Izrada aplikacije za vođenje kviza

Ribarić, Marko

Sveučilište u Rijeci, Fakultet informatike i digitalnih tehnologija

Sveučilišni preddiplomski studij Informatika

Marko Ribarić

Izrada aplikacije za vođenje kviza

Završni rad

Mentor: Doc. dr. sc. Lucia Načinović Prskalo

Rijeka, rujan 2023.

Sadržaj

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

Rijeka, 28.4.2023.

Zadatak za završni rad

Pristupnik: Marko Ribarić

Naziv završnog rada: Izrada aplikacije za vođenje kviza

Naziv završnog rada na engleskom jeziku: Creating an application for quiz management

Sadržaj zadatka: Zadatak završnog rada je izraditi aplikaciju za vođenje kvizova. U radu će se izraditi klijentska i poslužiteljska strana aplikacije te će se opisati sve tehnologije koje su se koristile prilikom izrade aplikacije i sve funkcionalnosti koje aplikacija nudi.

Mentor

Doc. dr. sc. Lucia Načinović Prskalo

Načinović Prskalo

Voditelj za završne radove

Doc. dr. sc. Miran Pobar

Zadatak preuzet: datum 30.6.2023

Marko R.

(potpis pristupnika)

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

1. SAŽETAK

U ovom radu je objašnjeno je korisničko sučelje aplikacije za vođenje kviza te načini na koje korisnici mogu uređivati bazu podataka, pripremiti kviz te voditi kviz.

Također su objašnjene tehnologije *React, React Bootstrap, Node.Js, Express, Socket.IO* i SQLite koje su korištene za izradu aplikacije za vođenje kviza. Uz primjere programskom koda je objašnjeno kako se te tehnologije koriste za izradu aplikacije za vođenje kviza.

Objašnjen je način autentifikacije korisnika s haširanjem lozinka i korištenjem tokena.

Ključne riječi: kviz, React, Node.Js, Express, SQL, SQLite baza podataka, Socket.IO, Bootstrap komponente, autentifikacija

2. UVOD

Kvizovi postoje gotovo jednako dugo kao i samo emitiranje na televizijske ekrane. U ranim fazama, emisije poput „Professor Quiz“ i „Ask-It Basket“ jednostavno su postavljale pitanja članovima publike i davale najboljem igraču 25 dolara. U rasponu prošlog stoljeća, kvizovi su prošli kroz mnogo transformacija u načinu igranja i tema o kojima kviz postavlja pitanja. Neke od najpopularnijih kvizova koje se trenutno prikazuju na televiziji su „Wheel of Fortune“, „The Price Is Right“ te „Jeopardy“ kojim je inspirirana i aplikacija koja je izrađena u okviru završnog rada (Rhinewald 2021). Na televiziji se prikazuju kvizovi koje je u današnje vrijeme moguće osobno igrati sa svojim prijateljima. Aplikacija za vođenje kviza je posebno osmišljena kako bi podržala tu mogućnost.

U ovom radu ću detaljno razmotriti izradu aplikacije za vođenje kviza, te objasniti tehnologije i funkcionalnosti koje su korištene. Objasniti ću tehnologije React-a, React Bootstrap-a, Node.js-a, Express-a, Socket.IO-a i SQLite-a integrirane u proces izrade aplikacije. Osim toga, opisati ću korisničko sučelje, analizirati funkcionalnost njegovih elemenata te objasniti kako korisnik može koristiti aplikaciju za postavljanje i igranje kviza.

Također ću se osvrnuti na bazu podataka koja sadrži teme pitanja, pitanja, korisnike i spremljene ploče svakog korisnika te ću objasniti kako se ta baza podataka uređuje i održava. Opisati ću opcije konfiguracije kviza, proces vođenja kviza i komunikaciju između igrača putem realno-vremenske (engl. real-time) komunikacije putem *web socketa*.

Jedan od važnih aspekata koji ću istražiti je i autentifikacija korisnika. Proučit ću kako se korisnici autentificiraju unutar aplikacije i kako se koriste *tokeni* radi sigurnosti i privatnosti.

Na kraju rada, dati ću svoj zaključak o aplikaciji i važnosti tehnologija koje su korištene.

3. KORIŠTENE TEHNOLOGIJE

Aplikacija je izrađena u JavaScript programskom jeziku korištenjem tehnologija *React i react-bootstrap* za prednji dio aplikacije (engl. frontend), te *Node.js i Express* za poslužiteljski dio aplikacije (engl. backend). Također se koristi i *Socket.Io* za komunikaciju između voditelja kviza i igrača kviza. Kao baza podataka koristi se SQLite.

2.1 React

React je biblioteka za razvoj korisničkog sučelja temeljena na *JavaScriptu* koja se naširoko koristi u web razvoju. *React* su kreirali „Facebook“ i zajednica programera otvorenog koda. *React* je u aplikaciji korišten kako bi prednji dio aplikacije bio što ljepše i jednostavnije izrađen te kako bi se omogućilo brzo i efikasno upravljanje stanjem korisničkog sučelja. Jedna od ključnih karakteristika *Reacta* je reaktivno ažuriranje korisničkog sučelja. Kada se promijene podaci ili stanje u aplikaciji, *React* ažurira samo potrebne dijelove korisničkog sučelja, što doprinosi boljim perfomansama. *React* dijeli korisničko sučelje na izolirane dijelove koda višekratne upotrebe poznate kao komponente (Jordana 2023). *React* pruža mali broj funkcija, preciznije kuka (engl. hooks) koje mu omogućuju praćenje vrijednosti u komponentama i održavanje sinkronizacije stanja i korisničkog sučelja. Za pojedinačne vrijednosti možemo koristiti *useState* kuku. *React* također pruža *useEffect* kuku kako bismo mogli bolje kontrolirati nuspojave i integrirati ih u životni ciklus komponenti. Kuka *useEffect* omogućuje sinkronizaciju komponente s vanjskim sustavom te pruža mogućnost da odredimo kada će se neka komponenta osvježiti (Larsen 2021). U aplikaciji se neke komponente osvježavaju prilikom učitavanja stranice, a neke kada se promijeni vrijednost neke varijable.

2.2. React Bootstrap

Bootstrap je jedan od najpopularnijih *CSS* okvira. To je kolekcija *CSS*, *HTML* i *JavaScript* alata otvorenog koda koja programerima pomaže smanjiti vrijeme kodiranja pri izradi web stranica i web stranica. Ključna značajka Bootstrap okvira je responzivnost, tj. aplikacija se automatski prilagođava veličini ekrana. *React-Bootstrap* preuzima CSS okvir Bootstrapa i zamjenjuje bilo koji postojeći *JavaScript* sa striktno *React* komponentama. Zbog toga je jQuery[1] izbačen jer *React Bootstrap* direktno koristi *react* (Pluralsight 2022). Neke od

[1] JavaScript biblioteka koja programerima omogućuje brzo pisanje JavaScripta sažimanjem nekoliko zadataka u jednu liniju koda. (Pluralsight 2022)

React Bootstrap komponenati koje su korištene u aplikaciji su *Button, Nav, NavDropdown, Table* i *Form.*

2.3. Node.Js i Express

Node.Js je višeplatformsko okruženje za izvršavanje[2] koje se koristi za izvršavanje JavaScript koda (Semah 2023). Express JS je okvir baziran na Node.js-u dizajniran za brzu izgradnju Web API-jeva na više platformi i olakšavanje korištenja Node.js-a pomažući u upravljanju poslužiteljima i rutama. Aplikacija koristi Express kako bi stvorila poslužitelja, za srednji sloj (engl. middleware) i obradu zahtjeva, upravljanje rutama i obradu zahtjeva, interakcijom s bazom podataka i integracijom s Socket.Io bibliotekom. Neke od funkcija Express-a koje su korištene u aplikaciji su *app.use, app.post* i *app.delete.*

2.4. Socket.IO

Socket.IO je JavaScript biblioteka za web aplikacije u stvarnom vremenu koja omogućuje dvosmjernu komunikaciju u stvarnom vremenu između web klijenata i poslužitelja pomoću web socket[3] veza. Ima biblioteku na strani klijenta koja se izvodi u pregledniku i biblioteku na strani poslužitelja za node.js (Rai 2013). Socket.Io korišten je u aplikaciji kako bi svi igrači u jednoj sobi mogli vidjeti isto stanje igre. Korisnik se spaja na poslužiteljsku stranu aplikacije kako bi uspostavio dvosmjerni komunikacijski kanal pomoću web socketa u stvarnom vremenu. Ova veza korisniku omogućuje interakciju s drugim povezanim korisnicima i primanje trenutnih ažuriranja bez potrebe za stalnim ručnim osvježavanjem ili ponovljenim zahtjevima. *Socket.IO* sobe su značajka koja omogućuje organiziranje povezanih klijenata u zasebne komunikacijske grupe unutar jedne instance *Socket.IO* poslužitelja (socket.io n.d.).

