

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

Stručni prijediplomski studij Računarstvo

Izrada kviz aplikacije upotrebom Material Design 3
paketa

Završni rad

Matej Mijok

Osijek, 2024.

Obrazac Z1S: Obrazac za ocjenu završnog rada na stručnom prijediplomskom studiju

Ocjena završnog rada na stručnom prijediplomskom studiju

Ime i prezime pristupnika:	Matej Mijok
Studij, smjer:	Stručni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	AR4866, 27.07.2021.
JMBAG:	0165090855
Mentor:	prof. dr. sc. Krešimir Nenadić
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 1:	prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 2:	Robert Šojo, univ. mag. ing. comp.
Naslov završnog rada:	Izrada web kviza upotrebom Material Design 3 paketa
Znanstvena grana završnog rada:	Programsko inženjerstvo (zn. polje računarstvo)
Zadatak završnog rada:	Kratko opisati zahtjeve i funkcionalnosti koje bi trebala imati web aplikacija za rješavanje kvizova iz različitih područja. Navesti nekoliko sličnih postojećih rješenja te napraviti usporedbu s izrađenom web aplikacijom. Projektirati i izraditi bazu podataka koju će koristiti web aplikacija te opisati postupak izrade. Navesti specifičnosti izrade web aplikacije korištenjem Material Design 3 paketa te kratko usporediti s Material Design 2. Detaljno opisati postupak izrade web aplikacije kao i izrađene funkcionalnosti. Web aplikacija treba podržavati pitanja i kvizove iz više područja. Tema
Datum ocjene pismenog dijela završnog rada od strane mentora:	23.09.2024.
Ocjena pismenog dijela završnog rada od strane mentora:	Izvrstan (5)
Datum obrane završnog rada:	7.10.2024.
Ocjena usmenog dijela završnog rada (obrane):	Izvrstan (5)
Ukupna ocjena završnog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio stručni prijediplomski studij:	07.10.2024.

IZJAVA O IZVORNOSTI RADA

Osijek, 07.10.2024.

Ime i prezime Pristupnika:	Matej Mijok
Studij:	Stručni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	AR4866, 27.07.2021.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom: **Izrada web kviza upotrebom Material Design 3 paketa**

izrađen pod vodstvom mentora prof. dr. sc. Krešimir Nenadić

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	1
1.1. Zadatak završnog rada	1
2. PREGLED POSTOJEĆIH ANDROID I WEB APLIKACIJA	2
2.1. Kahoot	2
2.2. QuizzLand	2
2.3. Britannica	3
2.4. Trivia Crack	4
3. TEORIJSKA POZADINA, PREGLED KORIŠTENIH TEHNOLOGIJA, FUNKCIONALNI I NEFUNKCIONALNI ZAHTJEVI APLIKACIJE	5
3.1. Material Design 3	5
3.1.1. Razlike Material Design 3 i Material Design 2	5
3.2. React.js	6
3.3. PHP	7
3.4. Bootstrap	8
3.5. Funkcionalni zahtjevi za aplikaciju	9
3.6. Nefunkcionalni zahtjevi za aplikaciju	9
3.7. Projektiranje baze podataka	10
4. IZRADA APLIKACIJE	11
4.1. Navigacijska komponenta	11
4.2. Komponenta početne stranice	13
4.3. Komponente skočnih prozora za prijavu i registraciju	15
4.4. Komponenta stranice za početak kviza	21
4.5. Komponenta skočnog prozora za promjenu postavki kviza	23
4.6. Komponenta za igranje kviza	25
4.7. Komponenta stranice korisničkog profila	29
4.8. Komponenta stranice rezultata	30
4.9. Prilagodba komponenti	31

