG:	0165071403
Mentor:	izv. prof. dr. sc. Ivica Lukić
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 1:	izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 2:	izv. prof. dr. sc. Mirko Köhler
Naslov diplomskog rada:	Aplikacija za stvaranje i vođenje kvizova
Znanstvena grana diplomskog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	Potrebo je izraditi desktop aplikaciju za vođenje kvizova koristeći Electron i Reacta. Funkcionalnosti aplikacije podrazumijeva obrazac za prijavu timova, upis bodova za svaki krug i automatski zbroj ukupnih bodova i sortiranje. Također je potrebno napraviti vizualizaciju bodova po krugovima, statistike timova i samih kvizova.
Datum ocjene pismenog dijela diplomskog rada od strane mentora:	13.09.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	1.10.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	01.10.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 01.10.2024.

Ime i prezime Pristupnika:	Matija Kaučić
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1130R, 13.10.2020.
Turnitin podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Aplikacija za stvaranje i vođenje kvizova**

izrađen pod vodstvom mentora izv. prof. dr. sc. Ivica Lukić

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD

Pub kvizovi postaju, iz dana u dan, sve češća pojava u svakodnevnom životu. Skoro pa ne postoji veći grad u Hrvatskoj u kojemu se oni, u bilo kojem obliku, ne održavaju. Kako kvizovi postaju popularniji, ljudi nastoje olakšati proces stvaranja i vođenja kvizova pa tako nastaju razne web stranice i aplikacije koje se bave tom tematikom. U Hrvatskoj, najpoznatija stranica je ona HKS-a [1], odnosno Hrvatskog kviz saveza, koja ima ogromnu količinu podataka, od repozitorija pitanja, državnih rang lista pa sve do lokacija provođenja kvizova, no ne nudi opciju igranja kvizova *online*.

Prema priči sa stranice Hrvatskog kviz saveza [2], početci kvizova su sredinom 70-ih godina 20. stoljeća kada su Sharon Burns i Tom Porter formirali poslovno partnerstvo kako bi doveli ljude u pubove na dane kad pubovi obično imaju malo prometa. Najprije su osnovali i organizirali 32 pub kviz ekipe koje su igrale tri lige nedjeljom navečer po jugu Engleske, a onda su nekoliko godina putovali po državi objašnjavajući pubovima zbog čega je to dobro za njih. Već 80-ih godina 20. stoljeća, uz pikado i biljar, kvizovi su bili standardna ponuda britanskih pubova, a da bi došli u Hrvatsku trebalo je proći još dvadesetak godina. Oni kombiniraju socijalni aspekt druženja ljudi, edukacijski aspekt koji podrazumijeva učenje novih stvari te natjecateljski koji je možda i glavni razlog nagle popularizacije kvizova.

U ovom će se radu objasniti postupak izrade aplikacije za stvaranje i vođenje kvizova korištenjem React *JavaScript* biblioteke. Također, biti će izrađeno i grafičko korisničko sučelje (engl. *Graphical User Interface* - GUI) pomoću Electron razvojnog okvira (engl. *Framework).*

U drugom poglavlju opisano je trenutno stanje razvoja aplikacija za stvaranje i vođenje kvizova. U trećem poglavlju opisane su teorijske osnove te tehnologije i tehnike korištene za potrebu izrade aplikacije. U četvrtom poglavlju opisan je razvoj aplikacije za stvaranje i vođenje kvizova, arhitektura same aplikacije, glavne komponente i implementacija ključnih funkcionalnosti. U petom poglavlju prikazani su rezultati rada i analiza performansi aplikacije. U zadnjem poglavlju, odnosno zaključku, prikazan je sažetak postignuća, jedinstvene značajke aplikacije, ograničenja trenutne verzije te smjernice za budući razvoj.

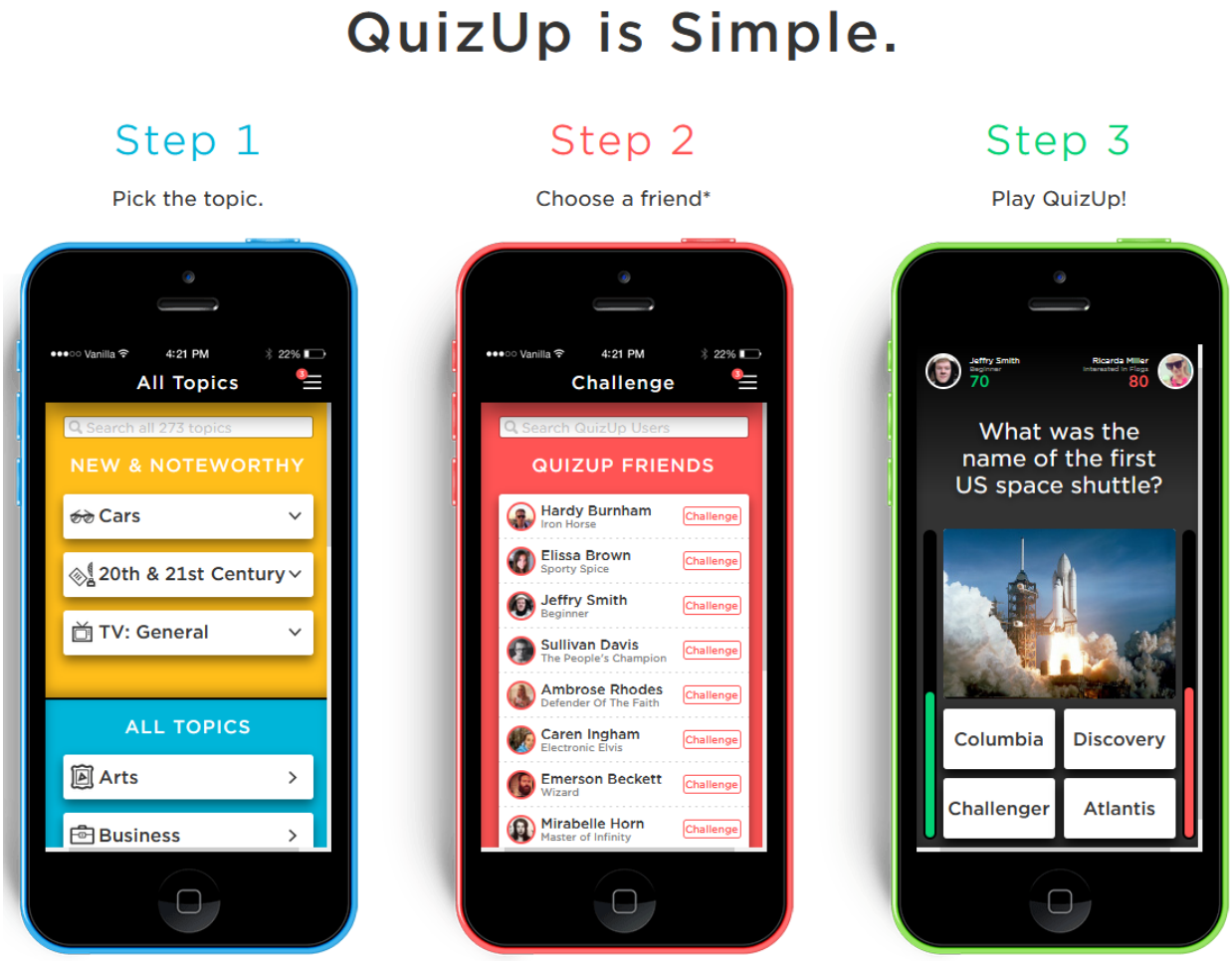
1.1. Zadatak diplomskog rada

U radu je potrebno opisati postupak izrade aplikacije za stvaranje i vođenje kvizova. Aplikacija mora sadržavati princip rukovanja pitanjima, baratanje ekipama, vođenje kvizova u stvarnom vremenu i spremanje rezultata kvizova.

2. PREGLED PODRUČJA TEME

Kviz aplikacije su jedne od najbrže rastućih aplikacija zbog svoje jednostavnosti, ali i edukativnog i kompetitivnog karaktera. Krase ih vrlo intuitivna korisnička sučelja kao i dovoljno jednostavna, ali i zanimljiva pitanja.

Po mnogima preteča popularnih kviz aplikacija bio je proizvod iz 2013. godine islandske razvojne kompanije Plain Vanilla Games koji je razvio QuizUp [3] koji je napravljen kao mobilna aplikacija (slika 2.1.), a u prvih osam dana postojanja proglašena je najbrže rastućom iPhone igricom [4]. Nažalost, u veljači 2021. godine, objavljeno je da će QuizUp biti uklonjen s App Store-a i da će aplikacija biti prekinuta [5].

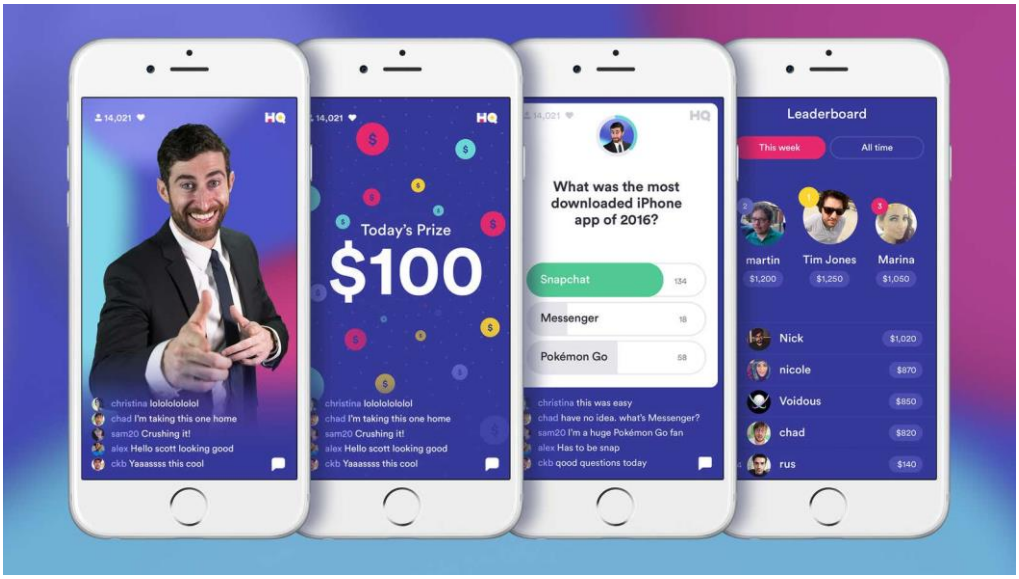


Slika 2.1. *Korisničko sučelje aplikacije QuizUp*

Jedna od najpoznatijih aplikacija te tematike je Trivia Crack (slika 2.2.) koja je vrlo slična QuizUp-u u smislu mogućnosti igranja protiv prijatelja i drugih korisnika, no razlikuje se po tome što su kategorije pitanja odabrane vrtnjom kola s kategorijama.

Slika 2.2. *Korisničko sučelje aplikacije Trivia Crack*

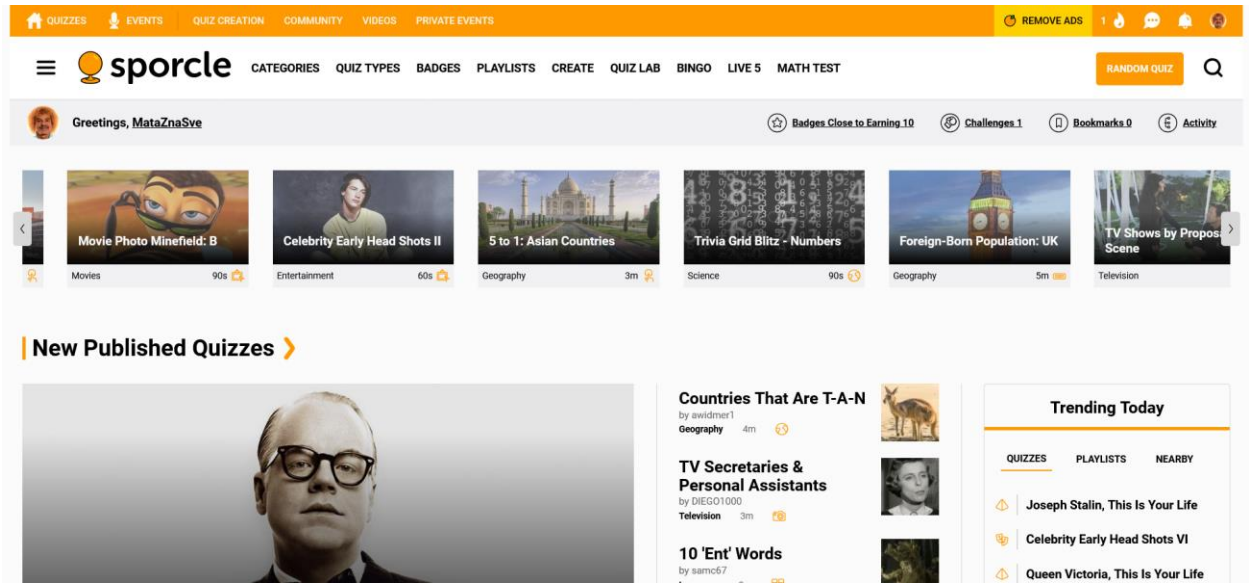
HQ Trivia (slika 2.3.) je također mobilna aplikacija koja je igrana u stvarnom vremenu, gdje nekoliko puta dnevno, voditelj zabavlja i komunicira sa sudionicima te čita pitanja i ponuđene odgovore nakon kojih otkriva točne odgovore. Pitanja se kreću od lakših prema težima, a nakon svakog pitanja sudionici koji su ponudili krivi odgovor otpadaju. Sudionici koji točno odgovore na posljednje pitanje su pobjednici te dijele određeni novčani fond nagrada na jednake dijelove.



Slika 2.3. *Korisničko sučelje aplikacije HQ Trivia*

Mnoge aplikacije proizašle su iz istoimenih kvizova koji su se prvotno emitirali na televiziji, a to su Tko želi biti milijunaš, Jeopardy, Trivial Pursuit itd.

Konkretno, ovaj rad se neće baviti samim kviz aplikacijama nego onima koje služe za njihovo stvaranje i vođenje. Neke od gore nabrojanih aplikacija imaju i mogućnost kreiranja kvizova iako im je primarna funkcionalnost igranje već postojećih. Sporcle [6] je web stranica razvijena od strane trivija entuzijasta Matt Ramme-a koji ju je predstavio davne 2007. godine i koja danas broji preko 5 milijardi odigranih kvizova. Sporcle ime duguje leksičkoj mješavini riječi *sports* i *oracle*. Nudi stvaranje kvizova ili igranje drugih kvizova kreiranih od strane aktivne zajednice pomoću svog vrlo intuitivnog korisničkog sučelja (slika 2.4.).

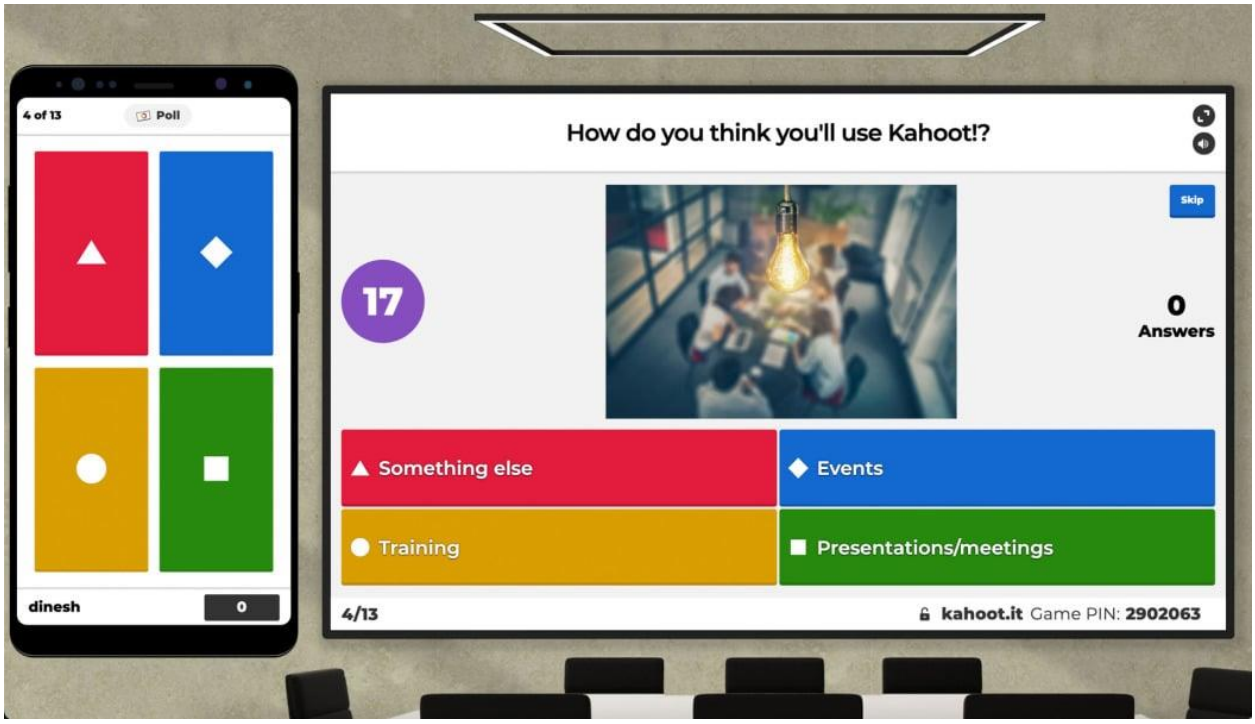


Slika 2.4. *Korisničko sučelje web stranice Sporcle*

Od doba COVID-a populariziraju se individualni *online* kvizovi i kviz lige. Neke od popularnijih su kviZDing [7] koji je proizvod Kvizaške udruge Zadar, a konkretnije Roka Svena Suraća koji piše i organizira sva pitanja te KFL (*Knowledge*. *Facts*. *Ladder*.) ukrajinskog autora Vadyma Bondara [8]. Te dvije kvizaške lige, a vjerojatno i mnoge druge, koriste *Google Forms* kao oblik kreiranja i popunjavanja pitanja. Google Forms je besplatna programska podrška za administraciju anketa razvijena od strane Google-a.

Kahoot! [9] je norveška *online* platforma za stvaranje interaktivnih kvizova s vremenskim ograničenjem, što potiče natjecateljski duh među sudionicima. Aplikacija je primarno web bazirana, u smislu prikazivanja pitanja, no odgovaranju na pitanja se može pristupiti i putem

aplikacije (slika 2.5.). Kahoot! je stekao veliku popularnost zahvaljujući svom živopisnom sučelju i gamifikacijskom pristupu.



Slika 2.5. *Korisničko sučelje aplikacije Kahoot!*

Aplikacija za samo vođenje kvizova nema puno, a ako ih ima, vjerojatno se koriste u privatne svrhe u smislu olakšavanja vođenja lokalnih kvizova te za njih ne postoji jednako veliko tržište kao za ono za igranje istih. Većina postojećih rješenja su web aplikacije, što može predstavljati izazove u pogledu latencije i pouzdanosti u određenim scenarijima korištenja. *Desktop* aplikacija koja bi kombinirala prednosti lokalnog izvođenja s mogućnošću sinkronizacije podataka u oblaku mogla bi ponuditi optimalno korisničko iskustvo. Uzimajući u obzir trenutno stanje na tržištu, jasno je da postoji prostor za inovaciju u području aplikacija za kvizove. Razvoj nove aplikacije koja bi adresirala navedene nedostatke i integrirala nove funkcionalnosti mogao bi značajno unaprijediti iskustvo stvaranja i vođenja kvizova, kako za organizatore, tako i za sudionike.

3. TEORIJSKE OSNOVE I KORIŠTENE TEHNOLOGIJE

Razvoj moderne aplikacije za stvaranje i vođenje kvizova zahtijeva kombinaciju različitih tehnologija koje omogućuju stvaranje interaktivnog, responzivnog i skalabilnog rješenja. U ovom poglavlju detaljno ću opisati ključne tehnologije korištene u razvoju aplikacije. Tehnologije korištene u izradi ove aplikacije su JavaScript biblioteka React, razvojni okvir (engl. *Framework)* Electron, platforma za razvoj mobilnih i web aplikacija Firebase te biblioteka Material-UI (odnosno MUI) koja je korištena za dizajn aplikacije.