2.5. SQLite

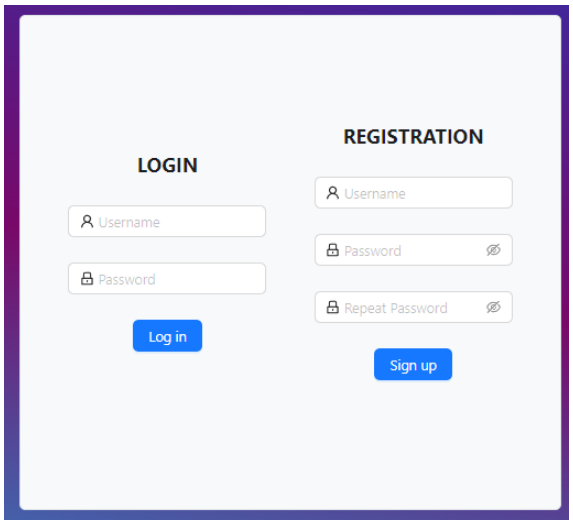
SQLite je ugrađeni sustav za upravljanje relacijskom bazom podataka bez poslužitelja (nije mu potreban drugačiji poslužiteljski proces ili sustav za rad). SQLite je softver otvorenog koda. Baza podataka se nalazi u datoteci u poslužiteljskom dijelu koda (Ravikiran 2023). SQLite nam omogućuje da imamo bazu podataka u koju možemo upisati korisnike, pitanja, odgovore i sve potrebne podatke za aplikaciju. SQLite koristi SQL za upravljanje i manipuliranje relacijskim bazama podataka te je iz tog razloga pristup SQLite bazi podataka

[2] Izvršna okruženja (skraćeno RTE) djeluju kao mali operativni sustavi i pružaju sve funkcije potrebne za izvođenje programa. (Ionos 2020)
[3] WebSocket je dvosmjeran, full-duplex protokol koji se koristi u scenariju komunikacije klijent-poslužitelj (Asati 2023)

jednostavan sa SQL upitima. SQL upiti (engl. Structured Query Language queries) su komande ili naredbe koje se koriste za interakciju s bazom podataka. Ovi upiti omogućavaju komunikaciju s bazom podataka, izvršavajući različite operacije kao što su čitanje, unos, ažuriranje i brisanje podataka. Neki od osnovnih SQL upita uključuju SELECT (za čitanje podataka), INSERT (za unos podataka), UPDATE (za ažuriranje podataka) i DELETE (za brisanje podataka) (aws.Amazon 2022).

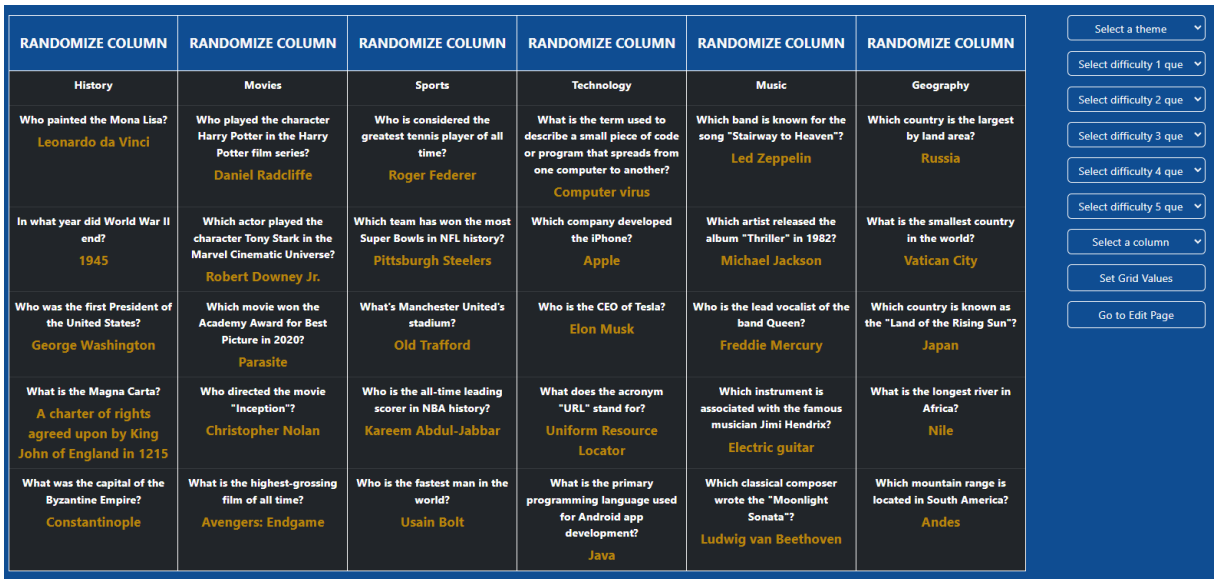
4. KORISNIČKO SUČELJE APLIKACIJE

Pri ulasku u aplikaciju korisniku se prikazuje zaslon za autentifikaciju korisnika na kojem su prikazane opcije za prijavu i registraciju korisnika. Prikaz opcije za prijavu i registraciju korisnika se nalazi na slici 1.



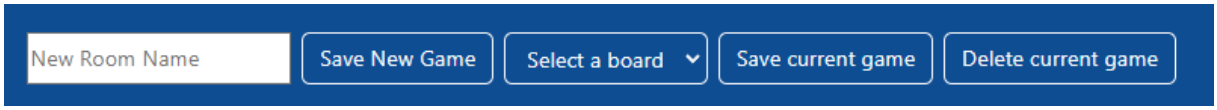
Slika 1 Prikaz korisničkog sučelja za registriranje i prijavljivanje

Nakon autentifikacije, prikazana je stranica konfiguracije kviza s mogućnošći dohvaćanja nasumične teme i pet pitanja klikom na „Randomize column“ i mogućnošću ručnog biranja teme klikom na gumb „Select a theme“. Prikaz konfiguracije ploče se nalazi na slici 2.

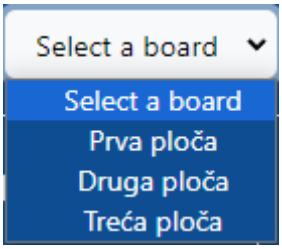


Slika 2 Prikaz korisničkog sučelja za konfiguraciju ploče

Također, na vrhu stranice se nalazi izbornik koji pruža mogućnost spremanja ploče, mogućnost odabira ploče koju želimo koristiti te opcije za brisanje ploče koja je trenutno prikazana. Te mogućnosti su prikazane na slici 3 i slici 4.



Slika 3 Prikaz korisničkog sučelja za opcije spremanja ploče, odabira ploče i brisanja ploče



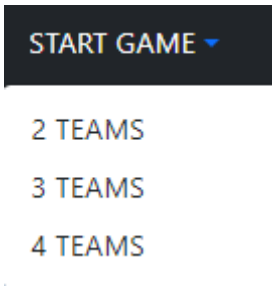
Slika 4 Prikaz korisničkog sučelja za padajući izbornik odabira ploče

Aplikacija također pruža korisniku mogućnost dodavanja vlastitih pitanja te pregled svih pitanja klikom na gumb „Go to Edit Page“ kao što je prikazano na slici 5.

Slika 5 Prikaz korisničkog sučelja za uređivanje baze podataka

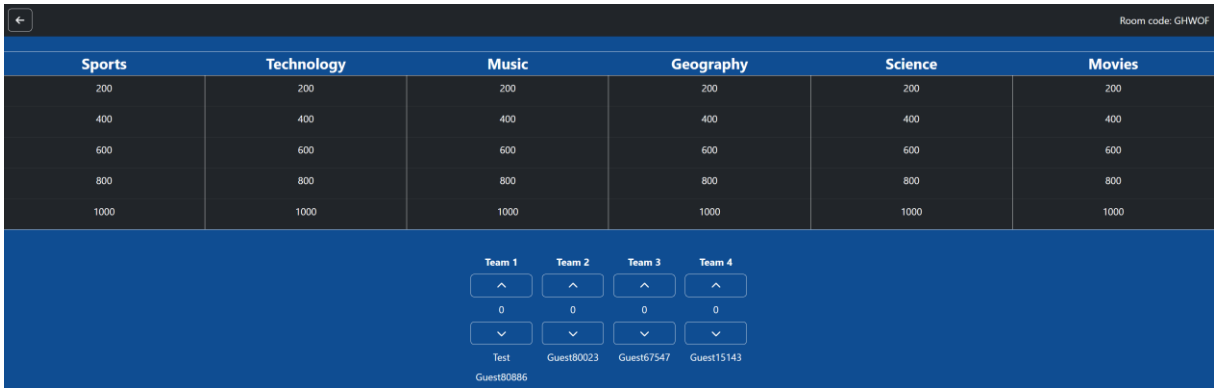
Na stranici za uređivanje baze podataka korisnik može dodati više pitanja u isto vrijeme te izbrisati pitanja koja smo nadodali u slučaju da ih ne želimo više imati u bazi podataka.

Nakon pripreme ploče, korisnik može započeti igru klikom na gumb „Start game“ nakon čega se odabire broj timova koji će igrati igru kao što je prikazano na slici 6.



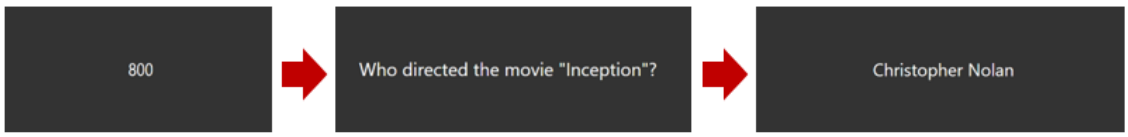
Slika 6 Prikaz korisničkog sučelja za gumb "Start game"

Kad započnemo kviz, ostali igrači kviza mogu ući u kviz iz svoje konfiguracije koristeći kod koji se nalazi u gornjem desnom kutu ili mogu ući s web poveznicom te im se u tom slučaju dodjeljuje nasumično ime. Prikaz primjera igre sa igračima u određenim timovima je prikazan na slici 7.