5. IZGLED IZRAĐENE APLIKACIJE.....	32
6. ZAKLJUČAK.....	37
LITERATURA	38
SAŽETAK.....	39
ABSTRACT	40

1. UVOD

U današnjem digitalnom dobu, mobilni uređaji i računala postala su neizostavan dio svakodnevnog života. Aplikacije na raznim uređajima omogućuju svojim korisnicima brojna rješenja koja im olakšavaju svakodnevne zadatke. Kroz razvoj tehnologije, dizajn aplikacija je postao ključan dio pružanja kvalitetnog korisničkog iskustva. Tvrtka Google razvila je skup smjernica kako bi programerima i dizajnerima omogućila lakše stvaranje atraktivnih sučelja i sučelja koja se lako prilagođavaju raznim vrstama uređaja. *Material Design 3* je najnovija verzija ovih smjernica koja znatno povećava prilagodljivost aplikacija u odnosu na dosadašnje komponente. *Material Design 3* posvećuje posebnu pažnju na pristupačnost, čineći sučelja jednostavnijim za korištenje osobama s invaliditetom. Korisnicima se olakšava praćenje promjena na zaslonu putem animacija i raznolikih boja. Jedan od glavnih izazova pri izradi web aplikacija je osiguranje dosljednosti sučelja na različitim uređajima. Sučelje mora biti prilagodljivo različitim veličinama zaslona i pritom zadržati smisleno organiziranu strukturu. Sučelje također mora biti prilagođeno na smislen način kako bi se osigurala pristupačnost što većem broju korisnika. Boje koje se koriste moraju imati dovoljno velik kontrast da elementi sučelja budu raspoznatljivi, ali kontrast ne smije biti prevelik jer tako može uzrokovati neugodu pri korištenju aplikacije. Cilj je stvoriti vizualno privlačno i dosljedno sučelje, ali isto tako olakšati pristup što većem broju korisnika.

Aplikacija izrađena u sklopu završnog rada prvo je uspoređena sa već dostupnim aplikacijama. U sljedećem poglavlju opisane su razlike između *Material Design 3* i *Material Design 2* smjernica, tehnologije upotrijebljene pri izradi aplikacije, funkcionalni i nefunkcionalni zahtjevi, te je prikazana baza podataka. Nakon toga detaljno su opisane komponente koje se koriste unutar aplikacije. Zatim je napravljen pregled konačnog sučelja aplikacije.

1.1. Zadatak završnog rada

Kratko opisati zahtjeve i funkcionalnosti koje bi trebala imati web aplikacija za rješavanje kvizova iz različitih područja. Navesti nekoliko sličnih postojećih rješenja te napraviti usporedbu s izrađenom web aplikacijom. Projektirati i izraditi bazu podataka koju će koristiti web aplikacija te opisati postupak izrade. Navesti specifičnosti izrade web aplikacije korištenjem *Material Design 3* paketa te kratko usporediti s *Material Design 2*. Detaljno opisati postupak izrade web aplikacije kao i izrađene funkcionalnosti. Web aplikacija treba podržavati pitanja i kvizove iz više područja.

2. PREGLED POSTOJEĆIH ANDROID I WEB APLIKACIJA

U ovom poglavlju prikazane su slične i već dostupne Android i web aplikacije kako bi se kroz analizu istaknule njihove funkcionalnosti i izbjegle potencijalne greške prilikom izrade aplikacije.

2.1. Kahoot

Kahoot je web aplikacija za grupno igranje kvizova. Kvizu se može pridružiti upisivanjem zaporce sobe koja se kreira pokretanjem kviza. Svako od pitanja može imati vremensko ograničenje koje postavi autor kviza. Ponudeni odgovori mogu biti u nasumičnom redoslijedu što je važno kod ovakvih aplikacija kako bi aplikacija spriječila pamćenje mjesta točnog odgovora. Slika 2.1 prikazuje izgled Kahoot aplikacije [1]. Za razliku od aplikacije izrađene u sklopu ovog rada, Kahoot podržava više istovremenih korisnika. Na desnom rubu zaslona prikazan je broj koji označava koliko korisnika je odgovorilo na trenutno pitanje. U Kahoot aplikaciji bodovi se dodjeljuju na temelju brzine odgovora, a u aplikaciji izrađenoj u sklopu završnog rada broji se samo je li pitanje točno odgovoreno ili ne.



Sl. 2.1. Prikaz Kahoot aplikacije.

2.2. QuizzLand

QuizzLand je Android aplikacija za igranje kvizova. Aplikacija ima razine od kojih svaka sadrži nekoliko pitanja vezana za interese koje korisnik unese prilikom prvog pokretanja aplikacije.