3.1. React

React je moderna JavaScript biblioteka za izgradnju korisničkih sučelja, razvijena od strane Facebook-a. Njena popularnost proizlazi iz njene učinkovitosti, fleksibilnosti i snažnog ekosustava. Prema [10] React je 2019. godine preuzeo titulu najpopularnije biblioteke u anketi provedenoj na preko 90 tisuća programera. React se temelji na konceptu komponenti, što omogućuje razvoj skalabilnih i održivih aplikacija. Srž React-a čine komponente - samostalne cjeline koje enkapsuliraju dio korisničkog sučelja i njegovu logiku. Komponente mogu biti funkcijske ili klasne. Svaka komponenta može imati svoje vlastito stanje (engl. *State*) koje predstavlja podatke specifične za tu komponentu. U novijim verzijama React-a preferiraju se funkcijske komponente zbog svoje jednostavnosti i bolje integracije s *Hooks* aplikacijsko programskim sučeljem (engl. *Application programming interface - API*). *Hooks* su značajka koja omogućuje korištenje stanja i drugih React značajki u funkcijskim komponentama. Još jedno od korisnih prednosti React-a je Context API koji je React-ov mehanizam za učinkovito prosljeđivanje podataka kroz stablo komponenti bez potrebe za eksplicitnim prosljeđivanjem svojstava (engl. *Propreties*, ili skraćeno *props*) na svakoj razini. *Props* su način prosljeđivanja podataka od roditeljske komponente do dječje (engl. *Child*) komponente. Možemo ih zamisliti kao argumente funkcije ili atribute HTML elementa. Svojstva *propsa* su omogućavanje jednosmjernog toka podataka te nepromjenjivost.

3.2. Electron

Electron [11] (ranije poznat i kao Atom Shell) je besplatni razvojni okvir otvorenog koda originalno razvijen od strane *GitHub*-a, dok ga trenutno održavaju OpenJS Foundation i aktivna

zajednica suradnika. Electron kombinira okuženje za izvršavanje JavaScript-a zvano Node.js s web preglednikom Chromium što programerima omogućuje razvijanje višeplatformskih *desktop* aplikacija. Korištenje Electron.js-a omogućuje nam da iskoristimo prednosti web tehnologija za razvoj, istovremeno pružajući korisnicima iskustvo izvorne (engl. *native*) *desktop* aplikacije. Electron aplikacije se temelje na arhitekturi s dva glavna tipa procesa: glavni (engl. *Main*) proces i proces renderiranja (engl. *Renderer*). Prema [12], najveći nedostatci Electron-a su velika potrošnja memorije, veća veličina aplikacija, ograničen pristup resursima sustava te razni sigurnosni izazovi. Neke od popularnih aplikacija koje su razvijene pomoću Electrona su Discord, Figma, Signal, Skype, Slack i Tidal.

3.3. Firebase

Firebase je platforma za razvoj mobilnih i web aplikacija koju je razvio i održava Google. Pruža širok spektar alata i usluga koje pojednostavljuju proces razvoja, omogućujući programerima da se fokusiraju na stvaranje korisničkog iskustva bez potrebe za upravljanjem kompleksnom pozadinskom (engl. *Backend*) infrastrukturom. Firebase pruža pakete za razvoj programa (engl. *Software development kit* - SDK) koji se lagano koriste, pozadinske servise i gotove biblioteke korisničkog sučelja (engl. *User interface* - UI) za autentifikaciju korisnika. Firebase također omogućava korištenje Firestore-a [13] - fleksibilne skalabilne NoSQL baze podataka u oblaku za pohranu i sinkronizaciju podataka za klijentski i serverski razvoj.

3.4. Material-UI

Material-UI (kraće - MUI) je biblioteka komponenta koja implementira Google-ov sistem materijalnog dizajna [14]. MUI pruža sveobuhvatan set pred-izgrađenih komponenti koje omogućuju brz razvoj konzistentnog i estetski ugodnog korisničkog sučelja. MUI omogućava korištenje lako prilagodljivih komponenti poput gumbi, tekstualnih polja, alatnih traka i sl. Također, osigurava lako prilagođavanje izgleda aplikacije kroz sustav tema, responzivnost kroz komponente koje su inherentno responzivne, korištenje ikona za dodavanje vizualnih indikatora i poboljšanje korisničkog iskustva, makeStyles *hook*-a za stvaranje dodatno prilagođenih stilova te integraciju s React-om.

4. OSTVARENO PROGRAMSKO RJEŠENJE

Razvoj aplikacije za stvaranje i vođenje kvizova predstavlja složen proces koji obuhvaća integraciju različitih tehnologija i implementaciju brojnih funkcionalnosti. Ova aplikacija pruža sveobuhvatno rješenje za organizaciju i provođenje interaktivnih kvizova.

Arhitektura aplikacije temelji se na modularnom pristupu, gdje su različite funkcionalnosti enkapsulirane u zasebne React komponente. Ovo ne samo da olakšava razvoj i održavanje, već i omogućuje fleksibilnost u budućim nadogradnjama. Electron okvir omogućuje nam da ovu web aplikaciju pretvorimo u desktop aplikaciju, pružajući nativno iskustvo na različitim operacijskim sustavima.

Glavne komponente aplikacije uključuju početni zaslon, prilagodbu postavki kviza, listu ekipa, ekran vođenja aplikacije, rezultate i povijest igara. Svaka od ovih komponenti ima specifičnu ulogu u životnom ciklusu kviza, od početnog postavljanja do završne analize rezultata.

Implementacija ključnih funkcionalnosti poput stvaranja i uređivanja pitanja, vođenja kviza u stvarnom vremenu te bodovanja i rangiranja timova oslanja se na React *hooks* za upravljanje stanjem i efektima. Ovo omogućuje reaktivno ažuriranje korisničkog sučelja u skladu s promjenama u podacima.

Posebna pažnja posvećena je korisničkom iskustvu kroz implementaciju višejezičnosti i mogućnosti promjene teme. Korištenjem React Context API-ja, omogućeno je jednostavno prebacivanje između hrvatskog i engleskog jezika, kao i između svijetle i tamne teme, što aplikaciju čini pristupačnom širem krugu korisnika.

Integracija s Firebase-om predstavlja ključni aspekt *backend* funkcionalnosti aplikacije. Firebase Authentication koristi se za sigurno upravljanje korisničkim računima, dok Firestore baza podataka omogućuje pohranu i sinkronizaciju podataka o kvizovima, pitanjima i rezultatima u realnom vremenu.

Prilagodba aplikacije za *desktop* okruženje uključuje implementaciju prilagođene naslovne trake i upravljanje prozorima aplikacije. Ovo je postignuto korištenjem Electron API-ja, čime se postiže dojam *native desktop* aplikacije, istovremeno zadržavajući prednosti web tehnologija.

Kroz ovaj razvoj, neki od izazova bili su osiguravanje konzistentnog korisničkog iskustva na različitim platformama, optimizacija performansi pri radu s velikim brojem pitanja i timova.

U sljedećim potpoglavljima biti će detaljnije obrađen svaki od ovih aspekata razvoja, pružajući uvid u specifične tehnike i pristupe korištene u izgradnji aplikacije za stvaranje i vođenje kvizova.

4.1. Arhitektura aplikacije

Na najvišoj razini, aplikacija se sastoji od sljedećih ključnih komponenti:

1. Sučelje (engl. *Frontend*): korisničko sučelje aplikacije razvijeno je korištenjem React.js-a, organizirano u niz funkcionalnih komponenti. Glavne komponente uključuju:
    - App.js: korijenska komponenta koja upravlja usmjeravanjem (engl. *Routing*) i globalnim stanjem
    - StartScreen.js: početni zaslon s opcijama za pokretanje kviza, pregled povijesti i upravljanje pitanjima
    - QuizCustomization.js: sučelje za prilagodbu postavki kviza
    - QuizScreen.js: glavno sučelje za vođenje kviza
    - Scoreboard.js: prikaz rezultata timova
    - TeamList.js: upravljanje timovima koji sudjeluju u kvizu
    - GameHistory.js: pregled povijesti odigranih kvizova
2. Integracija radne površine: Electron.js omogućuje pakiranje React aplikacije u *desktop* aplikaciju. Ključne komponente uključuju:
    - main.js: glavni proces Electron aplikacije koji upravlja životnim ciklusom aplikacije i stvaranjem prozora
    - preload.js: skripta koja sigurno izlaže određene Electron API-je React aplikaciji
    - TitleBar.js: prilagođena naslovna traka za desktop aplikaciju
3. Pozadinske usluge: Firebase pruža pozadinsku infrastrukturu za aplikaciju, uključujući:
    - Firebase *Authentication*: upravljanje korisničkim računima i autentifikaciju
    - Firestore: NoSQL baza podataka za pohranu podataka o kvizovima, pitanjima i rezultatima
    - *Security Rules*: osiguravanje pristupa podacima samo autoriziranim korisnicima
4. Upravljanje stanjem aplikacije: postignuto kombinacijom lokalnog stanja komponenti (korištenjem useState *hook*-a) i globalnog stanja (korištenjem Context API-ja). Ključni konteksti uključuju:

    - *LanguageContext*: upravljanje višejezičnošću
    - *ThemeContext*: upravljanje temom aplikacije (svjetla/tamna)
5. Usmjeravanje: React Router koristi se za upravljanje navigacijom unutar aplikacije, omogućujući prijelaz između različitih zaslona bez potrebe za osvježavanjem stranice.
6. Stiliziranje: Material-UI koristi se za konzistentan dizajn komponenti, uz dodatne prilagodbe korištenjem makeStyles *hook*-a za stvaranje prilagođenih stilova.

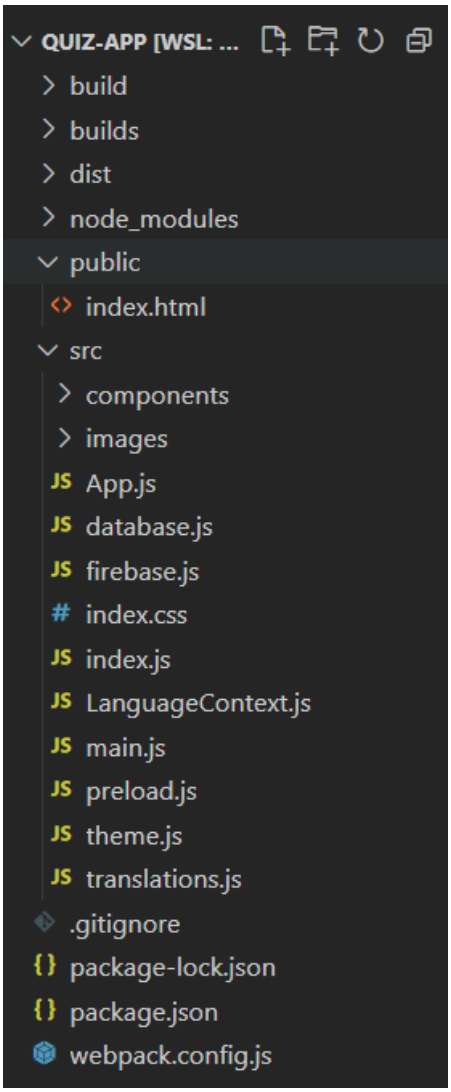
Ova arhitektura omogućuje jasnu separaciju odgovornosti, gdje je svaka komponenta zadužena za specifičan aspekt funkcionalnosti aplikacije. Modularni pristup olakšava održavanje i nadogradnju aplikacije, dok integracija s Electron.js-om i Firebase-om pruža robusnu osnovu za *desktop* funkcionalnosti i *backend* usluge.

Važan aspekt ove arhitekture je i njena prilagodljivost različitim scenarijima korištenja. Aplikacija može raditi *online*, koristeći Firebase za sinkronizaciju podataka u stvarnom vremenu, ali i *offline*, zahvaljujući Electron.js-u i lokalnom predmemoriranju podataka. Ovo osigurava da se kvizovi mogu provoditi čak i u situacijama s ograničenom internetskom vezom.

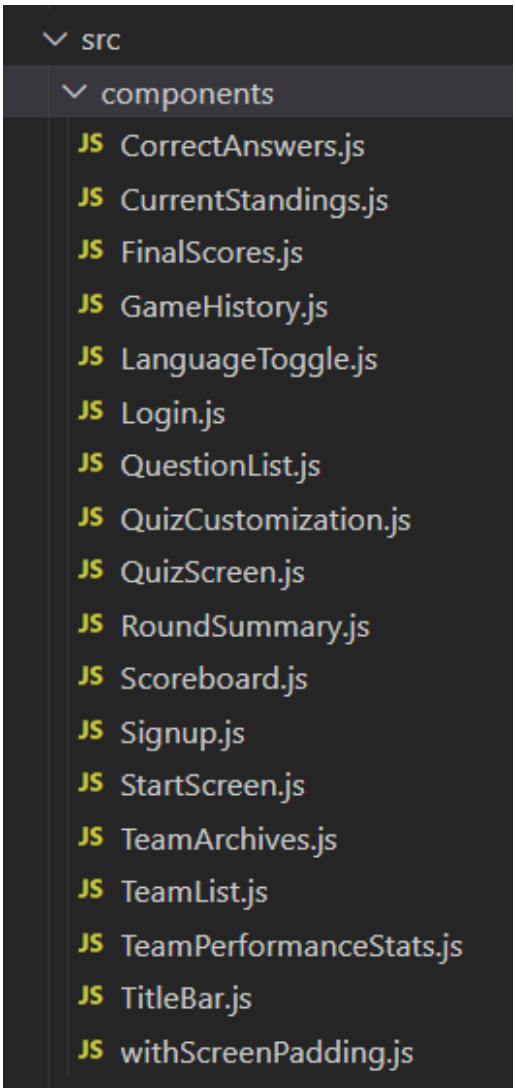
Konačno, ova arhitektura omogućuje skalabilnost aplikacije. Kako se baza korisnika i složenost kvizova povećava, arhitektura podržava dodavanje novih funkcionalnosti bez značajnih promjena u postojećoj strukturi koda.

4.2. Glavne komponente aplikacije

Prema strukturi aplikacije na slici 4.1., vidljivo je da se ona sastoji od velikog broja datoteka. Aplikacija se sastoji od nekoliko ključnih komponenti (sl. 4.2.), svaka sa svojom specifičnom ulogom u funkcioniranju cjelokupnog sustava. Ove komponente su implementirane koristeći funkcijske React komponente, što omogućuje učinkovito upravljanje stanjem i životnim ciklusom komponenti.



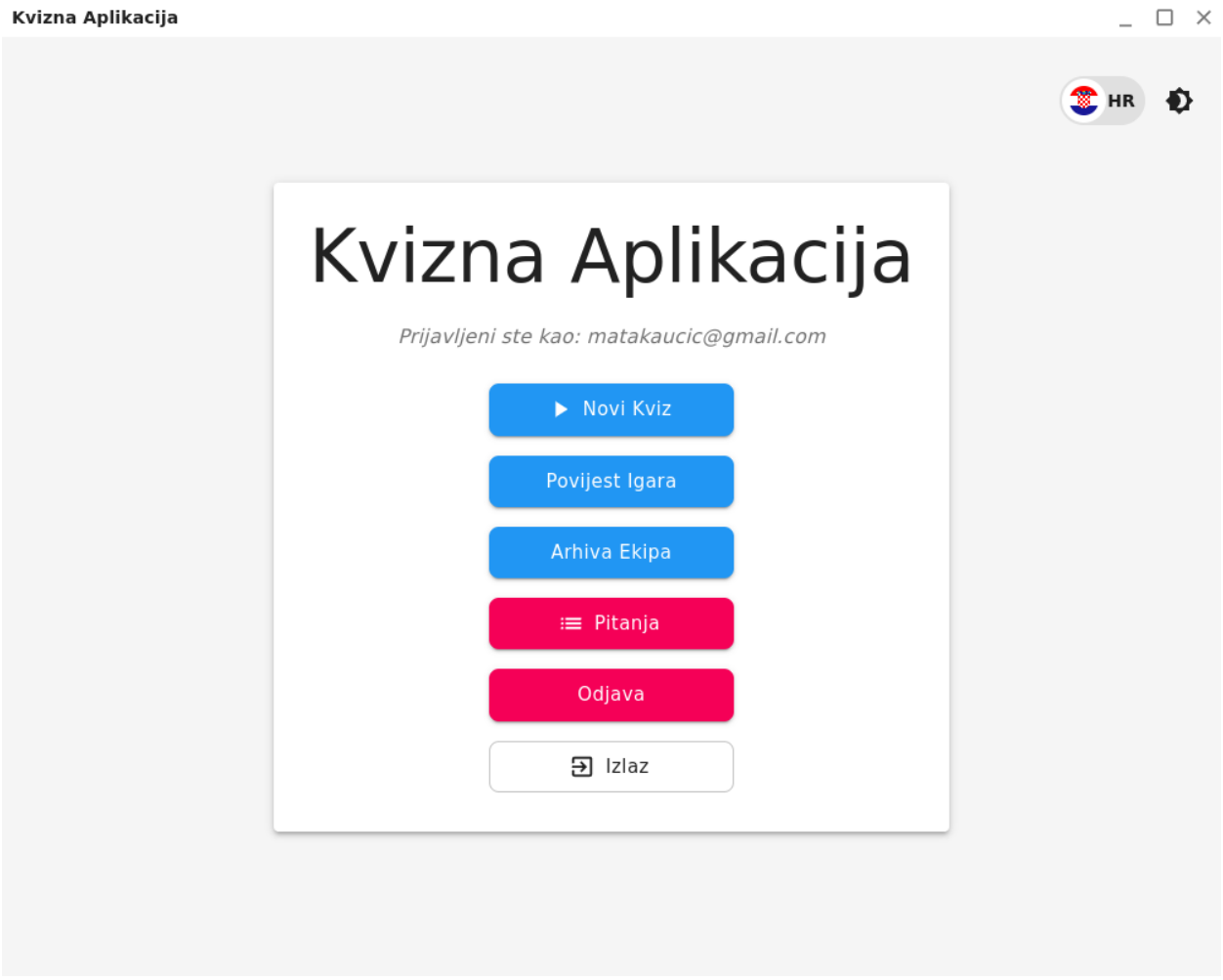
Slika 4.1. *Struktura aplikacije*



Slika 4.2. *Komponente aplikacije*

4.2.1. Početni zaslon

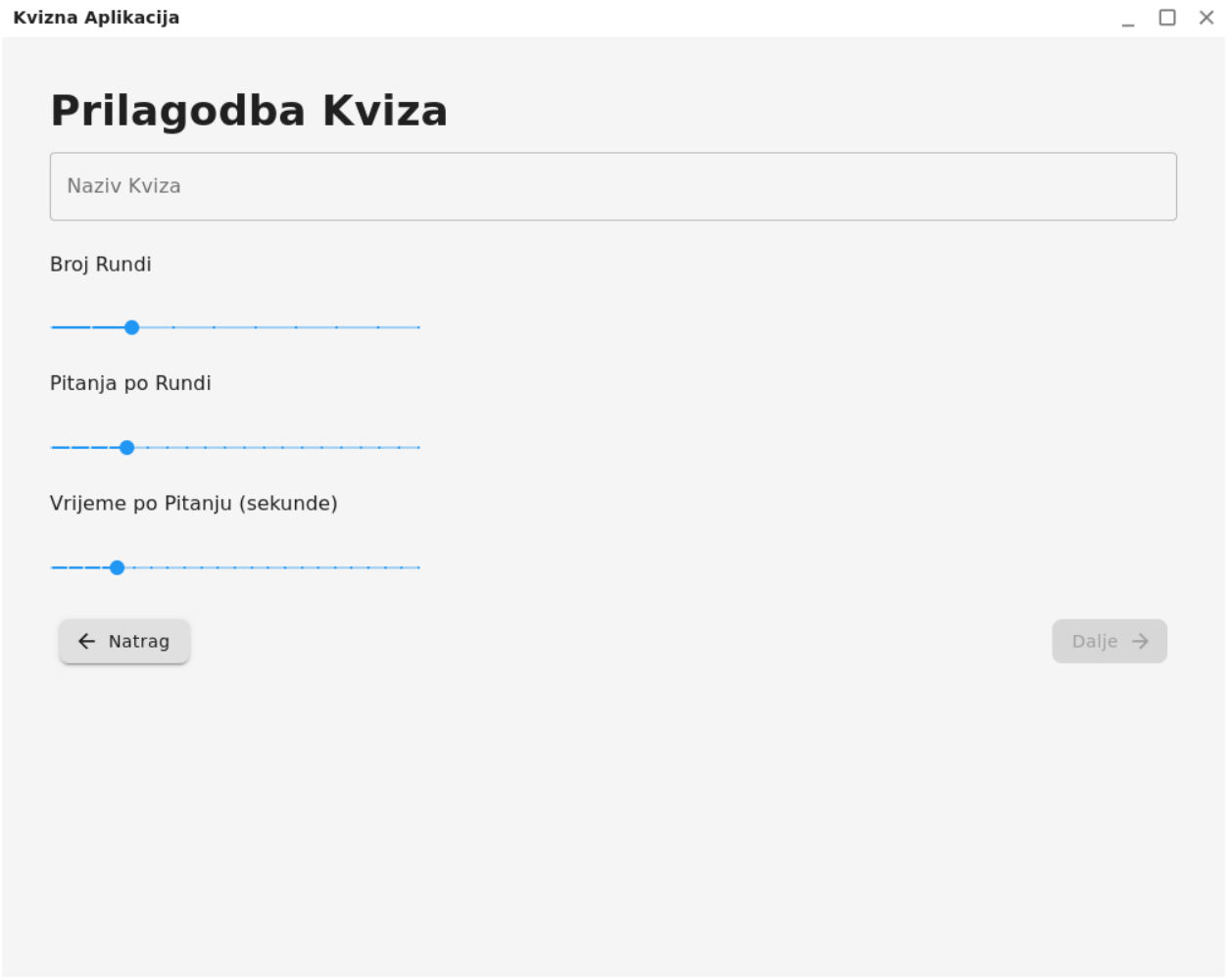
StartScreen komponenta predstavlja početni zaslon aplikacije. Dizajnirana je kao centralno mjesto iz kojeg korisnik može pristupiti svim glavnim funkcionalnostima aplikacije. Ova komponenta prikazuje gumbe za pokretanje novog kviza, pregled povijesti igara, pristup arhivi timova, upravljanje pitanjima i odjavu. Također, implementiran je sustav za promjenu jezika i teme aplikacije, koristeći *LanguageToggle* komponentu i Material-UI *ThemeProvider*. Početni zaslon aplikacije vidljiv je na slici 4.3.