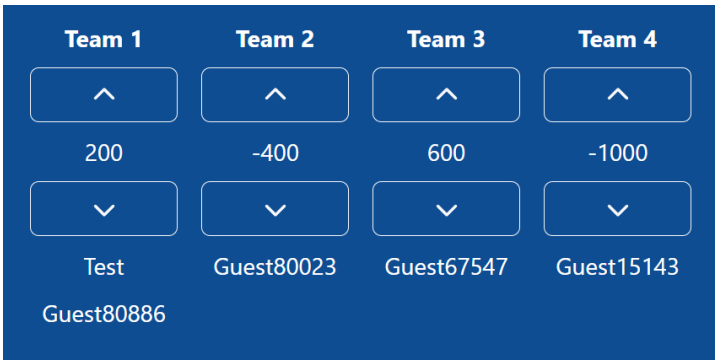
Slika 7 Prikaz korisničkog sučelja vođenja kviza

Klikom na jedno od polja, prikazuje se broj bodova koje nosi pojedino pitanje, ponovljenim klikom se prikazuje pitanje, te se još jednim klikom prikazuje odgovor. Prikaz postepenog prikazivanja pitanja i odgovora za 800 bodova teme filmova je prikazan na slici 8.



Slika 8 Prikaz korisničkog sučelja postepenog prikazivanja pitanja i odgovora

Tim koji točno odgovori na pitanje, dobiva vrijednost bodova ovisno o polju na koji je odgovoren. Na primjer, u slučaju da se odgovara na pitanje teme „Science“ za 400, tim koji točno odgovori na pitanje će dobiti 400 bodova. Pobjednik je tim koji skupi najviše bodova. Primjer prikupljenih bodova timova je prikazan na sljedećoj slici.

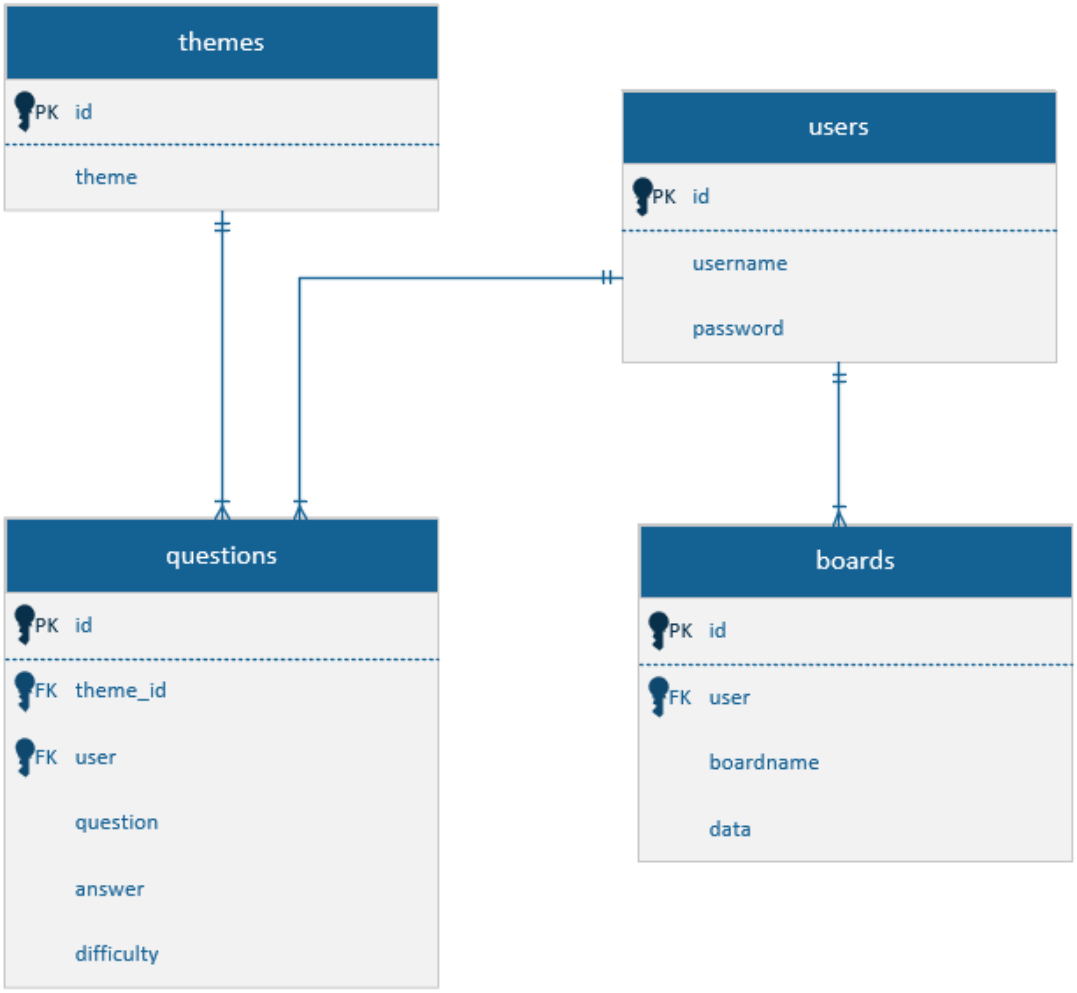


Slika 9 Prikaz timova sa bodovima

U slučaju da je na primjeru sa slike 9. kviz završen, pobjedio bi tim 3 sa 600 bodova.

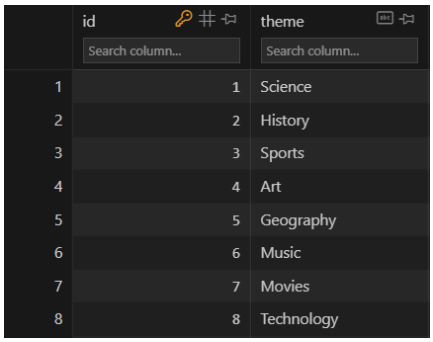
5. BAZA PODATAKA

Baza podataka aplikacije koristi sustav SQLite te se sastoji od 4 tablice pod imenom „themes“, „users“, „questions“ and „boards“. Dijagram baze podataka, napravljen prema notaciji "Crow's Foot notation" (Dybka 2016), prikazan je na slici 10.



Slika 10 Dijagram baze podataka (Crow's Foot notation)

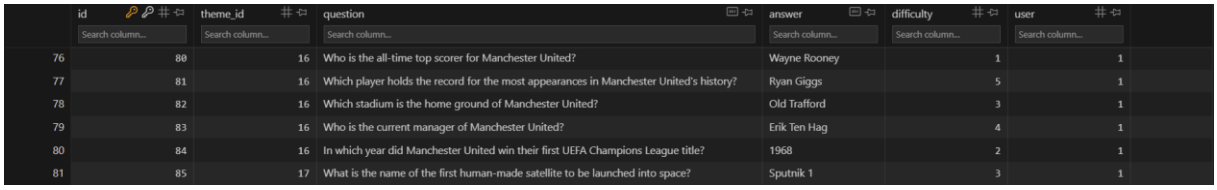
Tablica "themes" je osnova baze podataka koja organizira pitanja u teme koje se mogu koristiti prilikom vođenja kviza. Ova se tablica sastoji od dva atributa: "ID" i "themename". "ID" služi kao primarni ključ pomoću kojeg se pitanja povezuju s temom dok se "themename" koristi kao ime za temu. Primjer zapisa u tablici „themes“ je prikazan na slici 11.



	id	theme
	Search column...	Search column...
1	1	Science
2	2	History
3	3	Sports
4	4	Art
5	5	Geography
6	6	Music
7	7	Movies
8	8	Technology

Slika 11 Prikaz tablice "themes" u bazi podataka

Tablica "questions" ima ulogu spremanja pitanja u bazu podataka. Sadrži atribute "id", "theme_id", "question", "answer" i "difficulty". "ID" služi kao primarni ključ, jedinstveno identificirajući svaki unos pitanja. Atribut "theme_id" uspostavlja vanjski (engl. foreign key) ključ na tablicu "themes", povezujući svako pitanje sa svojom odgovarajućom temom. Takav odnos omogućuje jednostavno pronalaženje pitanja na temelju tema. Vanjski ključ na tablicu „theme“ je bitan kod funkcije dobivanja 5 nasumičnih pitanja određene teme i dobivanja svih pitanja određene teme. Atribut "pitanje" pohranjuje tekstualni prikaz samog pitanja, dok atribut "odgovor" pohranjuje odgovarajući odgovor kako bi se omogućilo prikazivanje prvog pitanja, pa onda odgovora dok traje kviz. Atribut "difficulty" kvantificira razinu složenosti svakog pitanja, pomažući u sortiranju i filtriranju pitanja kako bi se u slučaju poziva nasumične teme s pitanjima dobilo pitanja različitih složenosti. Atribut "user" je vanjski ključ na tablicu „users“ te se koristi za povezivanje pitanja s određenim korisnikom kako bi korisnik mogao imati svoja prilagođena pitanja. U slučaju kad atribut ima vrijednost „NULL“, pitanja mogu vidjeti svi korisnici čime se omogućuje da svi imaju osnovnih 8 tema koje mogu koristiti. Primjer tablice „questions“ je prikazana na sljedećoj slici.



	id	theme_id	question	answer	difficulty	user
	Search column...	Search column...	Search column...	Search column...	Search column...	Search column...
76	80	16	Who is the all-time top scorer for Manchester United?	Wayne Rooney	1	1
77	81	16	Which player holds the record for the most appearances in Manchester United's history?	Ryan Giggs	5	1
78	82	16	Which stadium is the home ground of Manchester United?	Old Trafford	3	1
79	83	16	Who is the current manager of Manchester United?	Erik Ten Hag	4	1
80	84	16	In which year did Manchester United win their first UEFA Champions League title?	1968	2	1
81	85	17	What is the name of the first human-made satellite to be launched into space?	Sputnik 1	3	1

Slika 12 Prikaz tablice "questions" u bazi podataka

Na slici 12 je prikazano šest pitanja sa svojim vlastitim id-ovima. Svih šest pitanja imaju vanjski ključ na id prvog korisnika. Prvih pet pitanja prikazanih na slici povezano je s vanjskim ključem na temu koja ima id 16, a šesta tema ima theme_id 17.