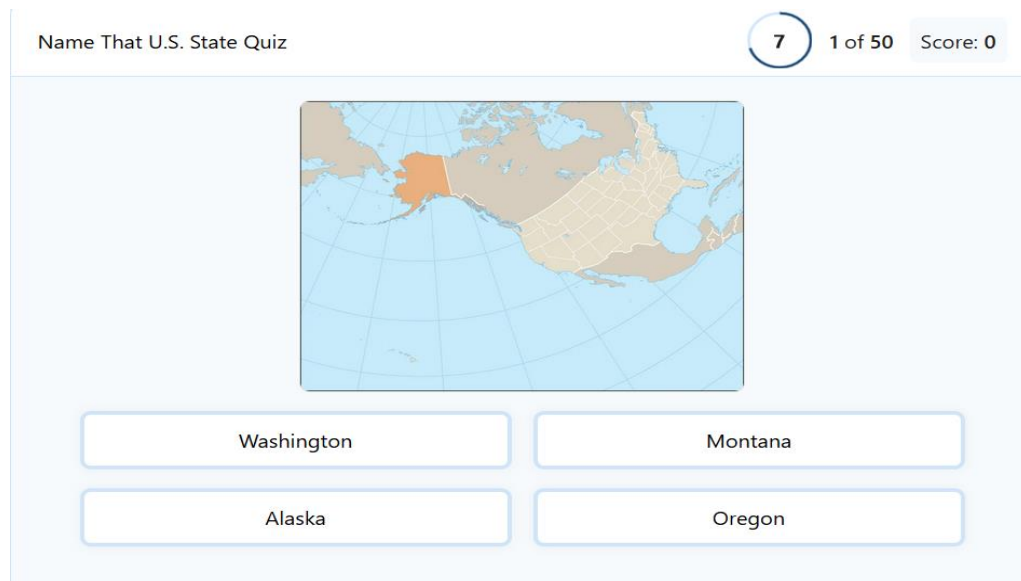
Nakon svakog pitanja aplikacija omogućuje korisniku čitanje više o temi o kojoj odgovara. Za svako od pitanja moguće je ostaviti komentar i vidjeti što su drugi korisnici komentirali. Slika 2.2 prikazuje izgled aplikacije QuizzLand [2]. Aplikacija QuizzLand koristi vrlo slične boje pozadinu i za tipke. Odgovori na postavljeno pitanje mogu biti teško uočljivi nekim korisnicima. Aplikacija QuizzLand omogućuje korisnicima pomoć pri rješavanju kviza što nije podržano u aplikaciji izrađenoj u sklopu završnog rada. Ima opcija za skrivanje dva netočna odgovora, prelazak na drugo pitanje, prikaz točnog odgovora i mogućnost ponovnog odgovora ako korisnik prvi put netočno odgovori na pitanje. QuizzLand je samo mobilna aplikacija i nije dostupna na računalu, a aplikacija izrađena u sklopu ovog rada podržava različite vrste uređaja.



Sl. 2.2. Prikaz QuizzLand aplikacije.

2.3. Britannica

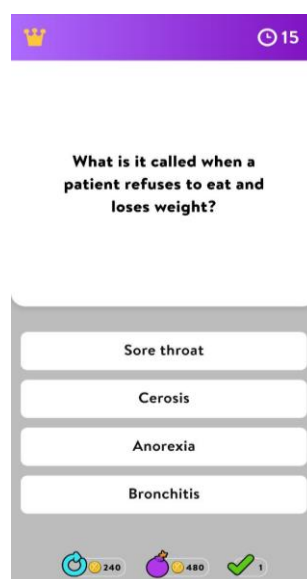
Britannica web aplikacija nudi raznovrsne igre, uključujući kvizove. Za pitanja se može postaviti vremensko ograničenje. Nakon isteka vremena, pitanje se gleda kao neodgovoreno i točan odgovor se označuje sjenčanjem zelenom bojom, te se pojavi tipka koja vodi na slijedeće pitanje. Vremensko ograničenje se može isključiti, ali se korisniku dodjeljuje manje bodova za svako točno odgovoreno pitanje nego s uključenim vremenskim ograničenjem. Na Slici 2.3. prikazana je Britannica web aplikacija [3]. Za razliku od aplikacije izrađene u sklopu ovog rada, nakon odgovora na pitanja, Britannica aplikacija prikazuje korisniku dodatne informacije o postavljenom pitanju. Najčešće je to dodatno objašnjenje o točnom odgovoru.



Sl. 2.3. Prikaz Britannica web aplikacije.