Slika 4.3. *Početni ekran aplikacije*

4.2.2. Prilagodba postavki kviza

U *QuizCustomization* komponenti, implementirano je sučelje za prilagodbu postavki kviza. Ova komponenta omogućuje korisniku da definira broj rundi, broj pitanja po rundi i vrijeme po pitanju, kao što je prikazano na slici 4.4. Korištene su Material-UI *Slider* komponente za intuitivno podešavanje ovih parametara.



Slika 4.4. *Prilagodba postavki kviza*

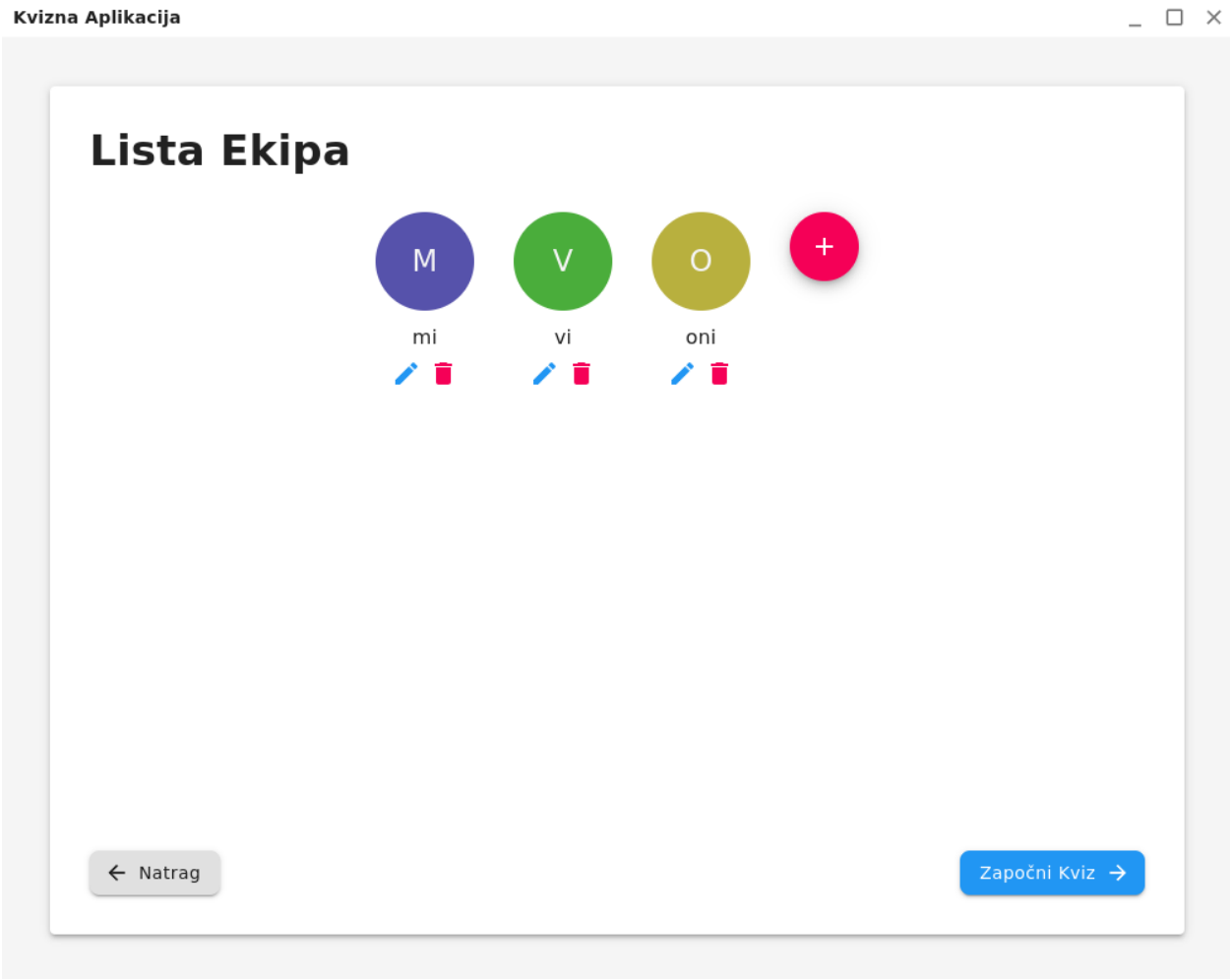
Komponenta također provjerava dostupnost dovoljnog broja pitanja za odabrane postavke, koristeći Firebase Firestore za dohvaćanje ukupnog broja pitanja u bazi (kôd 4.1.). Provjerava dostupnost tako što pomnoži broj odabranih rundi s brojem pitanja po runda te zatim taj broj uspoređuje s brojem pitanja u bazi podataka.

```
const checkQuestionAvailability = (roundsCount, questionsCount) => {
    const totalNeededQuestions = roundsCount * questionsCount;
    if (totalNeededQuestions > totalQuestions) {
      setError(
        `Not enough questions available. You need ${totalNeededQuestions}
questions, but only ${totalQuestions} are available.`
      );
      return false;
    }
    setError("");
    return true;
  };
```

Kôd 4.1. *Funkcija za provjeru dostupnosti dovoljnog broja pitanja*

4.2.3. Lista ekipa

TeamList komponenta služi za upravljanje ekipama koje sudjeluju u kvizu. Implementirana je funkcionalnosti za dodavanje, uređivanje i brisanje ekipa. Svaka ekipa je predstavljena Material-UI *Avatar* komponentom s prvim slovom imena ekipe (slika 4.5.). Komponenta također osigurava da postoji minimalan broj timova (dva) prije nego što kviz može započeti.

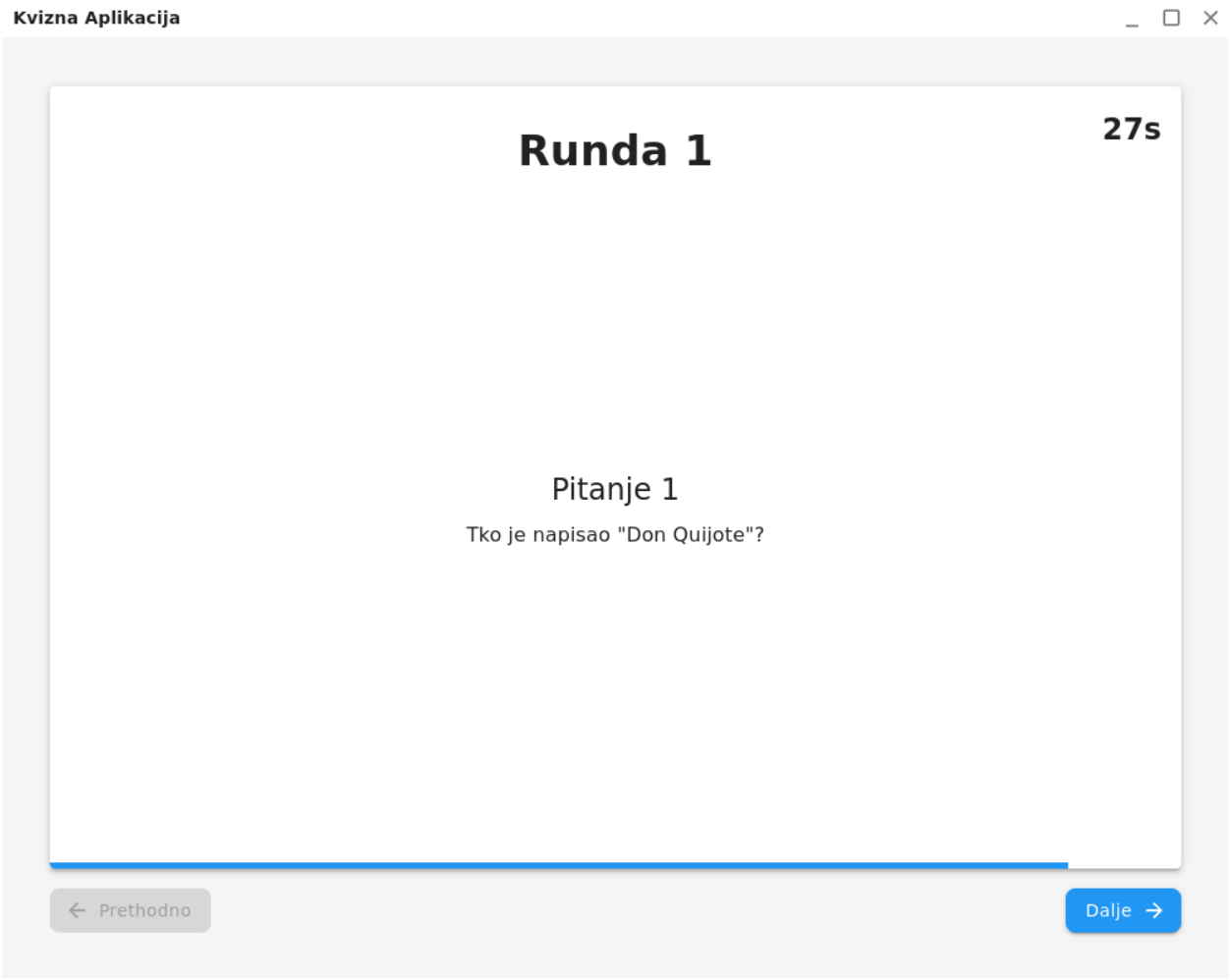


Slika 4.5. *Upravljanje ekipama*

4.2.4. Vođenje kviza

QuizScreen je središnja komponenta za vođenje kviza. Implementiran je prikaz trenutnog pitanja, odbrojavanje vremena i navigaciju između pitanja. Korišten je useEffect *hook* za upravljanje odbrojavanjem i automatskim prelaskom na sljedeće pitanje. Također, sastoji se od plave trake napretka koja grafički prikazuje koliko je još vremena ostalo za pitanje, a smanjuje se, odnosno

prazni, svake sekunde (slika 4.6.). Vraćanje na prethodna i sljedeća pitanja je moguće, a postiže se, ili pritiskom na gumbe na ekranu, ili pomoću tipkovnice.

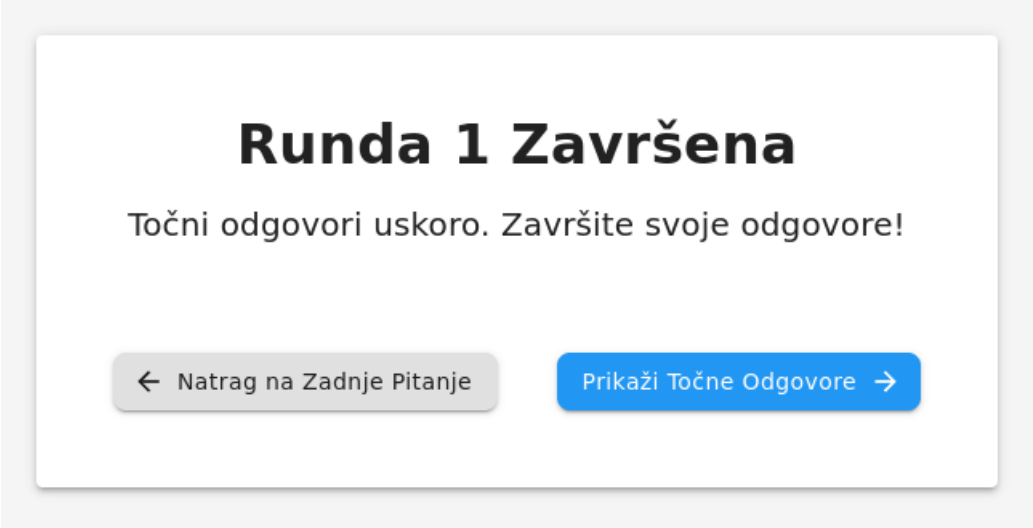


Slika 4.6. *Ekran provođenja kviza*

Komponenta također upravlja prijelazom između rundi i prikazom točnih odgovora na kraju svake runde. Funkcija *handleNextQuestion* (kôd 4.2) se bavi upravo time - po kraju isteka vremena za pitanje, ono prebacuje na iduće pitanje (ukoliko ono nije zadnje u trenutnom krugu) ili prebacuje na ekran sumiranja kruga (slika 4.7.) koji služi kao svojevrsna zadnja prilika igračima da dovrše svoje odgovore i/ili pogledaju još jednom pitanja koja nisu stigli riješiti.

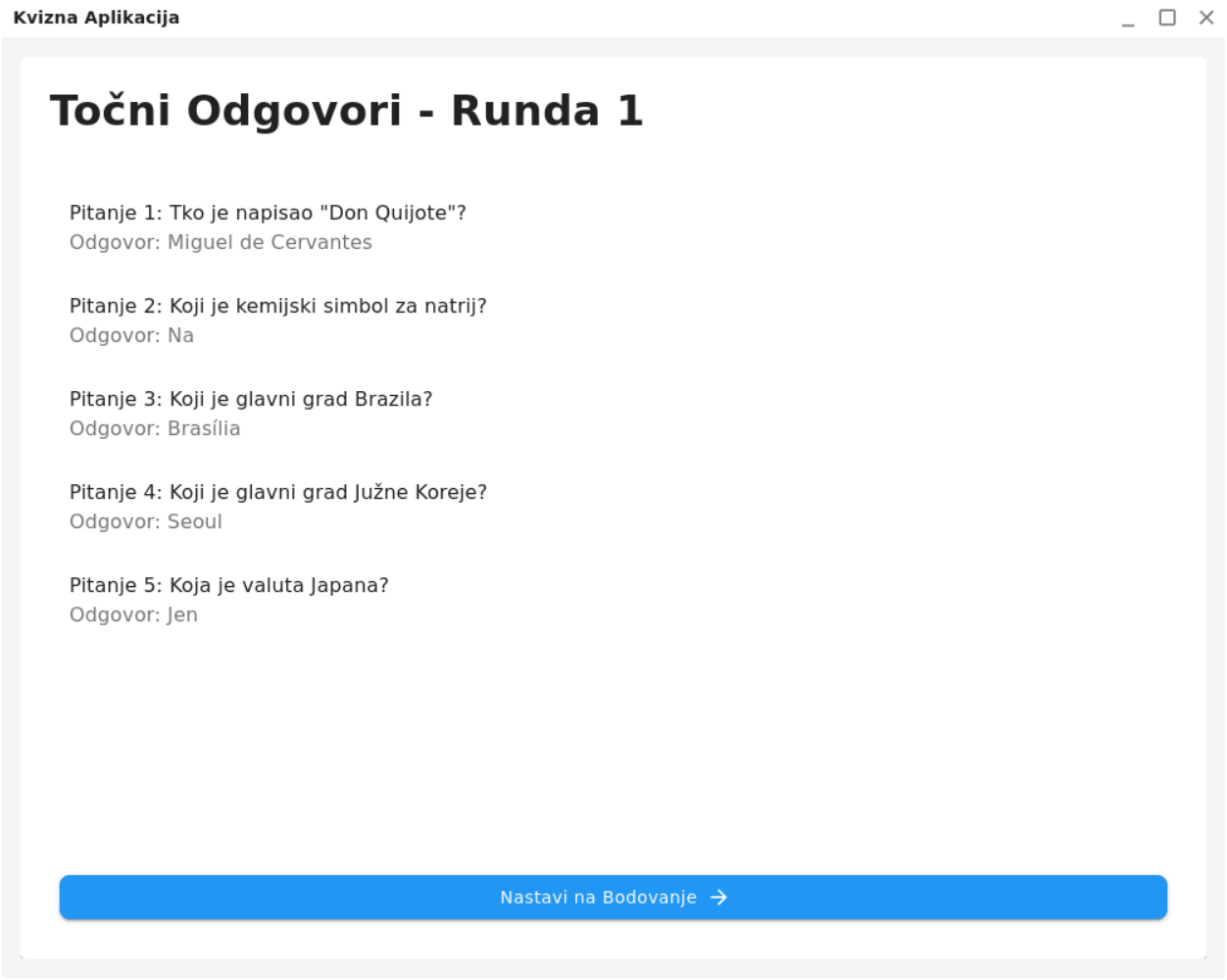
```
const handleNextQuestion = useCallback(() => {
    if (currentQuestion + 1 < currentRoundQuestions.length) {
      setCurrentQuestion(currentQuestion + 1);
      setTimeLeft(quizSettings.timePerQuestion);
    } else {
      setShowRoundSummary(true);
    }
  }, [currentQuestion, quizSettings, currentRoundQuestions.length]);
```