Tablica "users" se koristi za spremanje korisnika u bazu podataka. Tablica sadrži atribute "Id", "username", "password". "Id" funkcionira kao primarni ključ, jamčeći jedinstvenost

svakog korisnika unutar sustava. Korisnička imena i lozinke pridonose procesima provjere autentičnosti, osiguravajući da samo ovlaštene osobe mogu komunicirati sa sustavom.

	id	username	password
	Search column...	Search column...	Search column...
1	1162734	Marko	$2a$10$1uPpHQRuN...
2	1559317	Test1235	$2a$10$Wqt6OomDJ...
3	1834579	Test	$2a$10$izFmiJpo81y...

Slika 13 Prikaz tablice "users" u bazi podataka

Na slici 13 se može vidjeti nasumično generiran id, odgovarajuća korisnička imena i lozinke koje su haširane.

Posljednja tablica koja se nalazi u bazi podataka je tablica "boards". Tablica „boards“ sadrži odabrane spremljene ploče koje određeni korisnik može koristiti. Atribut "user" uspostavlja odnos vanjskog ključa s tablicom "users" te povezuje svaku ploču s korisnikom koji ju je izradio. Ova veza korisnicima omogućuje odabir skupova pitanja prilagođenih njihovim preferencijama te mogućnost korištenja ploče bilo kada u budućnosti. Atribut "boardname" služi za označavanje ploče sa imenom. Atribut ne može biti prazan te se koristi kako bi korisnik mogao spremiti svoje ploče po svojem ukusu. Atribut "podaci" prilagođava se pohranjivanju informacija specifičnih za ploču u obliku pitanja i odgovora. Atribut se sprema u JSON formatu. Primjer tablice nalazi se na sljedećoj slici.

	id	user	boardname	data
	Search column...	Search column...	Search column...	Search column...
1	2	2	QuickTest	[[["Movies"],["Who played the character Harry Potter in the Harry Potter film series?","Daniel Radcliffe"],["Which actor played the character Tony Stark in the M...

Slika 14 Prikaz tablice "boards" u bazi podataka

Na slici je prikazana jedna spremljena ploča nazvana „QuickTest“ korisnika koji ima id vrijednost „2“.

6. UREĐIVANJE BAZE PODATAKA

Uređivanje baze podataka se prikazuje klikom na gumb „Go to Edit Page“ gdje je moguće vidjeti sva pitanja koja su dostupna korisniku koji pristupa aplikaciji te dodavanje i brisanje pitanja u bazi podataka.

5.1. Prikaz svih pitanja

Sva moguća pitanja se dobivaju pozivom funkcije „fetchData“ koristeći „fetch“ API u poslužiteljskom dijelu aplikacije. Funkcija „fetchData“ se pozove jednom kada se komponenta montira (engl. mount) ili svaki put kada kliknemo na „refresh“ gumb. Aplikacija to postiže koristeći „useEffect“ hook koji je dio API-ja Reactove funkcionalnosti. U zaglavlju zahtjeva šaljemo token kako bi nas server identificirao i poslao samo pitanja kojima imamo pristup. U sljedećem tekstnom okviru je prikazana funkcija „fetchData“

```
function fetchData() {
    fetch('http://localhost:5000/api/all', {
      headers: {
        Authorization: `Bearer ${props.token}`
      }
    })
      .then((response) => response.json())
      .then((data) => {
        #ODGOVOR
      })
  }
```

Na server strani po primitku GET zahtjeva, dohvaća se token iz zaglavlja te dobiva korisnički ID iz dekodiranog tokena. Sljedeći korak je postavljanje upita bazi podataka kako bi se dohvatili podaci iz tablica teme i pitanja. Dohvaćeni podaci o temi pohranjeni su u polje „response.themes“ kako bi se kasnije mogla jednostavnije prikazati tablica (prikazuje se ime teme umjesto id-a theme). Upit za pitanja dohvaća retke gdje korisnički ID pitanja odgovara ID-u korisnika ili je korisnički ID pitanja NULL. Dohvaćeni podaci o pitanjima pohranjeni su u polje „response.questions“. Za kraj, server aplikaciji šalje JSON odgovor koji sadrži

dohvaćene podatke. U sljedećem tekstnom okviru je prikazan kod upita postavljen bazi podataka za dohvaćanje svih tema.

```
db.all('SELECT * FROM themes', (err, themeRows) => {
      if (err) {
        console.error(err.message);
        res.status(500).json({ error: 'Internal server error' });
        return;
      }
```

Nakon dohvaćanja svih tema, dohvaćamo sva pitanja gdje je „user = userId“ ili „user = NULL“. Kod je prikazan u sljedećem tekstnom okviru.

```
SELECT * FROM questions WHERE user = ? OR user IS NULL
```

Teme i pitanja su spremljeni u „response“ te je „response“ poslan kao JSON nazad klijentu kao što je prikazano u sljedećem tekstnom okviru.

```
response.themes = themeRows;
response.questions = questionRows;
res.json(response);
```

Odgovor fetch API-ja spremamo u varijablu stanja koristeći Reactov hook useState. Varijabla stanja inicijalizirana je s objektom koji sadrži teme i pitanja. Koristeći „setData“ ažurira se stanje s novim vrijednostima. U sljedećem tekstnom okviru se nalazi dio funkcije „fetchData“ i varijabla stanja „data“.

```
const [data, setData] = useState({ themes: [], questions: [] });

function fetchData() {

       …

      .then((response) => response.json())

      .then((data) => {

        console.log(data);

        setData(data);

      })

       …

  }
```

Nakon toga je moguće ispisivanje sadržaja baze podataka na ekran koristeći React Bootstrap komponentu „Table“. Svaki redak odgovara unosu pitanja i prikazuje ime teme, tekst pitanja, odgovor i složenost pitanja. Posljednji stupac sadrži gumb "Delete", koji se prikazuje samo ako korisnički ID povezan sa stavkom odgovara vrijednosti *props.id*. Klikom na gumb "Delete" poziva se funkcija „handleDeleteItem“ za uklanjanje odgovarajuće stavke iz baze podataka. U sljedećem tekstnom okviru je prikazano tijelo „Table“ komponente.

```
<tbody>

        {data.questions.toReversed().map((item, rowIndex) => (

          <tr key={item.id}>

            <td>{data.themes[item.theme_id - 1].theme}</td>

            <td>{item.question}</td>

            <td>{item.answer}</td>

            <td>{item.difficulty}</td>

            {item.user === props.id ? (

            <td>

            <button className='btn btn-outline-danger' onClick={() =>
handleDeleteItem(item.id)}>Delete</button>

            </td>

            ) : (

              <td></td>

            )}

          </tr>

        ))}

</tbody>
```

5.2. Brisanje iz baze podataka

Funkcija „handleDeleteItem“ šalje id pitanja koji je odabran u poslužiteljski dio aplikacije koristeći „fetch“ API. Poslužitelj šalje SQL upit za brisanje odgovarajućeg pitanja iz tablice „questions“. SQL zahtjev je prikazan u sljedećem tekstnom okviru.

```
DELETE FROM questions WHERE id = ? AND user = ?
```

U SQL zahtjevu stavljamo i uvjet „user = ?“ kao još jednu razinu sigurnosti u slučaju pokušaja SQL injection-a[4]. Prilikom klika na „Delete“ gumb, na ekranu nam se prikaže skočni prozor kako bi se izbjeglo slučajno brisanje pitanja iz baze podataka. Nakon što poslužitelj pošalje potvrdu da je brisanje pitanja iz baze podataka bilo uspješno, aplikacija poziva prethodno spomenutu funkciju „fetchData“ kako bi se prikazana tablica osvježila. U sljedećem tekstnom okviru se prikazuje kod gdje se poziva funkcija „fetchData“.

```
const handleDeleteItem = (itemId) => {

    const confirmed = window.confirm("Are you sure you want to delete this
item?");

    if (!confirmed) {

      return;

    }

    fetch(`http://localhost:5000/api/deleteItem/${itemId}`, {

      method: "DELETE",

    })

      .then((response) => response.json())

      .then((data) => {        fetchData();

      })

  };
```

5.3. Dodavanje u bazu podataka

Teme, pitanja i odgovori se spremaju u varijable stanja kako bi omogućio pristup kada korisnik klikne gumb „Add to database“. Varijable stanja se za početak i kod svakog unosa postavlja na početno stanje. U sljedećem tekstnom okviru su prikazane varijable stanje i postavljanje na početno stanje.