2.4. Trivia Crack

Trivia Crack je aplikacija za igranje kvizova. Podržava igranje više igrača istovremeno, te slanje poveznica drugim igračima koji se žele pridružiti igri. Ostale funkcionalnosti i sam izgled aplikacije su jako slični ranije spomenutoj aplikaciji QuizzLand uz dodatak vremenskog ograničenja za svako od pitanja. Na Slici 2.4 prikazana je aplikacija Trivia Crack [4]. Za razliku od aplikacije izrađene u sklopu ovog rada, aplikacija Trivia Crack korisnika uparuje sa drugom osobom koja istovremeno odgovara na ponuđena pitanja.



Sl. 2.4. Prikaz Trivia Crack aplikacije.

3. TEORIJSKA POZADINA, PREGLED KORIŠTENIH TEHNOLOGIJA, FUNKCIONALNI I NEFUNKCIONALNI ZAHTJEVI APLIKACIJE

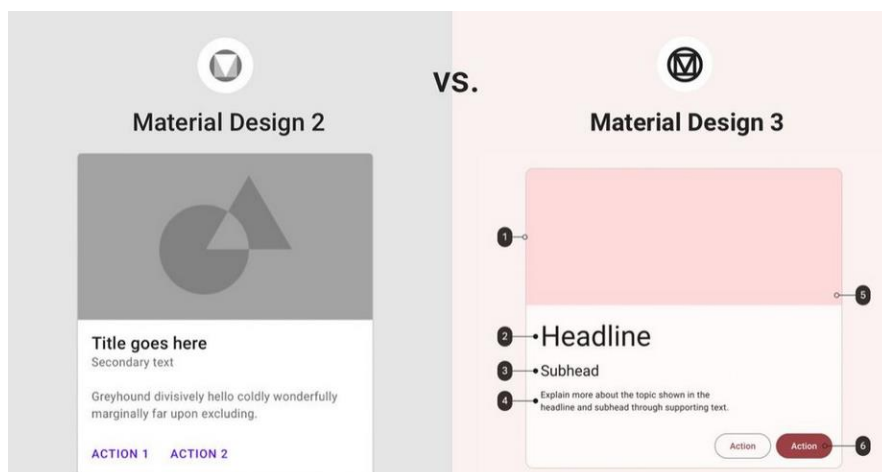
U ovom poglavlju prikazana je teorijska osnova i tehnologije koje su upotrijebljene za izradu aplikacije pomoću Material Design 3 paketa, te funkcionalni i nefunkcionalni zahtjevi aplikacije.

3.1. Material Design 3

Material Design je Googleov sustav za dizajn otvorenog koda. Cilj Material Design sustava je omogućavanje jedinstvenog korisničkog iskustva na različitim uređajima i platformama što od kraja 2023. godine uključuje i web aplikacije, od kada je dostupan Material Web paket. Omogućene su implementacije za Android, Flutter i web. Material Design simulira prave materijale i njihova svojstva kao što su sjene i dubina. Material Design također koristi glatke i intuitivne animacije koje pomažu korisniku vidjeti i razumjeti promjene u korisničkom sučelju. Za Material Design napisana je opširna dokumentacija u kojoj se nalaze upute za korištenje i dobra praksa prilikom izrade mobilnih ili web aplikacija [5].

3.1.1. Razlike Material Design 3 i Material Design 2

Material Design 3 uvodi korištenje dinamičkih boja. Ugrađen je generator palete boja koja može uzeti dominantne boje sa slike pozadine mobilnog uređaja i primijeniti ju na cijelu aplikaciju. Umjesto korištenja dinamičkih boja, moguće je definirati nekoliko osnovnih boja kako bi se osigurao vizualni identitet aplikacije, dok se unutar razvojnog okruženja dinamički generira ostatak palete za različite svrhe, poput boje za prikaz grešaka na komponentama. Rubovi komponenta sučelja su više zaobljeni u odnosu na Material Design 2. Još jedna novost koju je uveo Material Design 3 jest podrška za uređaje velikih zaslona i preklopnih uređaja. Velik utjecaj na konačni dizajn komponenti imala je pristupačnost korisnicima. Na Slici 3.1. prikazane su razlike u dizajnu između Material Design 2 i Material Design 3 [6].



Sl. 3 1. Prikaz razlika između Material Design 2 i Material Design 3.