Kôd 4.2. *Funkcija baratanja idućim pitanjem*



Slika 4.7. *Sumiranje runde*

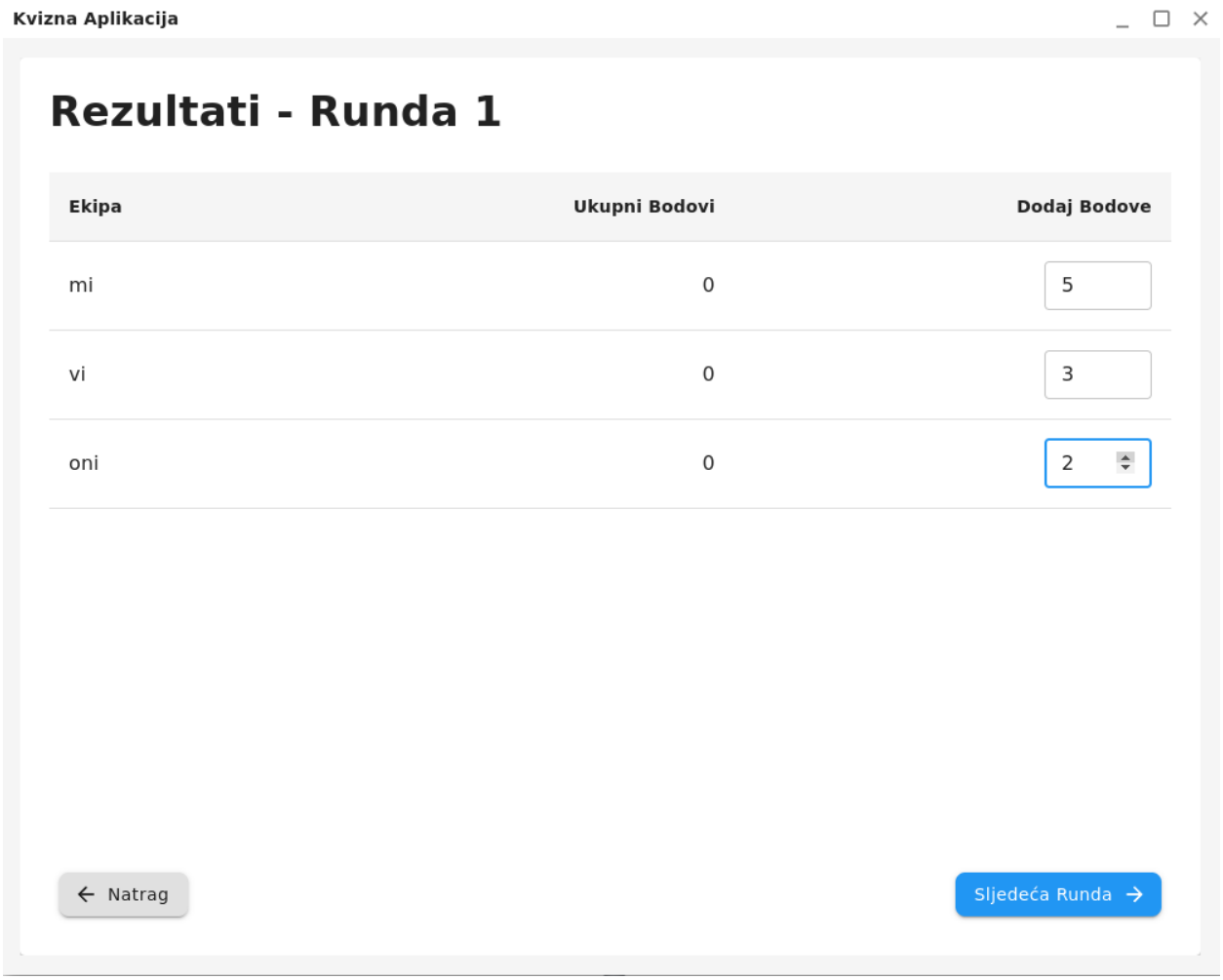
Nakon ekrana sumiranja kruga, postoji opcija prikazivanja točnih odgovora. Ona radi tako što u aplikaciji postoji varijabla stanja *usedQuestionIds* koja sadrži niz ID-eva pitanja koji su bili korišteni u prethodnim rundama te se na kraju runde dodaju ID-evi pitanja koji su bili korišteni te runde te pomoću tih ID-eva može prikazati točne odgovore na pitanja iz prijašnjeg kruga (slika 4.8.). Ono također osigurava praćenje svih korištenih pitanja te sprječava potencijalno ponavljanje istih pitanja kroz isti kviz. *UsedQuestionIds* se briše na kraju svakog kviza kako bi se on mogao igrati više puta. Komponente međusobno komuniciraju putem *prop*-ova i *callback* funkcija, dok se za globalno stanje koristi React Context API. Sama logika oko dohvaćanja pitanja biti će pomnije objašnjena u poglavlju 4.3.2.



Slika 4.8. *Točni odgovori*

4.2.5. Rezultati ekipa

U *Scoreboard* komponenti (slika 4.9.), implementirano je sučelje za unos i prikaz rezultata ekipa. Komponenta prikazuje trenutne rezultate svih ekipa i omogućuje unos bodova za trenutnu rundu. Nakon unosa bodova, komponenta automatski ažurira ukupne rezultate i rangira timove. Korištena je Material-UI *Table* komponenta za pregledan prikaz rezultata.



Slika 4.9. *Unos rezultata za prethodnu rundu*

Unos bodova je limitiran na pozitivne brojeve koji su manji ili jednaki broju pitanja koji su se nalazili u tom krugu (kôd 4.3.).

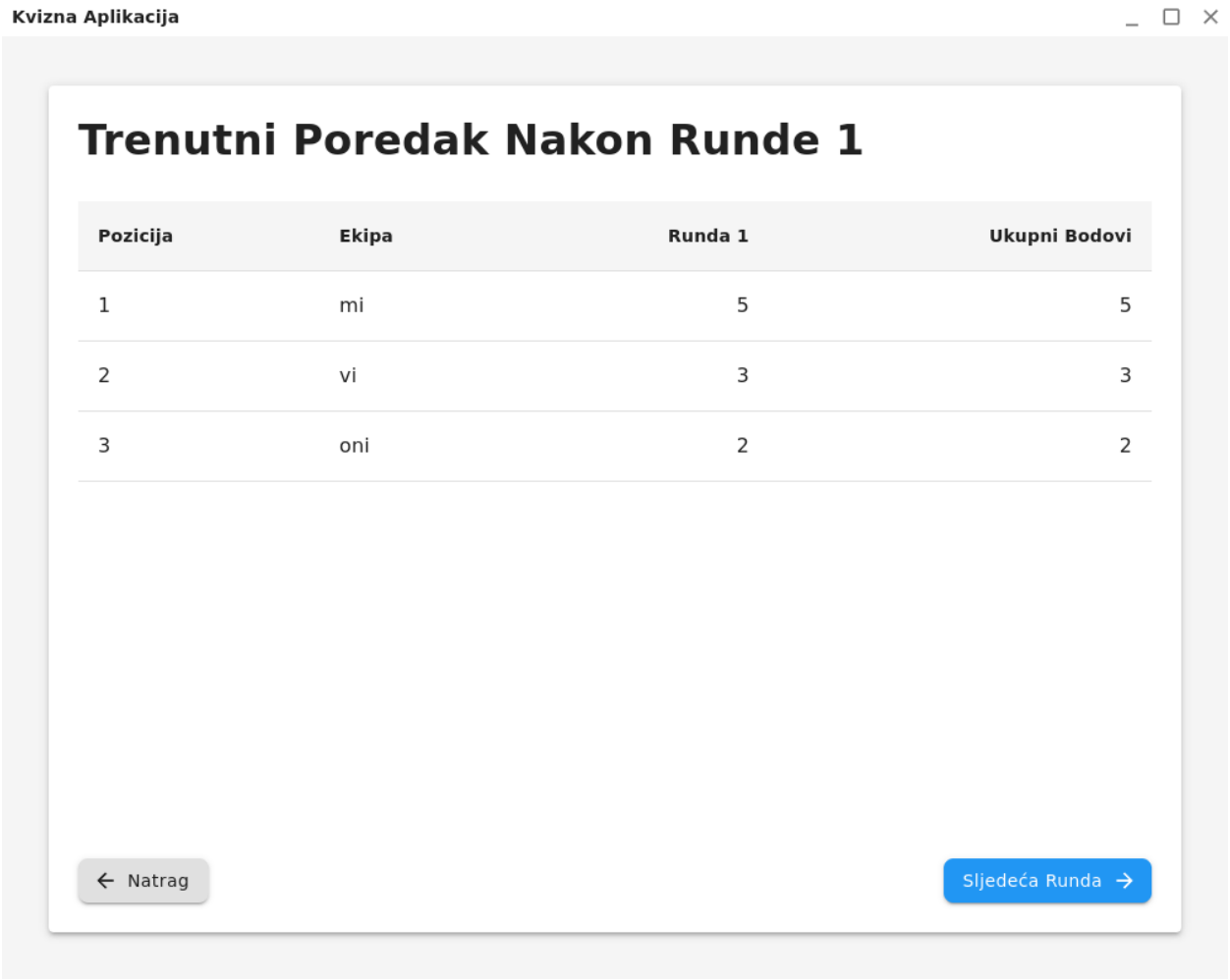
```
const handleScoreChange = (teamName, score) => {
    const numScore = Number(score);
    if (isNaN(numScore)) return;

    const maxScore = quizSettings.questionsPerRound;
    const validScore = Math.max(0, Math.min(numScore, maxScore));

    setNewScores((prev) => ({
      ...prev,
      [teamName]: validScore,
    }));
  };
```

Kôd 4.3. *Funkcija za unos rezultata*

Nakon unosa bodova za svaku ekipu (ako nije unesena nikakva brojka, ekipa dobiva 0 bodova), pojavljuje se ekran trenutnog poretka (slika 4.10.) u kojemu su ekipe sortirane po dosad ostvarenom broju bodova kroz kviz.

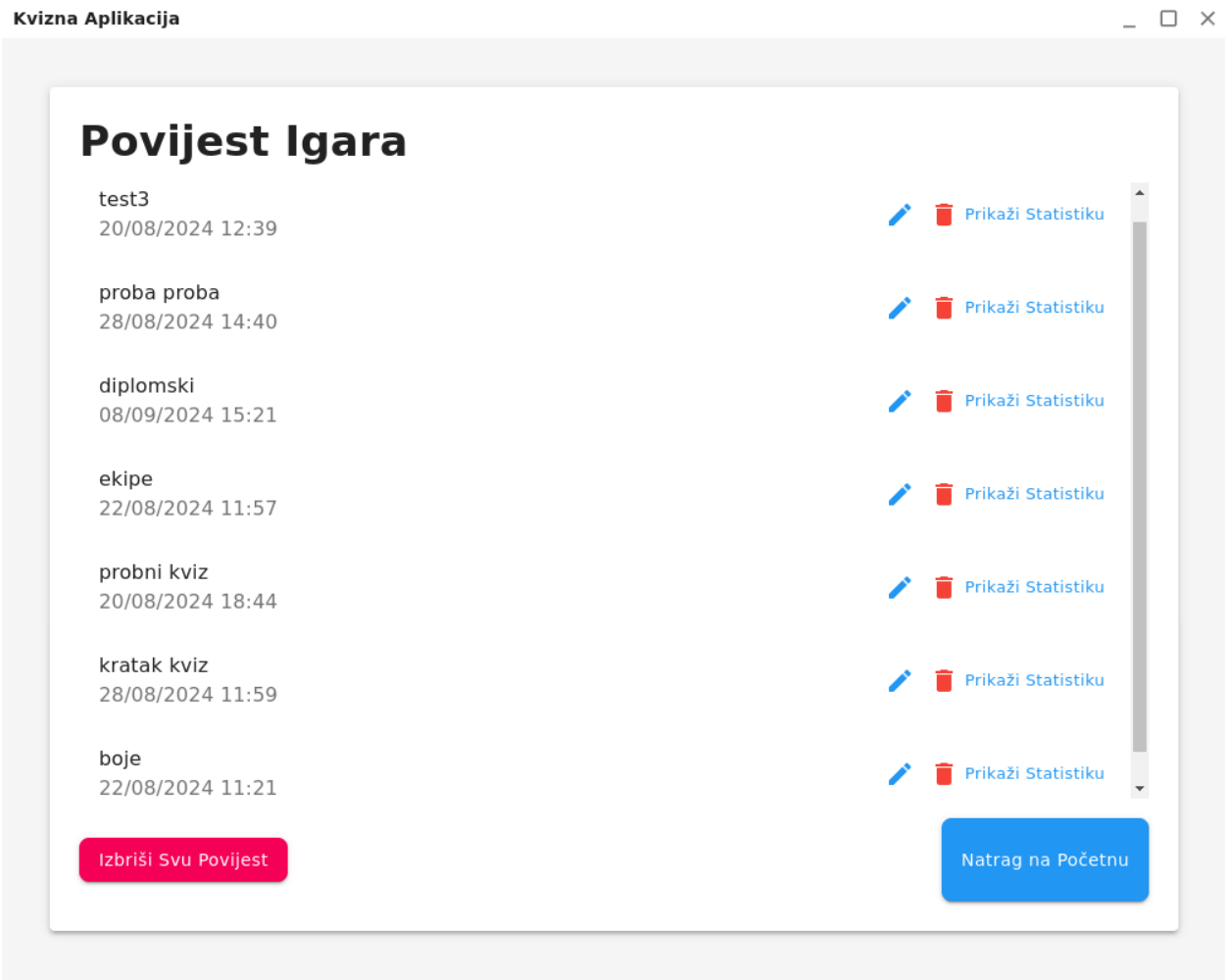


Slika 4.10. *Trenutni poredak*

U slučaju da je trenutno odigrana zadnja runda, umjesto opcije prelaska na sljedeću rundu ponuditi će se opcija završetka kviza i prikaza konačnog poretka. Pri ulasku u ekran konačnog poretka, kviz će se automatski spremiti u povijest te mu se kasnije može pristupiti preko početnog ekrana.

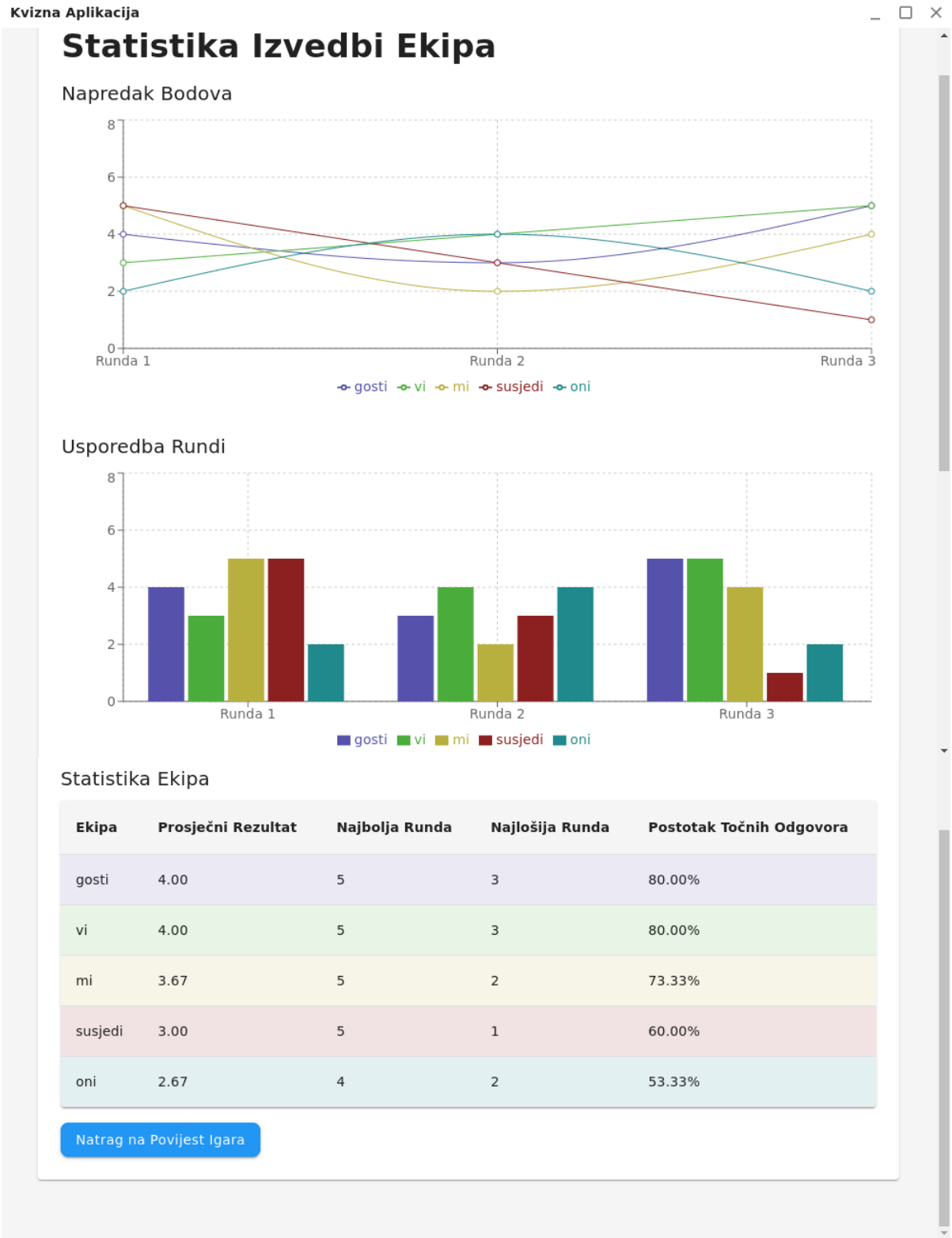
4.2.6. Povijest igara

GameHistory je komponenta koja prikazuje povijest odigranih kvizova (slika 4.11.). Implementiran je prikaz svih prethodnih igara s detaljima poput datuma, vremena i imena kviza. Komponenta omogućuje brisanje pojedinačnih zapisa ili cijele povijesti. Također, moguće je uređivanje pojedinih igara - naziva, datuma i sl.



Slika 4.11. *Povijest svih kvizova*

Za svaku igru, dodana je opcija za pregled detaljnih statistika koje se prikazuju u zasebnoj komponenti *TeamPerformanceStats* (slika 4.12.). U njoj su dodane vizualizacije kretanja bodova po krugu, izvedbe timova u vidu bodova po krugovima, minimalni i maksimalni rezultat te postotak točnih odgovora svake ekipe.



4.12. *Vizualizacija bodova ekipa u kvizu*

4.3. Implementacija ključnih funkcionalnosti

U procesu razvoja aplikacije za stvaranje i vođenje kvizova, implementirano je nekoliko ključnih funkcionalnosti koje čine srž aplikacije. Ove funkcionalnosti su pažljivo dizajnirane i implementirane kako bi se osiguralo optimalno korisničko iskustvo i učinkovito upravljanje kvizovima.

Dodavanje i uređivanje pitanja implementirano je kao temeljna funkcionalnost aplikacije. Korisnicima je omogućeno dodavanje novih pitanja, uređivanje postojećih i organiziranje pitanja u kategorije. Ova funkcionalnost je integrirana s Firebase Firestore bazom podataka, čime je osigurana postojanost podataka i mogućnost sinkronizacije između različitih uređaja.

Vođenje kviza u stvarnom vremenu implementirano je kao centralna funkcionalnost aplikacije. Ova funkcionalnost obuhvaća prikazivanje pitanja, upravljanje vremenom za odgovore, i navigaciju kroz rundu kviza. Posebna pažnja posvećena je osiguravanju glatkog korisničkog iskustva, s implementiranim animacijama za prijelaze između pitanja i rundu.

Bodovanje i rangiranje timova implementirano je kao ključni element natjecateljskog aspekta kviza. Razvijen je sustav koji omogućuje brz i jednostavan unos bodova za svaki tim nakon svake rundu. Implementiran je algoritam za automatsko rangiranje timova na temelju njihovih ukupnih bodova. Nije implementirano rukovanje slučajem da dvije ili više ekipa imaju podjednak broj bodova, ali to je jedno od potencijalnih unaprjeđenja.

Višejezičnost aplikacije implementirana je korištenjem React Context API-ja. Ova funkcionalnost omogućuje korisnicima da prebacuju jezik aplikacije između hrvatskog i engleskog u stvarnom vremenu. Svi tekstovi u aplikaciji su eksternalizirani u zasebne jezične datoteke, čime je olakšano dodavanje novih jezika u budućnosti. Trenutno su sadržana samo dva jezika, ali vrlo brzo i jednostavno ih se može dodati više.

Promjena teme također je implementirana korištenjem Context API-ja i Material-UI *ThemeProvider*-a. Korisnicima je omogućeno da prebacuju između svijetle i tamne teme, pri čemu se sve komponente aplikacije automatski prilagođavaju odabranoj temi.

Offline funkcionalnost implementirana je korištenjem mogućnosti Electron-a i lokalnog predmemoriranja podataka. Ovo omogućuje korisnicima da koriste aplikaciju čak i bez internetske veze, s naknadnom sinkronizacijom podataka kada veza postane dostupna.