[4] SQL injection (SQLi) sigurnosna je ranjivost na webu koja napadaču omogućuje miješanje u upite koje aplikacija postavlja svojoj bazi podataka (PortSwigger n.d.).

```
const [themeEntry, setThemeEntry] = useState("");
  const [qaEntries, setQAEntries] = useState([
    { question: "", answer: "", difficulty: 1},
    { question: "", answer: "", difficulty: 2},
    { question: "", answer: "", difficulty: 3},
    { question: "", answer: "", difficulty: 4},
    { question: "", answer: "", difficulty: 5}
  ]);
```

Kada se tekst u React komponenti „TextEntry“ promijeni, poziva se funkcija „handleQAEntryChange“ ili „handleThemeEntryChange“ ovisno o tome upisujemo li temu ili pitanja. Funkcije postavljaju trenutno stanje unosa teksta u varijable stanja. Odabir složenosti pitanja se obrađuje korištenjem html komponente „select“ koji nam pruža gumb s mogućnošću više odabira (složenosti od 1-5). U sljedećem tekstnom okviru je prikazan kod gdje se koriste komponente „TextEntry“ i „select“.

```
const handleThemeEntryChange = (value) => {
    setThemeEntry(value); #Postavljamo varijablu stanja na vrijednost teme
  };
<TextEntry value={themeEntry} onChange={handleThemeEntryChange} />
<TextEntry value={entry.answer}onChange={(value) => handleQAEntryChange(index,
"answer", value)}/>
<TextEntry value={entry.question} onChange={(value) =>
handleQAEntryChange(index, "question", value)}/>
<select value={entry.difficulty} onChange={(event) => handleDifficultyChange
(index, event.target.value)}>
                <option value={1}>1</option>
                <option value={2}>2</option>
                <option value={3}>3</option>
                <option value={4}>4</option>
                <option value={5}>5</option>
</select>
```

Kada korisnik klikne na gumb „Add to database“, poziva se funkcija „handleAddToDatabase“ koja u varijablu “dataToAdd“ sprema podatke koje korisnik želi poslati te šalje vrijednost varijable poslužiteljskoj strani aplikacije s „fetch“ API-jem. U varijablu „dataToAdd“ se spremaju samo unosi pitanja i odgovora koji nisu prazni (mogućnost

dodavanja samo jednog ili nekoliko pitanja u isto vrijeme). U sljedećem tekstnom okviru se nalazi programski kod koji se pokrene kada korisnik pritisne na gumb „Add to database“.

```
const handleAddToDatabase = () => {
    const entriesToAdd = qaEntries.filter(
      (entry) => entry.question !== "" && entry.answer !== "");
    const dataToAdd = {
      theme: themeEntry,
      entries: entriesToAdd,
    };
    fetch("http://localhost:5000/api/addToDatabase", {
      method: "POST",
      body: JSON.stringify(dataToAdd),
    })
};
```

U poslužiteljskom dijelu aplikacije se provjerava postoji li već dobivena tema u bazi podataka. Šalje se SQL upit bazi podataka te u slučaju da tema ne postoji („!row“), izrađuje se nova tema unosom u tablicu „questions“. U slučaju da tema postoji, poziva se SQL upit „INSERT“ sa varijablom „row.id“ (id koji smo dobili kad smo provjeravali nalazi li se tema u bazi podataka). U slučaju da tema ne postoji u bazi podataka, poziva se SQL upit „INSERT“ koda dodaje sadržaj u tablicu „themes“, te SQL upit „INSERT“ sa varijablom „this.lastID“ kako bi nadodali pitanje sa zadnjom temom koju smo dodali. Programski kod ovih koraka se nalazi u sljedećem tekstnom okviru.

```
db.get('SELECT id FROM themes WHERE theme = ?', [theme], (err, row) => {
       if (!row) {
         db.run('INSERT INTO themes (theme) VALUES (?)', [theme], function(err) {
         insertQuestions(this.lastID, entries, userId);
         });}
       else {
         insertQuestions(row.id, entries, userId);}
       });
```

7. KONFIGURACIJA KVIZA

Konfiguracija kviza omogućuje korisnicima odabir tema, pitanja i ploča za igranje kviza. Komponenta komunicira s poslužiteljem kako bi dohvatila teme, pitanja i podatke o ploči, a korisnici mogu postaviti ploču za igru dodjeljivanjem pitanja i tema pojedinačnim stupcima. Također se mogu spremati, ažurirati ili brisati konfiguracije igre ovisno o njihovoj želji.

6.1 Prikaz tema i pitanja

Priprema tema i pitanja za kviz funkcionira na način spremanja odabranih tema i pitanja u varijablu stanja „boardData“ koja sadrži informacije za svaku ćeliju u tablici koju aplikacija prikazuje na ekranu. Varijabla „boardData“ je inicijalizirana kao niz dimenzija 6x6 te svaki element u nizu predstavlja ćeliju na ploči za igru. Inicijalizacija varijable stanja „boardData“ je prikazana u sljedećem tekstnom okviru.

```
const [boardData, setBoardData] = useState(Array.from(Array(6), () => Array(6).fill("")));
```

Prilikom korisničkog namještanja ploče, aplikacija izvršava promjene nad varijabli stanja „boardData“ sa „setBoardData“ funkcijom koristeći klasu koja nam omogućuje pristup pitanju i odgovoru za jednostavnije korištenje. Klasa koja se koristi prikazana je u sljedećem tekstnom okviru.

```
export default class GridValues {

    constructor(theme, question1, answer1, question2, answer2, question3, answer3,
question4, answer4, question5, answer5) {

      return [

        [this.theme],

        [this.question1, this.answer1],

        [this.question2, this.answer2],

        [this.question3, this.answer3],

        [this.question4, this.answer4],

        [this.question5, this.answer5]

      ];

    }

  }
```

Tablica se prikazuje na ekranu koristeći Bootstrap komponentu „Table“ koja u prvom redu tablice sadrži nazive tema, a u svim drugim ćelijama određeno pitanje i odgovor. Kod Bootstrap komponente „Table“ prikazan je u sljedećem tekstnom okviru.

```
<Table>
        <tbody>
        {Array.from(Array(6)).map((_, rowIndex) => (
        <tr itemScope="col" key={rowIndex}>
                {Array.from(Array(6)).map((_, columnIndex) => (
                <td key={`${rowIndex}-${columnIndex}`} className="theme-name">
                <div className="question">{boardData[columnIndex][rowIndex][0]}</div>
                <div className="answer">{boardData[columnIndex][rowIndex][1]}</div>
                </td>))}</tr>))}
        </tbody>
</Table>
```

6.2 Promjena tema i pitanja

Teme i pitanja se mogu mijenjati na dva različita načina. Prvi način je koristeći „Randomize column“ gumb koji se nalazi u zaglavlju tablice (tablica u prošlom tekstnom okviru). Zaglavlje tablice je prikazano u sljedećem tekstnom okviru.

```
<thead>      <tr>
       {Array.from(Array(6)).map((_, buttonIndex) => {
        return <button key={buttonIndex} onClick={() => GetRandomColumnQuestion
       (buttonIndex)}>{`RANDOMIZE COLUMN`}</button>}
</tr>  </thead>
```

Korisničkim klikom na gumb se poziva funkcija „GetRandomColumnQuestion“ koja sprema podatke koji se nalaze u „boardData“ varijabli stanja u varijablu „FirstColumnValues“. Koristeći „fetch“, API aplikacija šalje varijablu „FirstColumnValues“ u poslužiteljski dio aplikacije kako bi poslužiteljski dio aplikacije vratio nasumičnu temu koja se trenutno ne nalazi u tablici. Spremanje u varijablu „FirstColumnValues“ i slanje u poslužiteljski dio aplikacije uz pomoć „JSON.stringify“[5] funkcije je prikazano u sljedećem tekstnom okviru.

```
const firstColumnValues = boardData.map(row => row[0]);
fetch(`http://localhost:5000/randomcolumn?themes=${JSON.stringify(firstColumnValues)}`{…}
```

[5] JSON.stringify() pretvara JavaScript vrijednost u JSON niz (Mozilla n.d.)

U poslužiteljskom dijelu aplikacije, dohvaćaju se poslane teme koristeći funkciju „JSON.parse“ i spremanjem podataka u varijablu „gottenthemes“. Aplikacija šalje SQL upit koji je prikazan u sljedećem tekstnom okviru u bazu podataka.

```
SELECT t.id, t.theme
      FROM themes t
      JOIN questions q ON t.id = q.theme_id
      WHERE t.theme NOT IN (${themeNames})
      AND (q.user = ? OR q.user IS NULL)
      GROUP BY t.id, t.theme
      HAVING COUNT(DISTINCT q.difficulty) >= 5
      ORDER BY RANDOM()
      LIMIT 1
```

SQL upit započinje odabirom stupca „id“ i „theme“ iz tablice „themes“ koje ćemo koristiti. Koristimo klauzulu "JOIN", gdje se formira veza između tablice "themes" i tablice "questions". Povezivanje se uspostavlja na temelju uvjeta da stupac "id" tablice "themes" odgovara stupcu "theme_id" tablice "questions". Ova veza omogućuje dobivanje informacija o temama uz povezane podatke o pitanjima. Koristeći mehanizam filtriranja s klauzulom „WHERE“ isključujemo teme koje su prisutne u varijabli „gottenthemes“ kako poslužiteljski dio ne bi vratio temu koja se već nalazi u tablici. Klauzula "HAVING" izvodi filtraciju nakon grupiranja, zadržavajući samo one teme koje ispunjavaju specificirani uvjet. U ovom slučaju, teme moraju imati broj različitih vrijednosti složenosti povezanih s njihovim pitanjima jednak ili veći od pet kako bismo dobili pet pitanja različitih težina. Za kraj upita, primjenjuje se klauzula "LIMIT" kako bi se konačni izlaz ograničio na jednu temu. Postavljanjem ograničenja na jedan, upit osigurava da je samo jedna nasumično odabrana tema prikazana u konačnom skupu rezultata.