Analiza performansi timova implementirana je kao dodatna funkcionalnost koja pruža vrijedne uvide nakon završetka kviza. Ova funkcionalnost uključuje generiranje grafova i statistika koje prikazuju performanse timova kroz vrijeme i usporedbu između različitih kvizova.

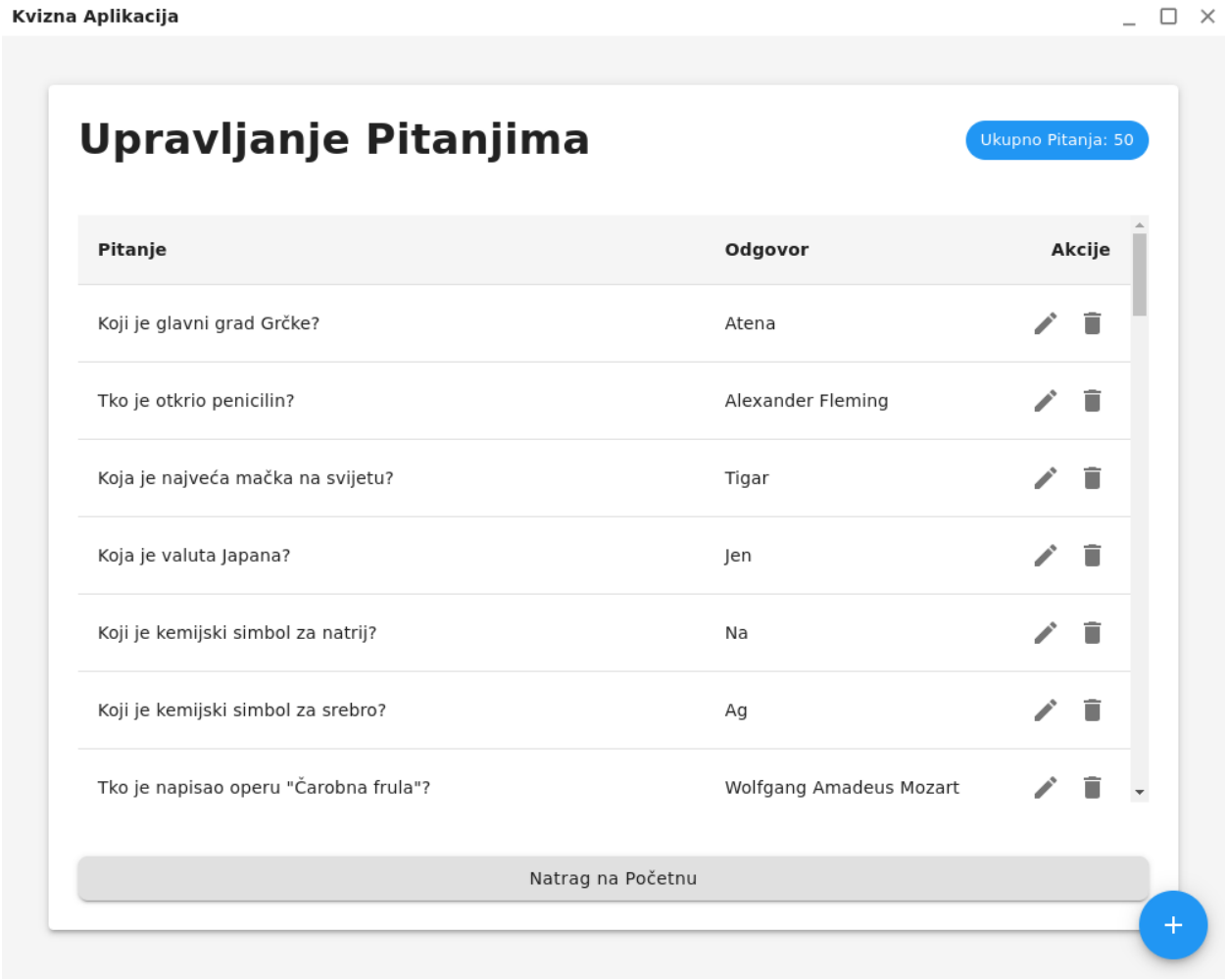
Sve ove funkcionalnosti su međusobno integrirane kako bi pružile cjelovito i koherentno korisničko iskustvo. Posebna pažnja posvećena je optimizaciji performansi, osiguravanju responzivnost sučelja i implementaciji robusnog sustava za upravljanje greškama.

4.3.1. Dodavanje i uređivanje pitanja

Dodavanje i uređivanje pitanja najvažnija je funkcionalnost ove aplikacije jer su općenito najvažniji dijelovi kviza sama pitanja. Ova funkcionalnost je dizajnirana s ciljem pružanja intuitivnog i učinkovitog sučelja za upravljanje bazom pitanja.

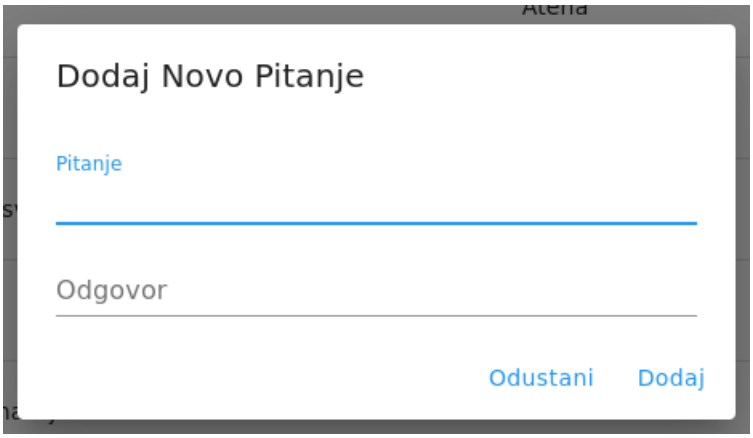
Za implementaciju ove funkcionalnosti korištena je zasebna komponenta *QuestionList*. Ova komponenta je dizajnirana kao modalno sučelje koje se može pozvati iz glavnog izbornika aplikacije. Unutar *QuestionList* komponente, implementirane su sljedeće ključne funkcije:

- Prikaz postojećih pitanja: Postojeća pitanja prikazana su u obliku liste, koristeći Material-UI *Table* komponentu. Svaki redak tablice predstavlja jedno pitanje, prikazujući tekst pitanja i pripadajući odgovor. Implementirana je i paginacija za učinkovito upravljanje velikim brojem pitanja. Uz to, dodan je brojač pitanja u gornjem desnom uglu ekrana kako bi se vrlo lako znalo koliko pitanja postoji u bazi podataka (slika 4.13.).



Slika 4.13. *Upravljanje pitanjima*

- Dodavanje novog pitanja: Za dodavanje novog pitanja, implementiran je modalni dijalog koji se otvara klikom na gumb u donjem desnom rubu ekrana. Ovaj dijalog sadrži dva *TextField* elementa - jedan za unos pitanja, drugi za unos odgovora (slika 4.14.).

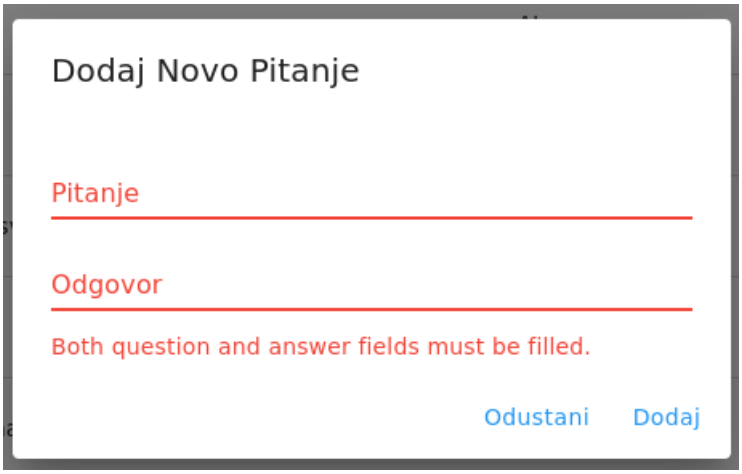


Slika 4.14. *Forma za dodavanje novog pitanja*

Implementirana je i validacija unosa kako bi se osiguralo da su oba polja popunjena prije spremanja (kôd 4.4.). Funkcija prvo provjerava je li korisnik prijavljen, zatim jesu li jedno ili oba polja pitanja (polje pitanje i polje odgovora) prazni, a zatim pomoću Firebase-ove funkcije *addDoc* dodaje pitanje u kolekciju *questions*. U slučaju da su jedno ili oba polja prazni, prikazuje se poruka greške kao što je vidljivo na slici 4.15.

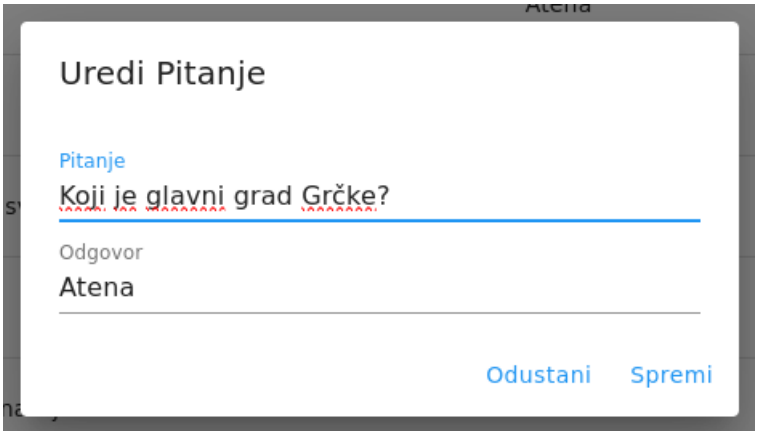
```
const handleAddQuestion = async () => {
    if (!user) {
      setError("You must be logged in to add a question");
      return;
    }
    if (!newQuestion.question.trim() || !newQuestion.answer.trim()) {
      setValidationError("Both question and answer fields must be filled.");
      return;
    }
    try {
      await addDoc(collection(firestore, "questions"), {
        ...newQuestion,
        userId: user.uid,
      });
      setOpenDialog(false);
      setNewQuestion({ question: "", answer: "" });
      setValidationError("");
      fetchQuestions();
    } catch (error) {
      console.error("Error adding question:", error);
      setError("Failed to add question. Please try again.");
    }
  };
```

Kôd 4.4. *Funkcija za dodavanje pitanja*



Slika 4.15. *Pogreška pri dodavanju pitanja*

- Uređivanje postojećeg pitanja: Uređivanje postojećeg pitanja omogućeno je klikom na ikonu olovke pored svakog pitanja u listi. Ova akcija otvara isti modalni dijalog kao i za dodavanje, ali s unaprijed popunjenim poljima postojećim podacima (slika 4.16.). Implementirana je logika koja razlikuje dodavanje novog i uređivanje postojećeg pitanja. Nakon uređivanja pitanja, podaci se mijenjaju ali ID pitanja ostaje jednak.



Slika 4.16. *Uređivanje pitanja*

Logika uređivanja pitanja je vrlo slična kod one za dodavanje pitanja, samo što ovdje već postoji ID željenog pitanja pa ga prvo prikažemo te se opet nudi opcija otvaranja dijaloga koji sprema željene promjene u bazu (kôd 4.5.).

```
const handleEditQuestion = async () => {
    if (!user) {
      setError("You must be logged in to edit a question");
      return;
    }
    if (!newQuestion.question.trim() || !newQuestion.answer.trim()) {
      setValidationError("Both question and answer fields must be filled.");
      return;
    }
    try {
      const questionRef = doc(firestore, "questions", editingQuestion.id);
      await updateDoc(questionRef, newQuestion);
      setOpenDialog(false);
      setEditingQuestion(null);
      setNewQuestion({ question: "", answer: "" });
      setValidationError("");
      fetchQuestions();
    } catch (error) {
      console.error("Error editing question:", error);
      setError("Failed to edit question. Please try again.");
    }
  };
```

Kôd 4.5. *Funkcija za uređivanje pitanja*

- Brisanje pitanja: Pored svakog pitanja implementiran je gumb za brisanje. Klikom na ovaj gumb pitanje se uklanja iz baze podataka, a samim time brojač pitanja se smanji za onaj iznos koliko smo pitanja obrisali. Brisanje pitanja funkcionira tako što se usporedi ID željenog pitanja s pitanjima u bazi i jednostavno se ukloni pitanje iz kolekcije *questions* koje posjeduje taj ID (kôd 4.6.).

```
const handleDeleteQuestion = async (id) => {
    if (!user) {
      setError("You must be logged in to delete a question");
      return;
    }
    try {
      await deleteDoc(doc(firestore, "questions", id));
      fetchQuestions();
    } catch (error) {
      console.error("Error deleting question:", error);
      setError("Failed to delete question. Please try again.");
    }
  };
```

Kôd 4.6. *Brisanje pitanja*

4.3.2. Vođenje kviza u stvarnom vremenu

Korištenje ove aplikacije uvelike pojednostavljuje vođenje kviza u stvarnom vremenu. Ideja je da aplikacija služi kao alternativa drugim alatima za vođenje kvizova koji se trenutno koriste, a tu se konkretno misli na Microsoft PowerPoint prezentaciju u kojoj se prikazuju pitanja te Microsoft Excel za pohranjivanje bodova i uspjeha ekipa. Zamišljeno je da se voditelj kviza prijavi u aplikaciju i jedino on ima pristup svojim pitanjima, dodatno prilagodi opcije kviza, željena pitanja te nazive ekipa koje se natječu. Ekipe bi još uvijek morale ispred sebe imati manual za odgovore koje bi na kraju kruga predali voditelju kviza. Voditelj kviza pokreće kviz i na ekranu se počnu pojavljivati pitanja. Na kraju svakog kruga, voditelj ispravlja odgovore i unosi rezultate ekipa u poseban odjeljak aplikacije za bodovanje i tako do konačnih rezultata.

Za voditelja kviza, vođenje kviza u stvarnom vremenu je neophodna funkcionalnost. Ono omogućuje dinamično i interaktivno iskustvo za voditelja kviza i sudionike. Ova funkcionalnost je realizirana prvenstveno kroz *QuizScreen* komponentu, koja upravlja tokom kviza od početka do kraja.

Ključni aspekti implementaciji uključuju:

- Prikaz pitanja: Pitanja su prikazana jedno po jedno, koristeći React useState *hook* za praćenje trenutnog pitanja. Dohvaćanje pitanja se događa u nekoliko koraka. Prvo se u funkciji *fetchQuestions* događa inicijalno dohvaćanje (kôd 4.7) u kojem se upituje bazu podataka Firestore za sva pitanja koja pripadaju trenutnom korisniku. Zatim, pitanja se nasumičnog miješaju koristeći *sort* i *Math.random* funkcije. Na kraju ih se pohranjuje u *allQuestions* stanje.

```
const fetchQuestions = async () => {
  try {
    const q = query(
      collection(firestore, "questions"),
      where("userId", "==", user.uid)
    );
    const querySnapshot = await getDocs(q);
    const fetchedQuestions = querySnapshot.docs.map((doc) => ({
      id: doc.id,
      ...doc.data(),
    }));
    // ... upravljanje greškama ...
    const shuffled = fetchedQuestions.sort(() => 0.5 - Math.random());
    setAllQuestions(shuffled);
    setIsLoading(false);
  } catch (error) {
    // ... upravljanje greškama handling ...
  }
};
```

Kôd 4.7. *Inicijalno dohvaćanje podataka*

Nakon toga, funkcija je pozvana u prvom useEffect *hook*-u (kôd 4.8.) kada se *allQuestions* ili *currentRound* promijene te se tada pokreću pripreme za trenutni krug. Filtriraju se pitanja koja su već korištena (pohranjena u *usedQuestionIds*), odabire podskup ovih nekorištenih pitanja na temelju postavke *questionsPerRound* te se ova odabrana pitanja pohranjuje u *currentRoundQuestions*.

```
useEffect(() => {
  if (allQuestions.length > 0 && quizSettings) {
    const unusedQuestions = allQuestions.filter(
      (q) => !usedQuestionIds.includes(q.id)
    );
    const newRoundQuestions = unusedQuestions.slice(
      0,
      quizSettings.questionsPerRound
    );
    // ... error handling ...
    setCurrentRoundQuestions(newRoundQuestions);
    setCurrentQuestion(0);
  }
}, [currentRound, allQuestions, quizSettings, usedQuestionIds]);
```

Kôd 4.8. *Kuka useEffect za pripremu kruga*

Pitanja se zatim prikazuju jedan po jedan birana iz *currentRoundQuestions* stanja. Po završetku kruga, ID-ovi pitanja korištenih u ovom krugu dodaju se u stanje *usedQuestionIds* te se ovaj ažurirani popis pohranjuje u lokalnu pohranu kako bi mu se moglo pristupati tijekom sesije (kôd 4.9.).

```
const handleViewCorrectAnswers = () => {
    const newUsedQuestionIds = [
      ...usedQuestionIds,
      ...currentRoundQuestions.map((q) => q.id),
    ];
    setUsedQuestionIds(newUsedQuestionIds);
    localStorage.setItem("usedQuestionIds",
JSON.stringify(newUsedQuestionIds));

    navigate("/correct-answers", {
      state: {
        round: currentRound,
        questions: currentRoundQuestions,
      },
    });
  };
```

Kôd 4.9. *Funkcija za baratanje prikazom točnih odgovora*

Za svaku iduću rundu, problem dohvaćanja pitanja se rješava kombinacijom koraka pripreme runde i stanja *usedQuestionIds*. Kada započne novi krug, korak pripreme kruga automatski će isključiti prethodno korištena pitanja. Sve se ovo događa u početnom useEffecte-u (kôd 4.10.).

Ovaj kôd provjerava postoji li postojeći krug u tijeku i učitava odgovarajuće stanje ili pokreće novi kviz ako nije.

```
useEffect(() => {
  // ... druge postavke ...
  if (location.state && location.state.round) {
    setCurrentRound(location.state.round);
    const storedUsedQuestionIds = JSON.parse(
      localStorage.getItem("usedQuestionIds") || "[]"
    );
    setUsedQuestionIds(storedUsedQuestionIds);
  } else {
    setCurrentRound(1);
    setUsedQuestionIds([]);
    localStorage.removeItem("usedQuestionIds");
  }
  fetchQuestions();
}, [user, navigate, location]);
```

Kôd 4.10. *Kuka useEffect za ostale runde*

- Odbrojavanje vremena: Za svako pitanje implementirano je odbrojavanje vremena. Ovo je postignuto korištenjem useEffect *hook*-a. Preostalo vrijeme prikazano je vizualno pomoću Material-UI *LinearProgress* komponente, koja se ažurira u stvarnom vremenu. Tajmer se inicijalizira tako da mu se doda vrijednost u iznosu vremena odabranog u postavkama kviza. Pri prelasku na iduće i prethodno pitanje ono se resetira na jednaku vrijednost. Samo odbrojavanje vremena (kôd 4.11.) riješeno je na način da se uzme brojka koja je definirana u postavkama kviza i da se svakih 1000 milisekundi umanji za 1. Ako ta brojka dođe do 0, poziva se funkcija za baratanje idućim pitanjem. Funkcija čišćenja (*clearTimeout*) osigurava pravilno brisanje mjerača vremena kada se komponenta demontira ili kada se *timeLeft* promijeni.

```
useEffect(() => {
if (timeLeft > 0) {
const timer = setTimeout(() => setTimeLeft(timeLeft - 1), 1000);
return () => clearTimeout(timer);
} else if (timeLeft === 0 && quizSettings) {
handleNextQuestion();
}
}, [timeLeft, quizSettings, handleNextQuestion]);
```

Kôd 4.11. *Funkcija odbrojavanja vremena*

- Navigacija kroz pitanja: Implementirane su kontrole za ručnu navigaciju kroz pitanja, bilo prethodnog ili sljedećeg (kôd 4.12.), kao i automatski prijelaz na sljedeće pitanje

nakon isteka vremena. Ovo je postignuto kombinacijom useState *hook*-a za praćenje trenutnog indeksa pitanja i useCallback *hook*-a za optimizaciju performansi funkcija navigacije.

```
const handleKeyPress = useCallback(
    (event) => {
      if (event.key === "ArrowRight") {
        handleNextQuestion();
      } else if (event.key === "ArrowLeft") {
        handlePrevQuestion();
      }
    },
    [handleNextQuestion, handlePrevQuestion]
  );
```

Kôd 4.12. *Manualna navigacija kroz pitanja*

- Upravljanje rundama: Implementirana je logika za upravljanje rundama kviza. Na kraju svake runde, prikazuje se sažetak runde (*RoundSummary* komponenta) prije prelaska na točne odgovore. Ova komponenta služi kao svojevrsno sumiranje kruga u smislu da natjecateljima daje još malo vremena da završe svoja razmišljanja i upišu odgovore koje dosad nisu napisali i eventualno pogledaju neko prethodno pitanje ako iz bilo kojeg razloga nisu uspjeli.
- Prikaz točnih odgovora: Nakon svakog kruga, implementiran je prikaz točnih odgovora. Ovo je realizirano kroz zasebnu *CorrectAnswers* komponentu, koja se prikazuje kao modalni dijalog na kraju svakog kruga. Do ekrana prikaza točnih odgovora se može doći kako je već spomenuto u funkcionalnosti prikaza pitanja.
- Sinkronizacija s bazom podataka: Implementirana je sinkronizacija u stvarnom vremenu s Firebase Firestore bazom podataka. Ovo omogućuje ažuriranje stanja kviza u stvarnom vremenu, što je posebno korisno u scenarijima gdje više osoba prati isti kviz, odnosno u realističnoj kvizaškoj situaciji.
- Obrada korisničkog unosa: Iako sudionici ne unose odgovore direktno u aplikaciju, implementirano je sučelje za voditelja kviza da označi točne odgovore i dodijeli bodove timovima. Ovo je realizirano kroz *Scoreboard* komponentu, koja se prikazuje nakon svake runde.

- Upravljanje stanjem kviza: Globalno stanje kviza (trenutna runda, ukupni bodovi timova, itd.) upravljano je korištenjem React Context API-ja. Ovo omogućuje učinkovito dijeljenje stanja između različitih komponenti.
- Pohrana podataka: Nakon svakog uspješno završenog kruga, odnosno pri pristupanju ekrana konačnih rezultata, dolazi do pohrane podataka o trenutno odigranom kvizu, kao i o rezultatima. Uz to, implementirana je i arhiva ekipa za ekipe koje su se pojavljivali više puta kako im svi prijašnji rezultati bili na jednom mjestu i ostali upamćeni.

4.3.3. Bodovanje i rangiranje ekipa

Bodovanje i rangiranje ekipa doprinose kompetitivnom duhu kvizova, a u ovoj aplikaciji ostvareni su tako da se dio provodi ručno (unos bodova) dok se dio provodi automatizmom (rangiranje ekipa).

Implementirano je sučelje za unos bodova unutar *Scoreboard* komponente. Za svaku ekipu, voditelj kviza može unijeti broj točnih odgovora nakon svake runde. Korišteni su Material-UI *TextField* elementi s numeričkim ograničenjima kako bi se spriječili nevažeći unosi tako da je minimalan broj bodova 0 a maksimalan onaj broj koliko pitanja ima. U kôdu se nalazi varijabla koja označava broj bodova po pitanju i trenutno je postavljena na 1 ali to ovisi o voditelju kviza te željenom načinu bodovanja. Za upravljanje unosom novih rezultata zadužena je funkcija *handleSubmitScore* (kôd 4.13.) koja stvara novi niz *updatedTeams* tako što preslikava preko (engl. *Mapping over*) postojećeg niza *teams*. Za svaku ekipu, kreira objekt koji širi (engl. *Spread*) sva postojeća svojstva ekipe, ažurira rezultat ekipa tako što dodaje brojku trenutnom rezultatu (ili dodaje 0, ako nikakav broj nije odabran) te dodaje novi rezultat nizu *scoresByRound* (ili ga stvara ako on ne postoji). Implementirana je logika za automatsko ažuriranje ukupnog rezultata svake ekipe nakon unosa bodova za rundu. Ovo je postignuto korištenjem useEffect *hook*-a koji se aktivira pri svakoj promjeni bodova runde.

```
const handleSubmitScores = () => {
  const updatedTeams = teams.map((team) => ({
    ...team,
    score: (team.score || 0) + (newScores[team.name] || 0),
    scoresByRound: [...(team.scoresByRound || []), newScores[team.name] ||
0],
  }));
```

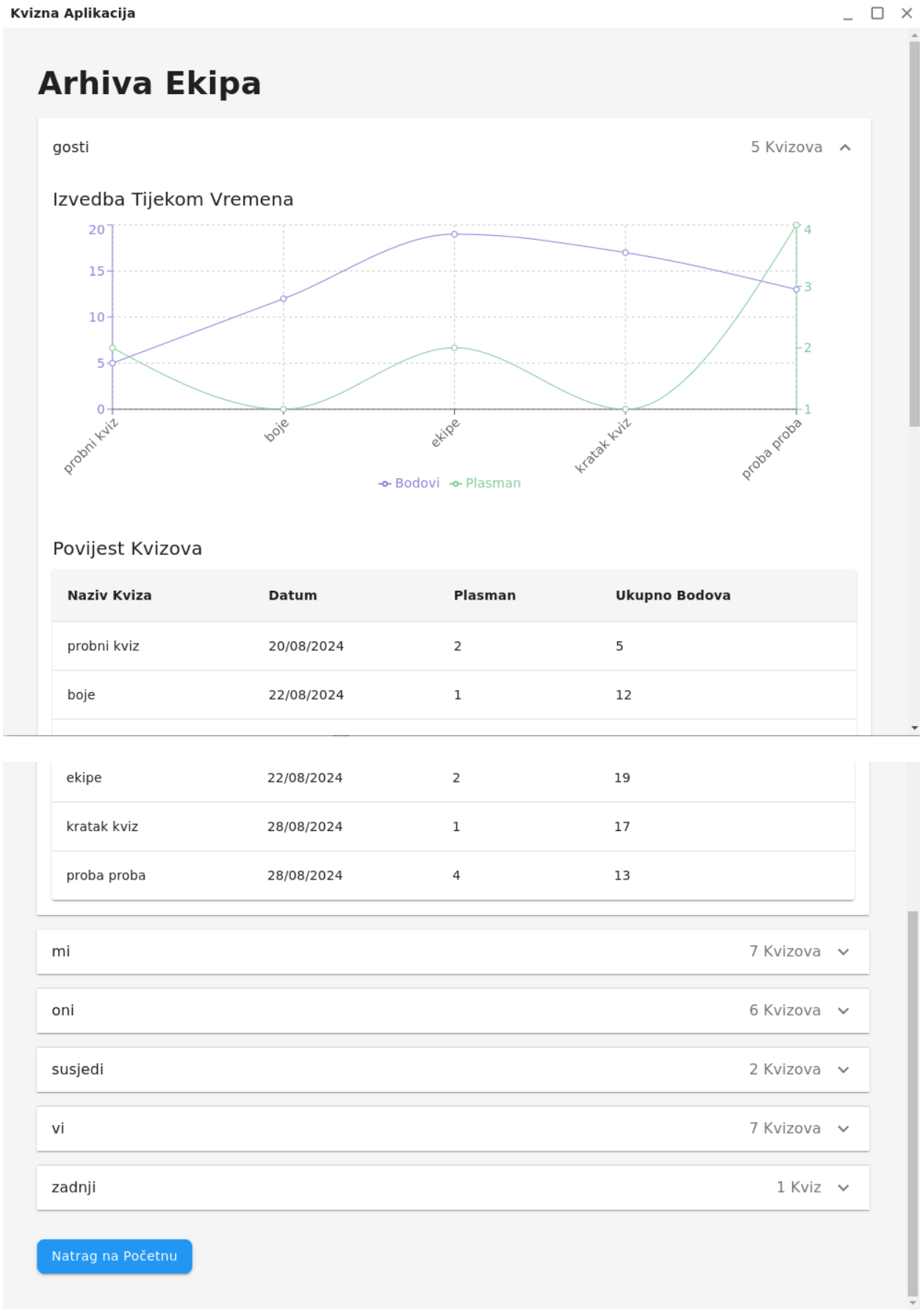
Kôd 4.13. *Funkcija za ažuriranje bodova*

Implementiran je algoritam za automatsko rangiranje ekipa na temelju njihovih ukupnih bodova. Funkcije iz kôda 4.14 služe za, redom, sortiranje ekipa po broju bodova u padajućem redoslijedu, dodjeljivanje mjesta u trenutnom poretku na temelju njihovog položaja u sortiranom nizu te ažuriranje stanja timova s novim rezultatima i plasmanima.

```
updatedTeams.sort((a, b) => b.score - a.score);
updatedTeams.forEach((team, index) => {
    team.placement = index + 1;
  });
setTeams(updatedTeams);
```

Kôd 4.14. *Funkcije za dodjeljivanje bodova i rezultata ekipa*

Trenutni poredak ekipa prikazan je u realnom vremenu koristeći Material-UI Table komponentu. Bodovi i rangiranje pohranjeni su u Firebase Firestore bazi podataka. Implementirana je funkcionalnost za pregled povijesnih rezultata ekipa kroz različite kvizove. Ovo je realizirano kroz *TeamArchives* komponentu (slika 4.17), koja prikazuje grafove performansi ekipa kroz vrijeme koristeći *recharts* biblioteku.



4.17. *Arhiva ekipa*

4.4. Višejezičnost i promjena tema

S ciljem povećanja dostupnosti i prilagodljivosti aplikacije širem krugu korisnika, implementirane su funkcionalnosti višejezičnosti i promjene tema. Ove funkcionalnosti omogućuju korisnicima personalizaciju iskustva korištenja aplikacije prema vlastitim preferencijama i potrebama.

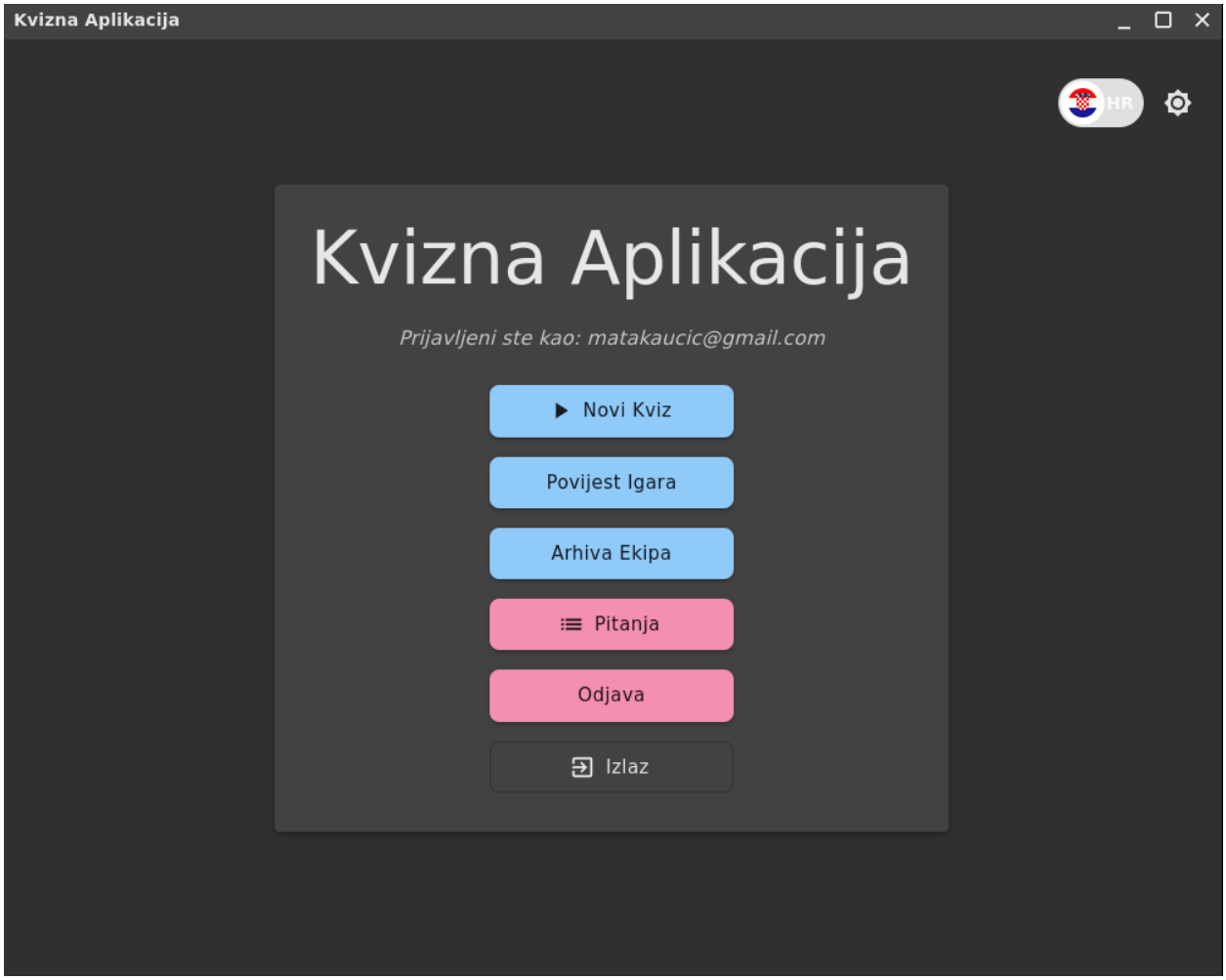
Implementacija višejezičnosti realizirana je korištenjem React Context API-ja i prilagođenog *hook*-a. Kreirana je JavaScript datoteka *LanguageContext* koja obuhvaća trenutno odabrani jezik i funkciju za promjenu jezika. Ovaj kontekst je dostupan svim komponentama u aplikaciji. Svi tekstualni elementi u aplikaciji ekstrahirani su u zasebne jezične datoteke (*translations.js*). Ova datoteka sadrži objekte s ključevima za svaki prevodivi tekst, s odgovarajućim prijevodima za svaki podržani jezik. Implementiran je prilagođen *hook useLanguage* koji omogućuje jednostavno dohvaćanje trenutnog jezika i funkcije za promjenu jezika u bilo kojoj komponenti. Kreirana je *LanguageToggle* komponenta (slike 4.18. i 4.19.) koja omogućuje korisniku jednostavnu promjenu jezika aplikacije. Ova komponenta prikazuje zastave zemalja čiji su jezici podržani a vidljiva je na početnom zaslonu. U svim komponentama, tekstualni elementi zamijenjeni su pozivima funkcije za prijevod, npr. `t('quizApp')` umjesto fiksnog teksta `"Quiz App"`.

Slika 4.18. *Gumb za odabir jezika kada je odabran hrvatski jezik*

Slika 4.19. *Gumb za odabir jezika kada je odabran engleski jezik*

Implementacija promjene tema između svijetle i tamne realizirana je korištenjem Material-UI *ThemeProvider*-a i prilagođenog *hook*-a. Koristi se *createAppTheme* funkcija koja stvara svijetlu i tamnu temu. Ova funkcija prilagođava različite aspekte teme uključujući boje, tipografiju i stilove komponenti a svi se ti podaci nalazi u JavaScript komponenti *theme.js*. Kada se pozove funkcija *toggleTheme*, gumbom na početnom zaslonu, ona ažurira stanje načina rada, što

zauzvrat uzrokuje ponovni izračun teme i ponovno renderiranje cijele aplikacije s novom temom (slika 4.20.).



Slika 4.20. *Tamni način rada aplikacije*

4.5. Integracija s Firebase-om

Integracija s Firebase-om predstavlja ključni aspekt *backend* funkcionalnosti aplikacije za kvizove. Firebase, kao *Backend-as-a-Service* (*BaaS*) platforma, omogućuje implementaciju robusnih rješenja bez poslužitelja (engl. *Serverless*) za autentifikaciju, pohranu podataka i sinkronizaciju u stvarnom vremenu. U ovom poglavlju detaljno je opisana implementacija integracije s Firebase-om u kontekstu aplikacije za stvaranje i vođenje kvizova.

Implementacija započinje inicijalizacijom Firebase-a u aplikaciji. Kreirana je zasebna datoteka firebase.js koja sadrži konfiguraciju i inicijalizaciju Firebase instance kao što je prikazano na kôdu 4.15.

```
import { initializeApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';
import { getFirestore } from 'firebase/firestore';

const firebaseConfig = {
  // Konfiguracija specifična za projekt
};

const app = initializeApp(firebaseConfig);
export const auth = getAuth(app);
export const firestore = getFirestore(app);
```

Kôd 4.15. *Inicijalizacija Firebase-a u vidu firebase.js komponente*

U području u kojem piše „konfiguracija specifična za projekt“ navode se stvari poput API ključa, autorove domene, ID projekta i slično.

4.5.1. Autentifikacija korisnika

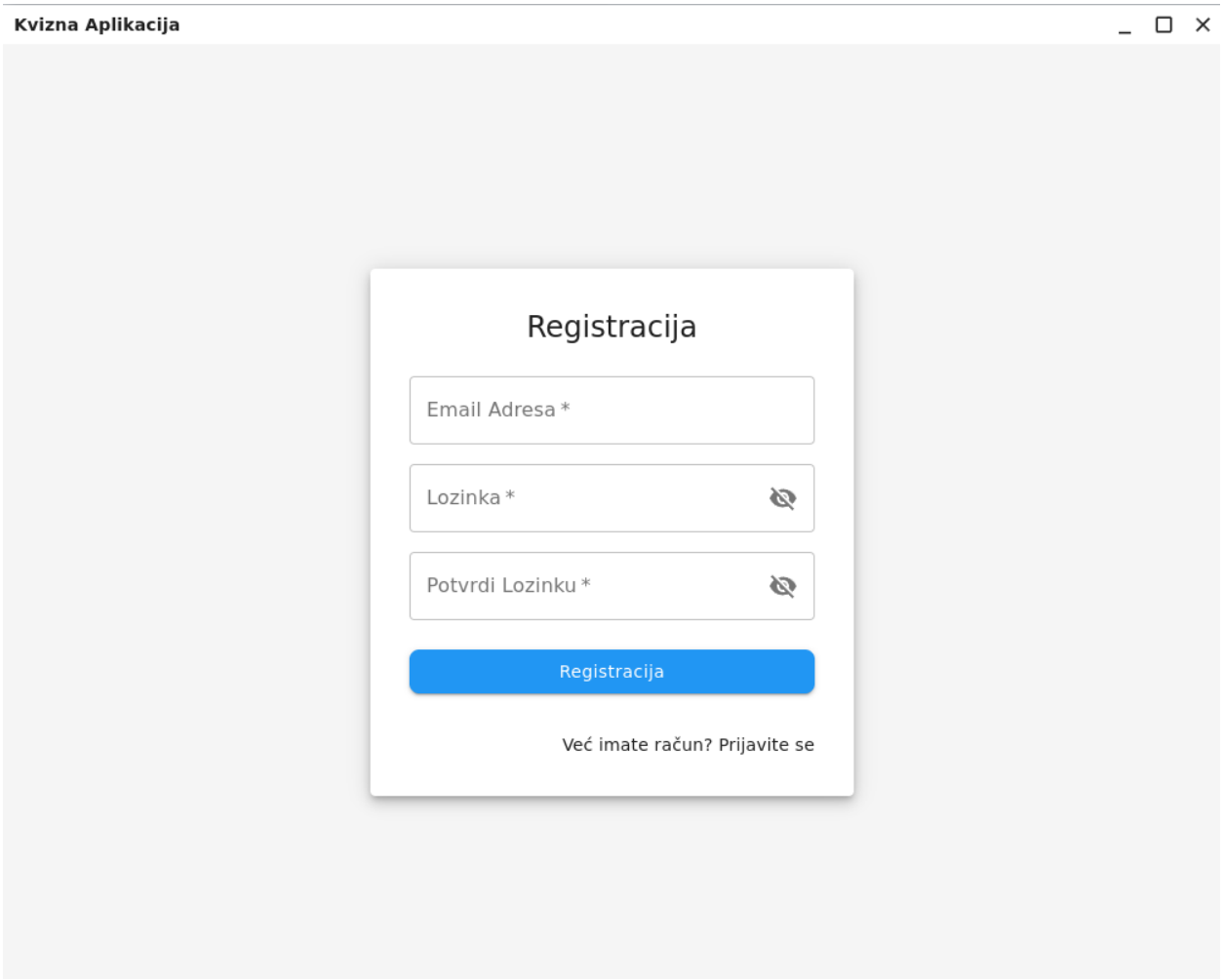
Implementacija autentifikacije korisnika realizirana je koristeći Firebase Authentication, pružajući sigurno i pouzdano rješenje za upravljanje korisničkim računima. Ova implementacija obuhvaća registraciju novih korisnika, prijavu postojećih korisnika, odjavu i praćenje stanja autentifikacije.