Drugi način na koji se mogu mijenjati teme i pitanja je korištenjem padajućih izbornika. Padajući izbornici nam pružaju mogućnost manualnog odabira tema i pitanja korisnicima kako bi se omogućio prilagođeni odabir. Padajući izbornici su napravljeni korištenjem html komponente *select*. Ova komponenta omogućava korisniku da odabere sve teme za koje postoji barem jedno pitanje, kao i sva pitanja odabrane teme koja su određene težine. Korisnik prvo odabire temu koju želi uključiti u tablicu. Aplikacija omogućuje ovaj odabir koristeći internu varijablu stanja nazvanu *themes*, koja se popunjava putem *fetch* API poziva prema poslužiteljskoj strani aplikacije. Ovaj API poziv se aktivira prilikom prvog učitavanja stranice. Poslužiteljska strana aplikacija izvodi SQL upit bazi podataka te vraća sve teme koje imaju

barem jedno pitanje koje je dostupno korisniku koji šalje upit. SQL upit je prikazan u sljedećem tekstnom okviru.

```
SELECT t.id, t.theme
      FROM themes t
      JOIN questions q ON t.id = q.theme_id
      WHERE q.user = ? OR q.user IS NULL
      GROUP BY t.id, t.theme
      HAVING COUNT(q.id) > 0 #Korisnik mora imati barem jedno dostupno pitanje
```

Nakon odgovora poslužiteljske strane aplikacije, teme se postavljaju kao opcija html komponente *select*. Html komponente *select* je prikazana u sljedećem tekstnom okviru.

```
<select onChange={handleThemeChange}>
       <option value="">Select a theme</option>
       {themes.map(theme => (
       <option key={theme.id} value={theme.id}>{theme.theme}</option>))}
</select>
```

Svakom korisničkim promjenom padajućeg izbora, poziva se funkcija *handleThemeChange* koja dohvaća pitanja vezana uz odabranu temu pomoću *fetch* API poziva poslužiteljskom dijelu aplikacije. U *fetch* API poziv pošalje se ID teme koju je korisnik odabrao te poslužitelj izvodi SQL upit bazi podataka kako bi dobio sva pitanja odabrane teme. Nakon dobivanja svih pitanja od poslužiteljske strane aplikacije, aplikacija izrađuje novi objekt pod nazivom *updatedQuestionByDifficulty* kako bi pohranila dobivena pitanja prema njihovoj složenosti. *Questions.forEach* petlja ponavlja se kroz svako pitanje, dohvaća vrijednost složenosti te umeće u odgovarajuće polje unutar *updatedQuestionByDifficulty* varijable stanja. Kod ovog postupka se nalazi u sljedećem tekstnom okviru.

```
const updatedQuestionsByDifficulty = {1: [], 2: [], 3: [], 4: [], 5: []};
questions.forEach(question => {
       const { difficulty } = question;
       updatedQuestionsByDifficulty[difficulty].push(question);
       });
setQuestionsByDifficulty(updatedQuestionsByDifficulty);
```

U sljedećem se koraku ažurira varijabla stanja *questionsByDifficulty* kako bi aplikacija mogla popuniti pet padajućih izbornika s odgovarajućim pitanjima. Aplikacija koristi *map*[6] funkciju kako bi iterirala kroz svih pet nizova u *questionsByDifficulty* stanju te ponovno koristi *map* funkciju kako bi iterirala kroz svako pitanje u tim nizovima. *map* funkcije su prikazane u sljedećem tekstnom okviru.

```
{Array.from(Array(5)).map((_, index) => (
     <select key={index} onChange={event => handleQuestionChange(event, index)}>
              {questionsByDifficulty[index + 1]?.map(question => (
              <option key={question.id} value={JSON.stringify(question)}>
                   {question.question}
              </option>))}
     </select>))}
```

Zadnji korak je odabir stupca kojeg korisnik želi promijeniti. To čini pomoću padajućeg izbornika koji ima opcije od jedan do šest. Nakon odabira svih padajućih izbornika, pritiskom na gumb „Set Grid Values“, mijenja se odabrani stupac u tablici kao što je prikazano u sljedećem tekstnom okviru.

```
const newGridValues = new GridValues(
      themes.find(theme => theme.id === parseInt(selectedThemeId))?.theme,
      ...selectedQuestions.flatMap(q => [q.question, q.answer])
); # Stvara se nova instanca GridValues.
setBoardData(prevGridValues => {
      const updatedGridValues = [...prevGridValues];
      updatedGridValues[selectedColumn - 1] = [...newGridValues.toArray()];
      return updatedGridValues;
}); # Sprema se instanca newGridValues u BoardData
```

6.3 Upravljanje pločama

Aplikacija također pruža mogućnost spremanja ploča za svakog korisnika. Ploče se spremaju u tablicu *boards* u bazi podataka. Trenutna ploča se može spremiti na dva načina. Prvi način je spremanje u novu instancu u *boards* tablici. U novu instancu se sprema ispunjavanjem *input* komponente i klikom na gumb „Save New Game“. Gumb poziva funkciju *handleSaveNewGame*. Funkcija šalje *fetch* API poslužiteljskom dijelu aplikacije kako bi se

[6] Metoda map() stvara polje pozivanjem određene funkcije na svakom elementu prisutnom u nadređenom polju. (Singh n.d.)

nova ploča zapisala u bazu podataka. Na poslužiteljskom dijelu aplikacije, izvodi se SQL upit bazi podataka. Kao što je prethodno spomenuto, u *boards* tablici se nalaze *user, boardname, data* i *id.* U slučaju da se ne upiše *id* prilikom SQL upita, generira se novi *id*. SQL upit se nalazi u sljedećem tekstnom okviru.

```
INSERT INTO boards (user, boardname, data) VALUES (?, ?, ?)', [userId,
name, JSON.stringify(gridValues)
```

Drugi način spremanja ploče je spremanje u već postojeću instancu u *boards* tablici. Korisnik na stranici konfiguracije ima opciju biranja između svih njegovih ploča u padajućem izborniku. Kada korisnik odabere jednu od opcija u padajućem izborniku, u varijablu stanja *selectedBoard* se sprema ID polja kojeg je korisnik odabrao. Prilikom svake promjene *selectedBoard* varijable stanja, poziva se *fetch* API kako bi aplikacija zamijenila podatke trenutne ploče sa odabranom pločom. Korisnik sada može mijenjati ploču koju je odabrao, te se pritiskom na gumb „Save current game“ poziva funkcija *handleSaveGame*. Funkcija *handleSaveGame* šalje *fetch* API poslužiteljskoj strani aplikacije. Poslužiteljska strana aplikacije šalje SQL upit bazi podataka kako bi ažurirala instancu koja ima ID ploče koja je trenutno odabrana. SQL upit je prikazan u sljedećem tekstnom okviru.

```
UPDATE boards SET user = ?, boardname = ?, data = ? WHERE id = ?',
[userId, name, JSON.stringify(gridValues), id]
```

Još jedna opcija koju aplikacija pruža korisniku je brisanje ploča. Brisanje ploča funkcionira na sličan način kao ažuriranje ploče, samo što se šalje *DELETE* metoda u *fetch* API-ju. Poslužiteljska strana aplikacija izvodi SQL upit bazi podataka kako bi obrisala trenutno odabranu ploču. SQL upit se nalazi u sljedećem tekstnom okviru.

```
DELETE FROM boards WHERE user = ? AND id = ?', [userId, parseInt(boardId)]
```

Nakon brisanja ploče, *selectedBoard* varijabla stanja se postavlja na *null* vrijednost.