Dodana je funkcija za registraciju novih korisnika koristeći *createUserWithEmailAndPassword* metodu (kôd 4.16.).

```
const handleSignup = async (e) => {
    e.preventDefault();
    if (password !== confirmPassword) {
      setError("Passwords do not match");
      return;
    }
    try {
      await createUserWithEmailAndPassword(auth, email, password);
      console.log("User registered successfully");
      navigate("/");
    } catch (error) {
      console.error("Error registering user:", error.message);
      setError(error.message);
    }
  };
```

Kôd 4.16. *Funkcija za registraciju korisnika*

Grafički izgled forme za registraciju prikazan je na slici 4.21.



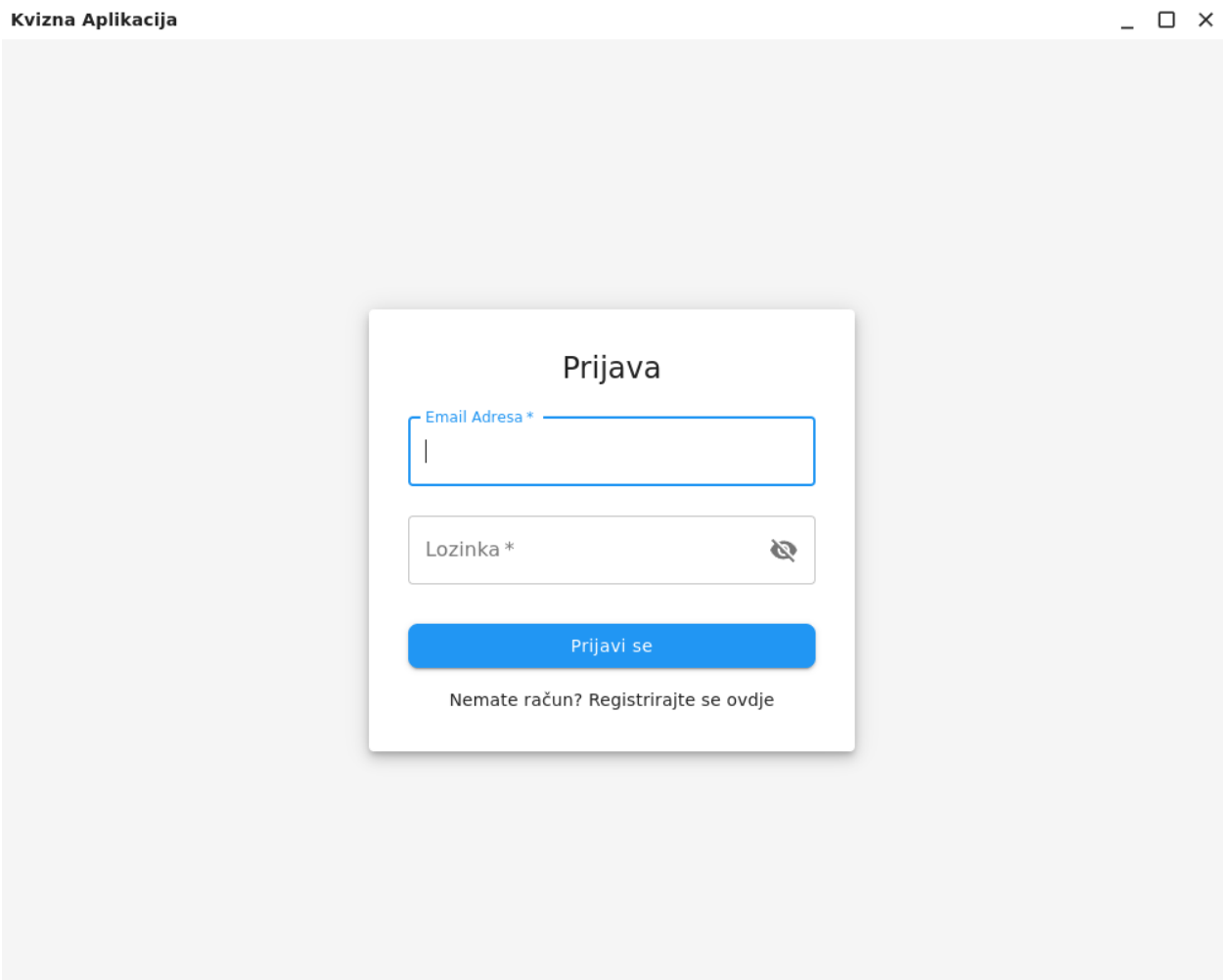
Slika 4.21. *Forma za registraciju korisnika*

Vrlo slično tome, dodane je i funkcija za prijavu postojećih korisnika koristeći *signInWithEmailAndPassword* metodu (kôd 4.17.).

```
const handleLogin = async (e) => {
    e.preventDefault();
    try {
      await signInWithEmailAndPassword(auth, email, password);
      console.log("User logged in successfully");
      navigate("/"); // Navigate to home page immediately after successful
login
    } catch (error) {
      console.error("Error logging in:", error.message);
      setError(error.message);
    }
  };
```

Kôd 4.17. *Funkcija za prijavu korisnika*

Grafički izgled forme za prijavu prikazan je na slici 4.22.



Slika 4.22. *Forma za prijavu postojećih korisnika*

Trenutno prijavljeni korisnik prikazan je na početnom zaslonu ispod naziva aplikacija. Dostupna je i opcija odjave korisnika koja se također nalazi na početnom zaslonu a pritiskom na taj gumb korisnik je preusmjeren na ekran prijave korisnika. Kroz cijelu aplikaciju korišten je useAuthState *hook* za praćenje stanja autentifikacije korisnika.

4.5.2. Pohrana podataka o kvizovima i rezultatima

Implementacija pohrane podataka o kvizovima i rezultatima realizirana je koristeći Cloud Firestore, NoSQL bazu podataka u oblaku koju pruža Firebase. Ova implementacija osigurava učinkovito spremanje, dohvaćanje i ažuriranje podataka vezanih za kvizove i rezultate timova. Podaci u Firestore-u su organizirani u tri kolekcije: *gameHistory* koja sadrži sve dokumente koji predstavljaju pojedinačne odigrane kvizove; *teamArchives* koja sadrži dokumente koji predstavljaju arhivu rezultata za svaku ekipu te *questions* koja sadrži dokumente koji predstavljaju pitanja te njihovog autora.

Princip spremanja podataka o kvizovima ostvaren je pomoću funkcije saveGameHistory (kôd 4.18) kojemu se može pristupiti s početnog ekrana. Ova funkcija je umotana u useCallback *hook* zbog optimizacije performansi i radi samo ako postoji korisnik, igra još nije spremljena i postoje timovi za spremanje. U nastavku će biti objašnjeni pojedini dijelovi funkcije.

```
const saveGameHistory = useCallback(async () => {
  if (!user || gameSaved || teams.length === 0) return;
  // ...
}, [user, gameSaved, teams, quizSettings, saveTeamResults]);
```

Kôd 4.18. *Funkcija za spremanje kviza u povijest igara*

Dio kôda (4.19) za kreiranje unikatnog ID-a i formatiranje datum i vremena jer sam htio da podaci odgovaraju europskom stilu pisanja datuma i vremena:

```
const now = new Date();
const quizId = now.getTime().toString();
const formattedDate = `${now.getDate().toString().padStart(2,
"0")}/${(now.getMonth() + 1).toString().padStart(2,
"0")}/${now.getFullYear()}`;
const formattedTime = `${now.getHours().toString().padStart(2,
"0")}:${now.getMinutes().toString().padStart(2, "0")}`;
```

Kôd 4.19. *Kreiranje unikatnog ID-a i formatiranje datum i vremena*

Dio kôda (4.20.) za računanje maksimalnog broja bodova koji će biti korišten u ekranu statistike učinka ekipa. Također, *pointsPerQuestion* je već spomenuta varijabla koja označava brojčanu vrijednost svakog pitanja koja se može mijenjati po želji voditelja kviza.

```
const pointsPerQuestion = 1;
const totalPossiblePoints = quizSettings.rounds *
quizSettings.questionsPerRound * pointsPerQuestion;
```

Kôd 4.20. *Računanje maksimalnog broja bodova*

Kreacija objekta povijesti igre, spremanje iste u Firestore te spremanje individualnih rezultata ekipa koje će se koristiti za arhivu ekipa prikazano je kôdom 4.21. a time i završava pregled *saveGameHistory* funkcije. Ova funkcija je pozvana u useEffect *hook*-u koja se aktivira pri pokretanju ekrana konačnih rezultata.

```
const newGame = {
  quizId: quizId,
  userId: user.uid,
  date: formattedDate,
  time: formattedTime,
  quizName: quizSettings.quizName || "Unnamed Quiz",
  teams: teams.map((team, index) => ({
    name: team.name,
    scoresByRound: team.scoresByRound || [],
    totalScore: team.score,
    placement: team.placement,
  })),
  rounds: quizSettings.rounds || 0,
  questionsPerRound: quizSettings.questionsPerRound || 0,
  totalPossiblePoints: totalPossiblePoints,
};

const docRef = await addDoc(collection(firestore, "gameHistory"), newGame);
console.log("Game history saved successfully with ID:", docRef.id);
setGameSaved(true);

await saveTeamResults(quizId);
```

Kôd 4.21. *Spremanje povijesti igre i individualnih rezultata*

Slično kao i spremanje povijesti igara, funkcija *saveTeamResults* pohranjuje rezultate svake ekipe u *teamArchives* kolekciju. Također, implementirana je i funkcija *fetchGames* za dohvaćanje igara trenutnog korisnika koja uspoređuje trenutni ID korisnika sa svim ID-ima u kolekciji *gameHistory* te ispisuje željene podatke o prijašnjim kvizovima poput naziva kviza, datumu i vremenu odigravanja, ekipama i slično.

Uz to, dodana je i funkcionalnost uređivanja i brisanja pojedinog kviza. Uređivanje već odigranog kviza trenutno omogućuje samo opcije promjene naziva kviza te datuma i vremena odigravanja kviza (slika 4.23.).

Slika 4.23. *Uređivanja odigranog kviza*

Kôd brisanja kviza iz povijesti igara prikazan je kôdom 4.22.

.

```
const handleDeleteGame = async (gameToDelete) => {
  try {
    const batch = writeBatch(firestore);

    const gameRef = doc(firestore, "gameHistory", gameToDelete.id);
    batch.delete(gameRef);

    for (const team of gameToDelete.teams) {
      const teamRef = doc(firestore, "teamArchives",
`${auth.currentUser.uid}_${team.name}`);
      const teamDoc = await getDoc(teamRef);
      if (teamDoc.exists()) {
        const teamData = teamDoc.data();
        const updatedQuizzes = teamData.quizzes.filter(
          (quiz) => quiz.quizId !== gameToDelete.quizId
        );
        if (updatedQuizzes.length === 0) {
          batch.delete(teamRef);
        } else {
          batch.update(teamRef, { quizzes: updatedQuizzes });
        }
      }
    }

    await batch.commit();
    console.log("Game and related team archive entries deleted
successfully");
    await fetchGames();
  } catch (error) {
    console.error("Error deleting game and updating team archives:", error);
  }
};
```

Kôd 4.22. *Funkcija za brisanje kviza iz povijesti igara*

Funkcija prima objekt *gameToDelete* kao parametar koji sadrži informacije o kvizu kojeg treba izbrisati. Pokreće operaciju skupnog pisanja (engl. *Batch write*) za Firestore što omogućuje atomsko izvođenje više operacija koje su sve uspješne ili neuspješne. Stvara referencu na dokument kviza u kolekciji *gameHistory* i dodaje operaciju brisanja u skup, a zatim ponavlja postupak za svaku ekipu koja je sudjelovala u tom kvizu. Dakle, funkcija stvara referencu na dokument ekipe u kolekciji *teamArchives*, dohvaća trenutne podatke za tu ekipu, ako dokument ekipe postoji, filtrira se unos kviza koji odgovara igri koja se briše. Ako nakon filtriranja za ekipu nema preostalih kvizova, briše se cijeli dokument ekipe. U suprotnom, dokument ekipe se ažurira novim popisom kvizova (isključujući izbrisanu igru). Nakon obrade svih timova, izvršava skupnu operaciju *commit* te ako je uspješna, bilježi poruku o uspjehu i poziva *fetchGames* za osvježavanje popisa igara za ažuriranje korisničkog sučelja. Ako se u bilo kojem trenutnu pojavi pogreška, hvata pogrešku i bilježi ju u konzolu.

4.6. Prilagodba za *desktop* okruženje

Implementacija *desktop* verzije aplikacije za kvizove realizirana je korištenjem razvojnim okruženjem Electron.js, koji omogućuje razvoj višeplatformskih *desktop* aplikacija koristeći web tehnologije. Ova prilagodba osigurava nativno korisničko iskustvo i dodatne funkcionalnosti specifične za *desktop* okruženje

Osnovna konfiguracija Electron.js aplikacije definirana je u *main.js* datoteci (kôd 4.23.):

```
const { app, BrowserWindow, ipcMain } = require("electron");
const path = require("path");
const isDev = process.env.NODE_ENV === "development";
function createWindow() {
  const win = new BrowserWindow({
    width: 1000,
    height: 800,
    frame: false,
    transparent: true,
    webPreferences: {
      nodeIntegration: false,
      contextIsolation: true,
      enableRemoteModule: false,
      preload: path.join(__dirname, "preload.js"),
    },
  });

  win.loadURL(
    isDev
      ? "http://localhost:3000"
      : `file://${path.join(__dirname, "../build/index.html")}`
  );
  win.removeMenu();
  if (isDev) {
    win.webContents.openDevTools();
  }
  ipcMain.on("minimize", () => win.minimize());
  ipcMain.on("maximize", () => {
    if (win.isMaximized()) {
      win.unmaximize();
    } else {
      win.maximize();
    }
  });
  ipcMain.on("close", () => win.close());
}
app.whenReady().then(createWindow);
app.on("window-all-closed", () => {
  if (process.platform !== "darwin") {
    app.quit();
  }
});
app.on("activate", () => {
  if (BrowserWindow.getAllWindows().length === 0) {
    createWindow();
  }
});
ipcMain.on("quit-app", () => {
  app.quit();
});

if (isDev) {
  require("electron-reload")(__dirname, {
    electron: path.join(__dirname, "..", "node_modules", ".bin", "electron"),
    forceHardReset: true,
    hardResetMethod: "exit",
  });
}
```

Kôd 4.23. *Osnovna konfiguracija Electron-a u aplikaciji*

Ova datoteka postavlja glavni proces Electron aplikacije, stvarajući prozor aplikacije, rukujući kontrolama prozora, upravljajući životnim ciklusom aplikacije i postavljajući razvojne alate i prilagođeno ponovno učitavanje u razvojnom načinu. Dizajnirana je za rad s React aplikacijom bez ikakvih poteškoća, učitavajući je s razvojnog poslužitelja ili iz izgrađenih (engl. *Build*) datoteka.

Kako bi se postigao dojam nativne *desktop* aplikacije, implementirana je prilagođena naslovna traka. Ovo je realizirano kroz *TitleBar* komponentu koja grafički prikazuje naslovnu traku koja sadrži ikone za minimiziranje, uvećanje i zatvaranje aplikacije. Naslovna traka se također mijenja prilikom promjene teme što nije bio slučaj s uobičajenom naslovnom trakom Electron aplikacije.

Za omogućavanje komunikacije između glavnog procesa i procesa renderiranja, implementirana je *preload.js* skripta (kôd 4.24.).

```
const { contextBridge, ipcRenderer } = require("electron");

contextBridge.exposeInMainWorld("electronAPI", {
  minimize: () => ipcRenderer.send("minimize"),
  maximize: () => ipcRenderer.send("maximize"),
  close: () => ipcRenderer.send("close"),
  quit: () => ipcRenderer.send("quit-app"),
});
```

Kôd 4.24. *Komunikacija između procesa*

Radi osiguravanja konzistentnog odvajanja (engl. *Padding*) i listanja (engl. *Scrolling*) na *desktop* verziji, implementirana je komponenta višeg reda (engl. *Higher-Order Component* - HOC).

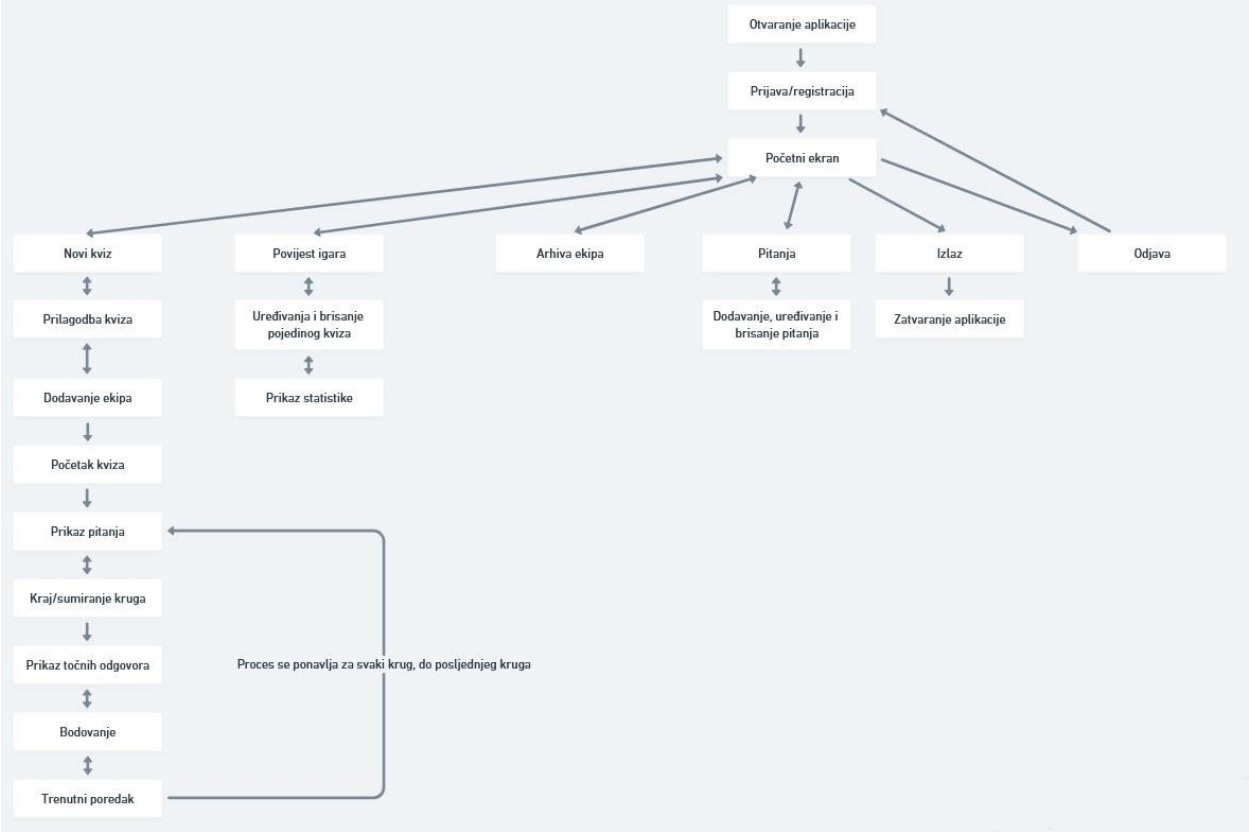
5. REZULTATI I ANALIZA

Rezultati razvijene aplikacije su i više nego zadovoljavajući. Postignuta je aplikacija za stvaranje i vođenje kvizova koja odgovara potrebama korisnika. Razvoj rezultirao je sveobuhvatnim rješenjem koje obuhvaća sljedeće ključne funkcionalnosti:

- Kreiranje i uređivanje kvizova s prilagodljivim brojem rundi i pitanja
- Upravljanje timovima i sudionicima
- Vođenje kviza u realnom vremenu s mjerenjem vremena
- Manualno bodovanje i automatsko rangiranje timova
- Pregled povijesti odigranih kvizova
- Analiza performansi timova kroz vrijeme
- Višejezična podrška (hrvatski i engleski)
- Prilagodljiva tema (svjetla i tamna)
- *Desktop* funkcionalnosti putem Electron-a
- Autentifikacija korisnika

5.1. Dijagram tijeka aplikacije

Dijagram tijeka aktivnosti aplikacije prikazan je slikom 5.1. Korisničko sučelje i ekrani su prikazani u prethodnim poglavljima.