8. VOĐENJE KVIZA

Za početak igre, korisnik može na konfiguracijskoj stranici pritisnuti na padajući izbornik „Start Game“ te dobiti opciju s koliko timova želi započeti sobu. Za kreiranje sobe, aplikacija koristi *Socket.IO* biblioteku. Kad korisnik započne igru, aplikacija kreira sobu za korisnika u instanci *Socket.IO* klase generiranjem nasumične šifre od pet slova i emitira poslužiteljskoj strani koda da kreira sobu. Kod koji služi za emitiranje poslužiteljske strane aplikacije se nalazi u sljedećem tekstnom okviru.

```
    const createNewRoom = (numberteams) => {
      if(selectedBoard != ""){
        const roomCode = generateRandomString();
        socket.emit('CreateRoom', {
            roomCode: roomCode,
            BoardID: selectedBoard.id,
            numberofteams: numberteams
        });
        navigate("/play/"+roomCode);
      }
  }
```

Aplikacija na poslužiteljskoj strani ulazi u sobu *Socket.IO* biblioteke čiji je naziv prethodno generirana šifra iz prijašnjeg koraka. Ulaskom u sobu u kojoj se niti jedan drugi korisnik ne nalazi, izrađuje se nova soba. Koristeći funkcije *console.log(io.sockets.adapter.rooms)* možemo prikazati sve trenutno aktivne sobe. Primjer ispisa funkcije se nalazi u sljedećem tekstnom okviru.

```
Map(4) {
  'C4iIYcgBz9QSPdvFAAAD' => Set(1) { 'C4iIYcgBz9QSPdvFAAAD' },
  'HAEEM' => Set(1) { 'C4iIYcgBz9QSPdvFAAAD' },
  'fvIYBD999KDN2mBRAAAH' => Set(1) { 'fvIYBD999KDN2mBRAAAH' },
  'CUZPR' => Set(1) { 'fvIYBD999KDN2mBRAAAH' }
}
```

Prva i treća soba se automatski generiraju kada se neki korisnik spoji sa *socket.io* web socketom, dok se druga i četvrta soba generiraju kada korisnik stvori novu sobu u kojoj će voditi kviz. Nakon što korisnik kreira sobu, aplikacija vodi korisnika na određenu rutu

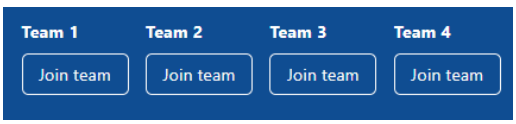
/play/{roomcode} koristeći navigaciju. Ova radnja se poduzima kako bi se prikazala stranica *playGrid* i omogućilo drugim korisnicima da se pridruže sobi. Drugi korisnici se mogu pridružiti sobi ispunjavanjem *Bootstrap* komponente *Form* sa šifrom sobe kojoj se žele pridružiti te pritiskom na „JOIN ROOM“ gumb. Na poslužiteljskoj strani aplikacije se prilikom kreacije sobe izrađuje instanca klase *RoomInfo* s podacima sobe. Prikaz klase *RoomInfo* se nalazi u sljedećem tekstnom okviru.

```
class RoomInfo {
  constructor(name, teams, scores, answeredQuestion, themes) {
    this.name = name;
    this.teams = teams;
    this.scores = scores;
    this.answeredQuestion = answeredQuestion;
    this.themes = themes;
}
```

U klasi se nalaze ime sobe (kod koji se generira pri ulasku u sobu), polja u kojima se nalaze timovi, polja u kojima se nalaze bodovi timova, pitanja koja su prethodno otkrivena te teme koje se nalaze u kvizu. Razlog zašto upisujemo teme, bodove timova te sudionike timova u instancu klase je omogućavanje praćenja posljednjeg stanja igre za igrače koji naknadno uđu u igru. Prilikom ulaska korisnika u igru, emitiraju se „Teams“, „updateScores“, „answeredQuestions“ i updateThemes varijable stanja kako bi svi sudionici na ekranu imali sinkroniziranu igru.

```
io.to(data.roomCode).emit('Teams', roomInfo[socket.roomCode].teams);
io.to(data.roomCode).emit('updateScores', roomInfo[socket.roomCode].scores)
io.to(data.roomCode).emit('answeredQuestions',roomInfo[socket.roomCode].answered
Question)
io.to(data.roomCode).emit('updateThemes', roomInfo[socket.roomCode].themes);
```

Prilikom ulaska u sobu, korisnik može odabrati tim u kojem želi igrati. Korisnik odabire tim pritiskom na jedan od gumbova ispod imena timova. Korisničko sučelje odabira je prikazano na slici 15.



Slika 15 Prikaz gumbova i opcija ulaska u tim

Pritiskom na jedan od gumbova, aplikacija emitira *joinTeam* s *index* i *username* varijablama poslužiteljskoj strani aplikacije. Poslužiteljska strana aplikacije dodaje igrača u tim koji je odabrao, te emitira svima u sobi tu promjenu. Kod poslužiteljske strane aplikacije se nalazi u sljedećem tekstnom okviru.

```
socket.on('joinTeam', (teamIndex, userName) => {
    roomInfo[socket.roomCode].teams[teamIndex].push(userName); #Dodavanje igrača u tim
    io.to(socket.roomCode).emit('Teams', roomInfo[socket.roomCode].teams); #Emitiranje
});
```

Svi korisnici na svojoj "playGrid" stranici osluškuju emitiranja. U slučaju emitiranja, svoju varijablu stanja *teams* promjene na trenutačno stanje timova. Kod osluškivanja emitiranja se nalazi u sljedećem tekstnom okviru.

```
socket.on('Teams', (gottenteams) => {
      setTeams(prevTeams => gottenteams ?? []);
    });
```

Nakon što aplikacija promijeni *teams* varijablu stanja, korisnik može na ekranu vidjeti sve igrače svakog tima kao što je prikazano na slici 16.

Slika 16 Prikaz igrača u timova

Kada korisnik kreira sobu, koristi se ploča koja je trenutno odabrana. Dok traje igra, samo korisnik može kontrolirati ploču i dodavati bodove timova. To se omogućuje na način da prilikom kreiranja sobe voditelj postavlja varijablu *roomCreator* na *true* te svaki pokušaj emitiranja promjena tablice od drugih korisnika neće biti moguć zbog jednostavne *if* funkcije.

Kviz sada može započeti. Klikom na neko od polja u tablici voditelj emitira pitanje koje se čita izravno iz baze podataka funkcijom *getQuestionAnswer*. U *getQuestionAnswer* funkciju šaljemo varijable *q, row, column,* i *socket.BoardID*. Kada voditelj kreira novu sobu i pokrene igru, *socket.BoardID* postavlja se na ID odabrane ploče za igru te se može koristiti u budućim navratima. Varijabla *q* ovisi o tome želi li aplikacija dobiti pitanje ili odgovor iz baze podataka. Funkcija *getQuestionAnswer* izvodi SQL upit bazi podataka kako bi dobila pitanje ili odgovor. Razlog zašto uzimamo jedno po jedno pitanje i odgovor iz baze podataka je dodatna razina

sigurnosti (sprječava se mogućnost da korisnik iz podataka o zahtjevu na server koji su dostupni u web pregledniku otkrije ostala pitanja i/ili odgovore). Pritiskom na jedno polje, ispred tablice prikazuje se *html* element s vrijednosti pitanja (čije prikazivanje kontrolira varijabla stanja *overlay1Visible*). Ponovnim klikom se ispred tablice prikazuje *html* element s tekstom pitanja (čije prikazivanje kontrolira varijabla stanja *overlay2Visible*). Trećim i posljednjim klikom se ispred tablice prikazuje *html* element s tekstom odgovora na pitanje (čije prikazivanje kontrolira varijabla stanja *overlay3Visible*). Odgovori i pitanje se prilikom dobivanja od poslužiteljske strane aplikacije spremaju u *currentShownValue* varijablu stanja. Prikaz koda za prikazivanje vrijednosti pitanja, pitanja i odgovora je prikazan u sljedećem tekstnom okviru.

```
socket.on('displayOverlay', (rowIndex, columnIndex) => {
      setClickedCell(prevState => ({ row: rowIndex, column: columnIndex }));
      setDisableQuestion(dc => [...dc, rowIndex+"_"+columnIndex]);
      setOverlay1Visible(true);});
socket.on('displayOverlay2', (question) => {
      setcurrentShownValue(question);
      setOverlay1Visible(false);
      setOverlay2Visible(true);});
socket.on('displayOverlay3', (answer) => {
      setcurrentShownValue(answer);
      setOverlay2Visible(false);
      setOverlay3Visible(true);});
socket.on('displayOverlay4', () => {
      setOverlay3Visible(false);});

#HTML KOD U NASTAVKU
{overlay1Visible && ( <div className="overlay"
onClick={handleOverlay1Click}>{values[clickedCell.row]}</div>)}
{overlay2Visible && ( <div className="overlay"
onClick={handleOverlay2Click}>{currentShownValue}</div>)}
{overlay3Visible && (div className="overlay"
onClick={handleOverlay3Click}>{currentShownValue}</div>)}
```

Posljednji dio vođenja kviza je dodavanje bodova timu koji odgovori točno na pitanje te oduzimanje bodova timu koji netočno odgovori na pitanje. U slučaju točnog odgovora na pitanje, uzima se vrijednost polja pitanja na koje smo kliknuli. Vrijednost šaljemo u poslužiteljski dio aplikacije kako bismo dodali bodove odgovarajućem timu u instanci *roomInfo* klase za našu igru. Prikaz slanja iz prednjeg dijela aplikacija u poslužiteljski dio aplikacije je prikazan na sljedećem tekstnom okviru.

```
function addPoints(teamIndex, q){
    if(q==0){
      socket.emit('adjustScore', lastSelectedScore.current, teamIndex);
    } #U SLUČAJU TOČNOG ODGOVORA
    else{
      socket.emit('adjustScore', -lastSelectedScore.current, teamIndex);
    } #U SLUČAJU KRIVOG ODGOVORA
}
```

U poslužiteljskom dijelu aplikacije, broj bodova se dodaje timu koji je točno odgovorio na pitanje te se emitira kako bi svi igrači imali sinkronizirani broj bodova. Prikaz tog procesa je prikazan na sljedećem tekstnom okviru.

```
socket.on('adjustScore', (score, teamIndex) => {
 roomInfo[socket.roomCode].scores[teamIndex] += score; #Dodavanje bodova
 io.to(socket.roomCode).emit('updateScores', roomInfo[socket.roomCode].scores); #Emitiranje
});
```

9. AUTENTIFIKACIJA KORISNIKA

Za autentifikaciju korisnika, u bazi podataka nam je potrebna tablica s *id*, *username*, i *password* stupcima. Kreiranjem novog korisnika poziva se „registerUser“ funkcija iz server.js datoteke. Generira se nasumičan sedmeroznamenkasti broj koji će se koristiti kao ID korisnika. Prikaz generiranja ID-a se nalazi u sljedećem tekstnom okviru.

```
function generateRandomId() {
    return Math.floor(1000000 + Math.random() * 9000000);
  }
```

Sljedeći korak je haširanje lozinke. Haširanje lozinke definira se kao provođenje lozinke kroz algoritam raspršivanja kako bi se čisti tekst pretvorio u nerazumljiv niz brojeva i slova. Ovo je važno za osnovnu sigurnost jer u slučaju proboja sigurnosti, neovlašteni korisnik ne može razumjeti lozinke. Zbog toga je krađa ovih podataka znatno teža. Za haširanje lozinka mogu se koristi različite funkcije kao što su *bcrypt* i *SHA* (Jung 2021). U sljedećem tekstnom okviru je prikazan dio koda aplikacije gdje se koristi bcrypt haširanje te u tablici umeće redak sa generiranim id-em, korisničkim imenom i haširanim passwordom.

```
bcrypt.hash(password, 10, (hashErr, hashedPassword) => { #HAŠIRANJE
          db.run('INSERT INTO users (id, username, password) VALUES (?,
          ?, ?)', [userId, username, hashedPassword], (insertErr) => {
            if (insertErr) {
              console.error(insertErr);
              res.status(500).send({ error: 'Internal Server Error' });
              return;
            }
            res.send({ success: true });
          });
        });
```

Prilikom prijavljivanja se iz baze podataka dohvaća haširani password te uspoređuje sa lozinkom koju je korisnik unio. Kod usporedbe se nalazi u sljedećem tekstnom okviru.

```
 bcrypt.compare(password, storedPassword, (compareErr, isMatch) => {
        if (compareErr) {
          console.error(compareErr);
          res.status(500).send({ validation: false, error: 'Internal
Server Error' });
          return;
        }
        if (isMatch) {
         # NASTAVAK
        } else {
          res.send({ validation: false });
        }
      });
```

Za korištenje aplikacije koristimo mehanizam tokena za provjeru autentičnosti. Token za provjeru autentičnosti sigurno prenosi informacije o identitetima korisnika između aplikacija i web stranice. Token omogućuje korisnicima interneta pristup aplikacijama, uslugama, web stranicama i sučeljima za programiranje aplikacija bez potrebe za unosom vjerodajnica za prijavu prilikom svake posjete. Umjesto unosa vjerodajnica prilikom svake posjete, korisnik se prijavljuje jednom, a jedinstveni token se generira i dijeli s povezanim aplikacijama ili web stranicama kako bi se potvrdio njihov identitet (Magnusson 2023).

Tokeni se generiraju u nekoliko koraka. Prvi korak je korisnička molba za pristup poslužitelju koja se tipično dešava prilikom prijave s lozinkom. Poslužitelj utvrđuje da osoba treba imati pristup te poslužitelj komunicira s uređajem za autentifikaciju, nakon čega poslužiteljska strana aplikacije izdaje token i prosljeđuje ga korisniku. Token se nalazi unutar korisničkog preglednika te ako korisnik pokuša posjetiti drugi dio poslužitelja, token ponovno komunicira s poslužiteljem. Pristup se odobrava ili odbija na temelju tokena. (Okta 2023)

Aplikacija za vođenje kviza koristi tokene pri svakom zahtjevu na serveru kako bi se utvrdilo jeli korisnik koji šalje zahtjev legitiman. Aplikacija koristi „JSON Web Token“ za kreiranje tokena. JSON Web Token (JWT) je otvoreni standard (RFC 7519) koji definira kompaktan i samostalan način za siguran prijenos informacija između strana kao JSON objekt (JWT n.d.).

U sljedećem tekstnom okviru se nalazi funkcija za generiranje tokena. Token ističe za 24 sata te tada postaje nevažeći kao što je prikazano u sljedećem tekstnom okviru.

```
function generateToken(userId) {
  return jwt.sign({ userId: userId }, secretKey, { expiresIn: '24h' });
}
```

Kad korisnik želi napraviti neku akciju (u ovom primjeru nabaviti stupac nasumične teme i pitanja), šalje token u backend uz pomoć „bearer“ dijela headera što je prikazano u sljedećem tekstnom okviru.

```
fetch(`http://localhost:5000/randomcolumn?themes=${JSON.stringify(firstColumnVal
ues)}`, {
      headers: {
        Authorization: `Bearer ${token}` ### SLANJE TOKENA
      }
    })
```

Kad server dobije token, provjerava ga s tajnim ključem kako bi autorizirao korisnika. Provjera je prikazana u sljedećem tekstnom okviru.

```
app.get("/randomcolumn", (req, res) => {
  const token = req.headers.authorization;
  const tokenWithoutBearer = token.replace("Bearer ", ""); # Dobivanje tokena iz
                                                                    headera
  jwt.verify(tokenWithoutBearer, secretKey, (err, decoded) => {
    if (err) {
      console.error('Error verifying token:', err);
      res.status(401).json({ error: 'Unauthorized' });
      return;
    }
```

10. ZAKLJUČAK

U ovom radu predstavljena je aplikacija za izradu i igranje kviza. Objašnjen je način korištenja aplikacije kako bi se mogla uređivati baza podataka, konfigurirati kviz te voditi kviz. Također su objašnjene tehnike i tehnologije *React, React Bootstrap, Node.Js, Express, Socket.Io* i *Sqlite* koje su korištene prilikom izrade aplikacije. Tehnologije su objašnjenje kroz primjere koje se koriste u aplikaciji te su opisani razlozi zašto su se koristili prilikom izrade aplikacije.

Aplikacija se u trenutnom stanju može koristiti, ali postoji puno različitih načina na koje može biti unaprijeđena kao na primjer dodavanje opcije da svaki tim nakon pitanja može podnijeti odgovor, dodavanje više opcija za konfiguraciju ploča i prebacivanje podatke u bazu podataka s boljim performansama.

Bibliografija

Ably. *Socket.IO: What it is, when to use it, and how to get started.* 5 2022. https://ably.com/topic/socketio.

Asati, Arpit. *Geeksforgeeks - What is web socket and how it is different from the HTTP?* 6 2023. https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/.

aws.Amazon. *What Is SQL (Structured Query Language)?* 2022. https://aws.amazon.com/what-is/sql/.

Dybka, Patrycja. *Vertabelo - Crow's Foot Notation.* 31. 3 2016. https://vertabelo.com/blog/crow-s-foot-notation/.

Ionos. *Runtime environments: explanation and examples.* 12 2020. https://www.ionos.com/digitalguide/websites/web-development/what-is-a-runtime-environment/.

Jordana, A. *Hostinger.* 7 2023. https://www.hostinger.com/tutorials/what-is-react.

Jung, Jason. *What are Salted Passwords and Password Hashing?* 7. 5 2021. https://www.okta.com/blog/2019/03/what-are-salted-passwords-and-password-hashing/.

JWT. *JWT.* n.d. https://jwt.io/introduction (pokušaj pristupa 7 2023).

Larsen, John. *React Hooks In Action.* Manning, 2021.

Magnusson, Andrew. *Token-based Authentication: Everything You Need to Know.* 19. 7 2023. https://www.strongdm.com/blog/token-based-authentication.

Mozilla, Developer. *Developer Mozilla.* n.d. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify (pokušaj pristupa 7 2023).

Okta. *What Is Token-Based Authentication?* 14. 2 2023. https://www.okta.com/identity-101/what-is-token-based-authentication/.

Pluralsight. *Understanding React-Bootstrap and Making It Work for You.* 11 2022. https://www.pluralsight.com/blog/software-development/react-bootstrap-explained.

PortSwigger. *SQL injection.* n.d. https://portswigger.net/web-security/sql-injection (pokušaj pristupa 6 2023).

Rai, Rohit. *Socket.IO Real-time Web Application Development.* Birmingham: Packt Publishing Ltd., 2013.

Ravikiran. *simplelearn.* 16. 2 2023. https://www.simplilearn.com/tutorials/sql-tutorial/what-is-sqlite.

Rhinewald, Shane. *A Brief History Of Game Shows.* 2. 6 2021. https://www.museumofplay.org/press-release/a-brief-history-of-game-shows/.

Semah, Benjamin. *FreecodeCamp - What Exactly is Node.js?* 8 2023. https://www.freecodecamp.org/news/what-is-node-js/.

Singh, Pankaj. *Geeksforgeeks.* n.d. https://www.geeksforgeeks.org/javascript-array-map-method/ (pokušaj pristupa 6 2023).

socket.io. *Socket.io.* n.d. https://socket.io/docs/v3/rooms/ (pokušaj pristupa 6 2023).

Popis slika