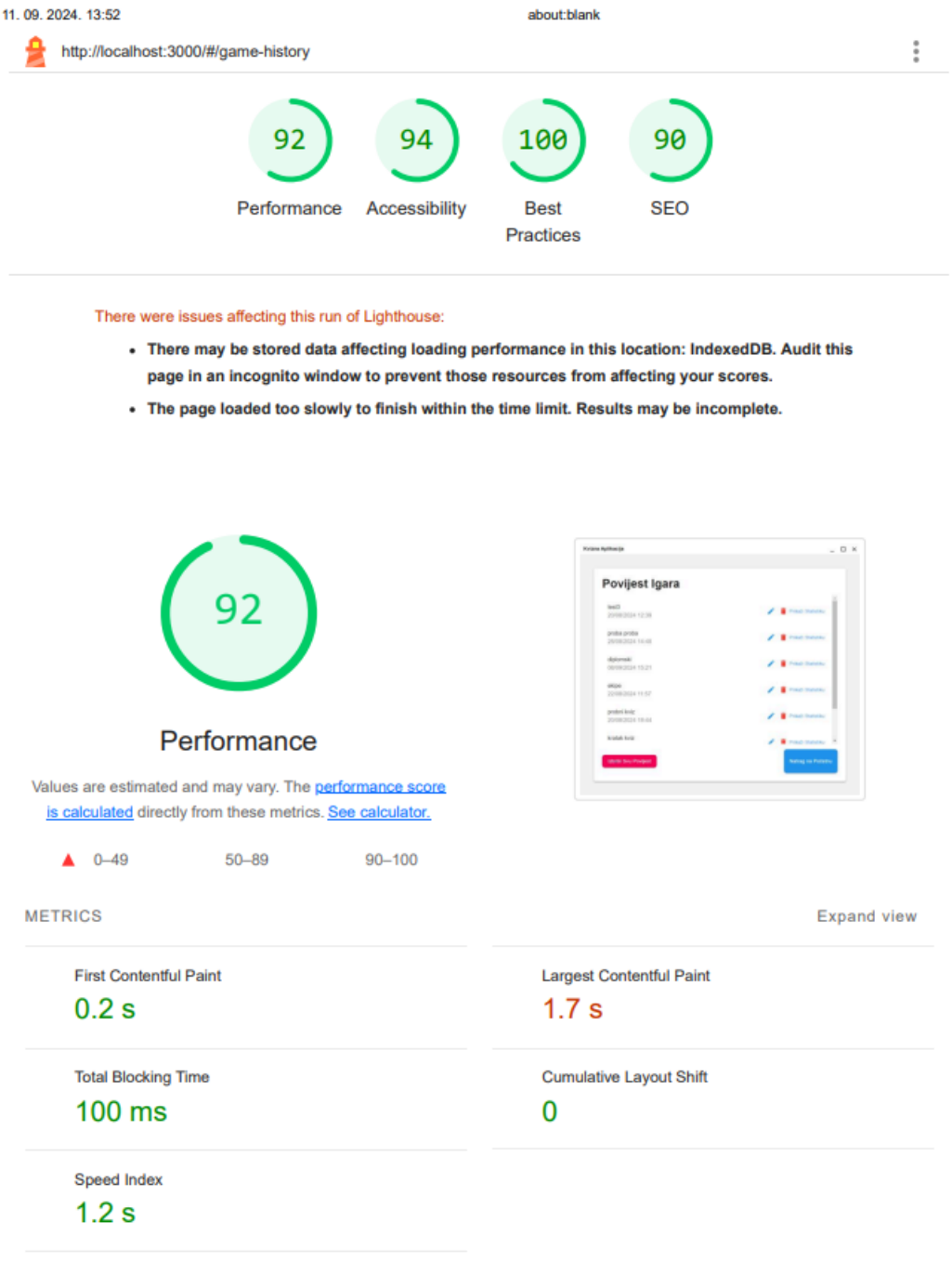
Slika 5.1. *Dijagram tijeka aktivnosti aplikacije*

5.2. Analiza performansi aplikacije

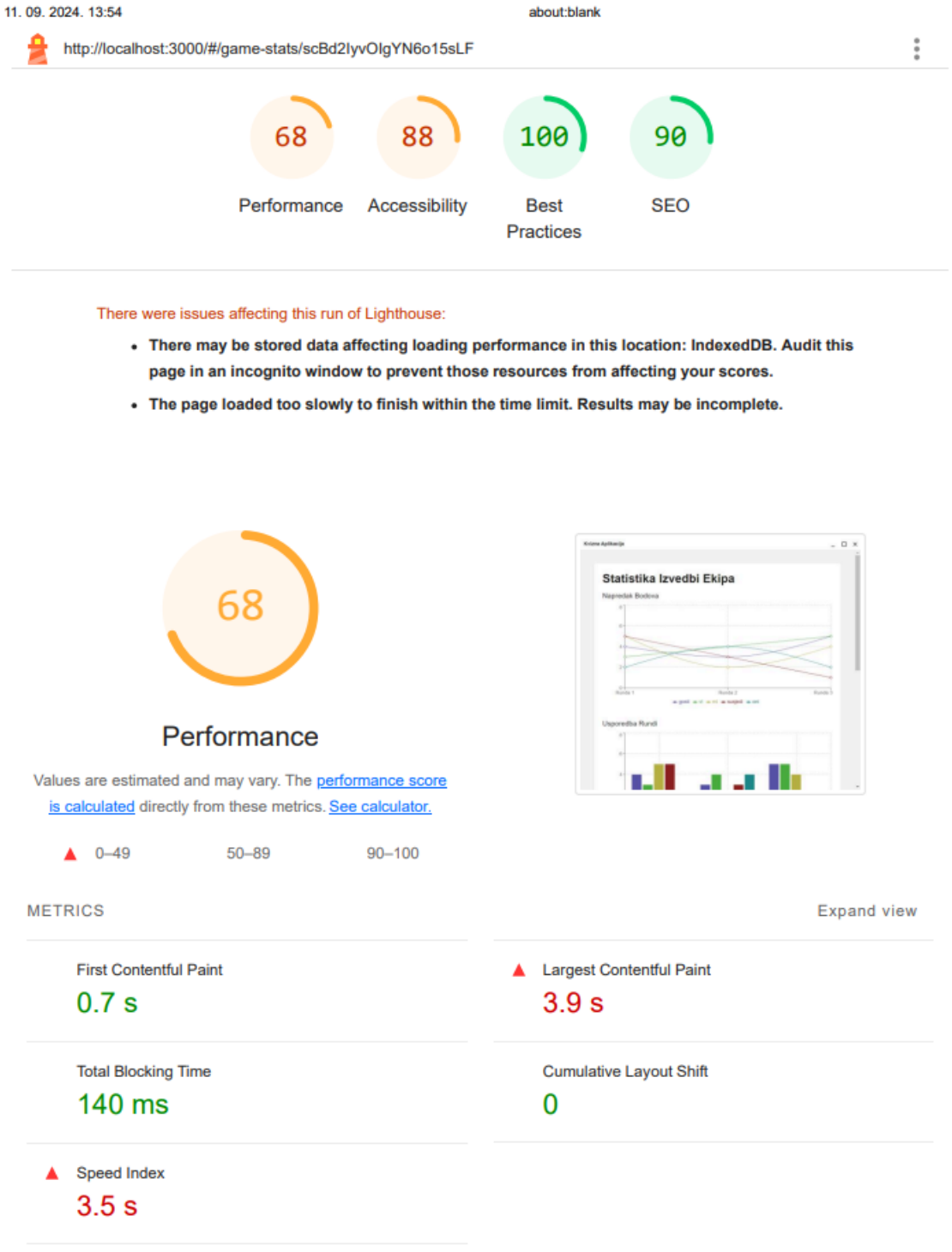
Analizu performansi aplikacije napravljena je korištenjem Google Lighthouse-a - automatiziranog alata otvorenog koda za mjerenje kvalitete web stranica. Ovaj alat možda nije optimalan za testiranje aplikacije razvijene u ovom diplomskog radu zbog toga što ova aplikacija nije u potpunosti slična ostalim web aplikacijama nego je primarno *desktop* aplikacija i rađena u programskom okviru Electron. No, ova aplikacija još uvijek koristi web tehnologija tako da je, u nedostatku optimalnih metričkih alata, testirana Google Lighthouse-om. Performanse su bile analizirane kroz devet ekrana s prosječnim rezultatima (od 100):

- Učinak (engl. *Performance*): 81,56
- Dostupnost (engl. *Accessibility*): 88,11
- Najbolji primjeri iz prakse (engl. *Best practices*): 100
- Optimizacija tražilice (engl. *Search engine optimization* - SEO) - 90,11

Najbolje performanse imaju ekrani točnih odgovora i povijesti igara (slika 5.2.), dok ekran vođenja kviza i ekran statistike učinka ekipa (slika 5.3.) imaju najviše prostora za poboljšanje.



Slika 5.2. *Analiza performansi ekrana povijesti igara*



Slika 5.3. *Analiza performansi ekrana statistike učinka ekipa*

Uobičajeni problemi svih zaslona su optimizacija JavaScript-a, tj. svi zasloni pokazuju potencijal za smanjenje JavaScript-a i smanjenje nekorištenog kôda. Zatim, mogućnosti za prethodno povezivanje sa potrebnim izvorima i poboljšanje predmemorije za statička sredstva što bi uvelike poboljšalo učitavanje resursa. Metrika koja označava koliko se brzo glavni sadržaj web stranice

učitava (LCP - *Largest Contentful Paint*) je vrlo loša na većini zaslona i trebalo bi ju poboljšati. Aplikacija sveukupno pokazuje dobru pristupačnost, s ocjenama u rasponu od 81 do 94. Glavni problemi su s označavanjem gumba, oznakama elemenata obrasca i redoslijedom naslova. SEO ima konstantno visoke rezultate, između 90 i 91, s primarnim prijedlogom da se dodaju meta opisi. Aplikacija pruža stabilno vizualno iskustvo, ima dobre CLS (engl. *Cumulative Layout Shift*) rezultate. Najviši mogući rezultati u segmentu „Najbolji primjeri iz prakse“ ukazuju na dobro strukturiranu i modernu web aplikaciju. Još jednom, ove analize performansi treba uzeti s rezervom jer smo aplikaciju analizirali kao web aplikaciju a ona zapravo služi kao *desktop* aplikacija.

6. ZAKLJUČAK

Aplikacija za stvaranje i vođenje kvizova uspješno je izrađena. Ovaj diplomski rad predstavio je proces razvoja i implementacije sveobuhvatne aplikacije za stvaranje i vođenje kvizova, kombinirajući web i *desktop* funkcionalnosti. Kroz detaljnu analizu zahtjeva, dizajn arhitekture, implementaciju ključnih funkcionalnosti i evaluaciju rezultata, ostvaren je cilj stvaranja robusnog i korisnički orijentiranog rješenja za organizaciju i provođenje interaktivnih kvizova.

Projekt je razvijen u razvojnom okruženju Microsoft Visual Code, koristeći JavaScript biblioteku React, razvojni okvir Electron, skalabilnu NoSQL bazu podataka u oblaku Firestore te Material UI za dizajn aplikacije. Projekt uspješno integrira navedene korištene tehnologije, stvarajući hibridno rješenje koje kombinira prednosti web i *destkop* aplikacija. Također, dane su teorijske podloge svake od korištenih tehnologija. Izgled aplikacije prikazan je dijagramom tijeka aktivnosti aplikacije i slikama korisničkog sučelja. Logika kviza dodatno je objašnjenja isječcima i objašnjenjima kôda.

Glavna svrha aplikacije je olakšavanje vođenja kviza organizatorima te poboljšan doživljaj igranja kviza sudionicima. Aplikacija je namijenjena da bude zabavnog, poučnog, ali i kompetitivnog karaktera. Korisničko sučelje dovoljno je jednostavno i intuitivno da ne bi trebalo biti nikakvog problema pri korištenju. Implementiran je i princip dualnog jezika, kao i mijenjanja između svijetle i tamne teme korisničkog sučelja kako bi se pristupilo većem spektru korisnika. Kroz proces razvoja, stečena su vrijedna iskustva u području *frontend* razvoja, integracije *backend* usluga i razvoja *desktop* aplikacija koristeći web tehnologije. Pri razvitku aplikacije došlo je do pojava raznih problema, konkretno - višestruko pojavljivanje istih pitanja tijekom jednog kviza te problem brisanja cijelom kviza iz povijesti igara, ali taj kviz bi se svejedno još uvijek referencirao u arhivi ekipa, što nije očekivano ponašanje. Ti problemi su se riješili ponovnim pisanjem logike kôda.

Usporedba s postojećim rješenjima pokazala je da razvijeno rješenje nudi jedinstvenu kombinaciju funkcionalnosti. Iako posjeduje limitiranu ciljanu publiku, ova aplikacija olakšava organizaciju i vođenje kvizova korištenjem samo jedne aplikacije, za razliku od minimalno dvije.

Potencijalne nadogradnje uključuju mogućnost postavljanja točnog poretka pitanja, dodavanje opcije kod kreacije pitanja u kojoj se odabire oblik pitanja (pitanje-odgovor, povezivanje, nadopunjavanje, više odgovora i sl.), uvođenje koncepta pitanja procjene ili neke druge metrike koja bi razriješila izjednačenja u poretku. Nadalje, dodavanje kategorije svakog pitanja kako bi

se mogli provoditi tematski kvizovi, pretraživanje pitanja u stvarnom vremenu kako bi se olakšao pronalazak željenog pitanja, dodavanje svojevrsne ljestvice uspješnosti ekipa kroz sve dosadašnje kvizove. Ukoliko bi tijekom kviza došlo do promjene vodeće ekipe, mogla bi se implementirani nekakva zvučna ili grafička notifikacija za to. Neke od mogućih nadogradnji su još i opcija ažuriranja korisničkog profila (dodavanje korisničkog imena, promjena lozinke, zaboravljena lozinka i sl.), točna poruka pogreške pri autentifikaciji („Korisnik s ovom e-mail adresom nije pronađen“, „Neispravna lozinka. Molimo pokušajte ponovno.“, „E-mail adresa je već u uporabi.“ i sl.). Mogućnost izvoza povijesti igara i ekipa putem CSV-a, uređivanje povijesti igara da se mogu mijenjati ostvareni i ukupni bodovi svake ekipe. S razine optimizacije performansi, implementacija *lazy loading*-a za pitanja kako bi se poboljšalo inicijalno vrijeme učitavanja te dodavanje mehanizama predmemoriranja za smanjenje upita baze podataka.

LITERATURA

[1] Službena stranica Hrvatskog kviz saveza, dostupno na: https://hrkviz.hr [28.6.2024.]

[2] Kratka povijest, Hrvatski kviz savez, dostupno na: https://hrkviz.hr/pub_kvizovi [28.6.2024.]

[3] QuizUp članak, Wikipedia, dostupno na: https://en.wikipedia.org/wiki/QuizUp [28.6.2024.]

[4] M. R. Dickey, This Is The Fastest-Growing iPhone Game Ever, Business Insider, dostupno na: https://www.businessinsider.com/quizup-success-2013-12 [28.6.2024.]

[5] Quizup has been discontinued, QuizUp Team, Glu Mobile, dostupno na: https://glumobile.helpshift.com/hc/en/94-quizup/faq/9680-quizup-has-been-discontinued/?s=general&f=quizup-to-be-discontinued&l=en [12.9.2024.]

[6] Službena stranice aplikacije Sporcle, dostupno na: https://www.sporcle.com/ [29.6.2024.]

[7] kviZDing!, Hrvatski kviz savez, dostupno na: https://kvizd.hr/kvizdarije/vijesti/kvizding [29.6.2024.]

[8] Little talks: VADYM BONDAR, Hrvatski kviz savez, dostupno na: https://kvizd.hr/en/inquizdition/news/little-talks-vadym-bondar [29.6.2024.]

[9] Službena stranica aplikacije Kahoot!, dostupno na: https://kahoot.com/ [30.6.2024.]

[10] T. Krotoff, Front-end frameworks popularity, GitHub Gist, dostupno na: https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190 [29.8.2024.]

[11] Službena stranica razvojnog okvira Electron, dostupno na: https://www.electronjs.org/ [29.8.2024.]

[12] E. Williams, Pros and Cons of Electron Js Development — Should You Use It?, Medium, dostupno na: https://medium.com/@emmaw4430/pros-and-cons-of-electron-js-development-should-you-use-it-407f67dae310 [29.8.2024.]

[13] Firestore službena stranica, dostupno na: https://cloud.google.com/firestore [29.8.2024]

[14] Material Design članak, Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Material_Design [29.8.2024.]

SAŽETAK

U radu je opisan proces razvoja aplikacije za stvaranje i vođenje kvizova. Kroz rad je opisano trenutno stanje razvoja kviz aplikacija i navedeni su primjeri. Dane su teorijske podloge korištenih tehnologija, a to su JavaScript biblioteka React, razvojni okvir Electron, baza podataka Firestore te Material UI zaslužen za dizajn aplikacije. Navedena je arhitektura aplikacije, glavne komponente i implementacija funkcionalnosti. Prikazani su rezultati rada u vidu korisničkog sučelja i analize performansi aplikacije. Izgled aplikacije je i više nego zadovoljavajuć, a performanse su se pokazale dobrim do vrlo dobrim, ovisno o količini učitanih podataka. Na kraju je prikazan sažetak postignuća, jedinstvene značajke aplikacije, ograničenja trenutne verzije te smjernice za budući razvoj.

Ključne riječi: aplikacija za radnu površinu, kviz, natjecanje, obrazovno-zabavni sadržaj, web tehnologije

ABSTRACT

Application for creating and managing quizzes

This thesis describes the process of developing an application for creating and managing quizzes. The paper describes the current state of quiz application development and provides examples. The theoretical foundations of the technologies used are given, namely the React JavaScript library, the Electron development framework, the Firestore database, and Material UI, which is responsible for the design of the application. The architecture of the application, the main components and the implementation of the functionality are all listed. The results of the work in the form of a user interface and application performance analysis are presented. The appearance of the application is more than satisfactory, and the performance turned out to be good to very good, depending on the amount of fetched data. Finally, a summary of achievements, unique features of the application, limitations of the current version and directions for future development are presented.

Keywords: competition, desktop application, edutainment, quiz, web technologies

ŽIVOTOPIS

Matija Kaučić rođen je 4. siječnja 1998. godine u Zagrebu. Svoje osnovnoškolsko obrazovanje započinje 2004. godine u Osnovnoj školi Davorin Trstenjak u Čađavici. 2006. godine, zajedno s obitelji, seli u Slatinu i tamo nastavlja svoje osnovnoškolsko obrazovanje do 2012. godine u Osnovnoj školi Josip Kozarac. Godine 2012. upisuje Opću gimnaziju u Srednjoj školi Marka Marulića u Slatini koju završava 2016. godine. Iste godine, upisuje se na Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku na sveučilišni preddiplomski studij računarstva. Preddiplomski studij završava 2020. godine i upisuje diplomski studij računarstva, smjer Informacijske i podatkovne znanosti.

.

_____

Potpis autora

PRILOG

GitHub repozitorij: https://github.com/mkaucic/kvizna-aplikacija