3.2. React.js

React.js je biblioteka otvorenog koda koju je razvila tvrtka Meta. Korištena je za razvoj sučelja kviz aplikacije. React.js korišten je zbog svog rada s komponentama. React.js omogućuje web aplikaciji izgled jedne stranice na kojoj se izmjenjuje sadržaj. Tijekom izrade aplikacije korišten je JSX (engl. *JavaScript XML*) dodatak za JavaScript koji omogućuje pisanje HTML (engl. *HyperText Markup Language*) unutar JavaScript koda. Za spremanje i upravljanje podataka korišten je *State*. *State* se koristi za dodjeljivanje stanja ili vrijednosti nekoj varijabli unutar same komponente. React.js biblioteka koristi virtualni DOM (engl. *Document Object Model*). Kada dođe do promjene na web stranici, React.js provjeri razlike između novog i starog stanja te korisniku pošalje samo promjene. Na Slici 3.2. prikazana je jednostavna komponenta koja koristi promjenu stanja u polju za upis teksta, te sprema vrijednost koja je upisana u varijablu *tekst* [7].

```

import React, { useState } from 'react';

function TekstualniUnos() {
  const [tekst, setTekst] = useState('');

  const handleInputChange = (event) => {
    setTekst(event.target.value);
  };

  return (
    <div>
      <h1>Unesi tekst:</h1>
      <input
        type="text"
        value={tekst}
        onChange={handleInputChange}
      />
      <p>Uneseni tekst: {tekst}</p>
    </div>
  );
}

```

Sl. 3.2. Prikaz komponente za unos.

3.3. PHP

PHP (engl. PHP: *Hypertext Preprocessor*) je programski jezik otvorenog koda koji se koristi za poslužiteljski dio web aplikacija. Tijekom izrade aplikacije korišten je za komunikaciju sa bazom podataka. Omogućava upis i čitanje podatka unutar baze podataka. Klijentski dio šalje zahtjev sa podacima poslužitelju, te poslužitelj izvrši potrebnu radnju i šalje klijentskom dijelu odgovor o uspješnosti radnje [8]. U aplikaciji se odgovor klijentskoj strani šalje u JSON (engl. JavaScript Object Notation) formatu.

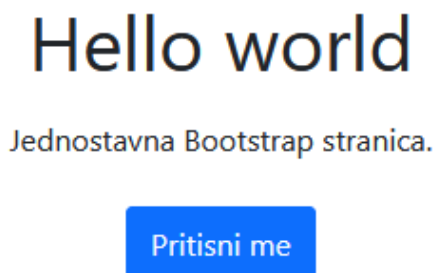
3.4. Bootstrap

Bootstrap je alat koji služi za pojednostavljivanje izrade responzivnog korisničkog sučelja prilikom izrade web stranica i aplikacija. Koristi se kao dodatak HTML elementima u obliku klase. Svakom elementu se može dodati odgovarajuća Bootstrap klasa koja može imati definirano pozicioniranje i izgled elementa. Bootstrap se dodaje svakom HTML dokumentu putem poveznica unutar head elementa. Na Slici 3.3. prikazan je primjer korištenja Bootstrap klase [9].

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap primjer</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet">
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"></script>
</head>
<body>
  <div class="container text-center mt-5">
    <h1 class="mb-3">Hello world</h1>
    <p class="mb-4">Jednostavna Bootstrap stranica.</p>
    <a href="#" class="btn btn-primary">Pritisni me</a>
  </div>
</body>
</html>
```

Sl. 3.3. Prikaz načina korištenja Bootstrapa.

Na Slici 3.4. prikazan je izgled web stranice koja koristi Bootstrap.



Sl. 3.4. Prikaz web stranice koja koristi Bootstrap.

3.5. Funkcionalni zahtjevi za aplikaciju

Funkcionalni zahtjevi odnose se na funkcionalnosti koje sustav treba imati kako bi omogućio svojim korisnicima obavljanje potrebnih zadataka. Funkcionalni zahtjevi opisuju ono što sustav mora raditi.

Funkcionalni zahtjevi koje obuhvaća izrađena aplikacija:

- Registracija i prijava korisnika
- Promjena kategorija pitanja i broja pitanja u kvizu
- Igranje kviza
- Pregled i promjena korisničkih podataka
- Dodavanje novih pitanja i kategorija ako je korisnik administrator
- Promjena naziva kategorija ako je korisnik administrator
- Brisanje kategorije i pitanja ako je korisnik administrator

3.6. Nefunkcionalni zahtjevi za aplikaciju

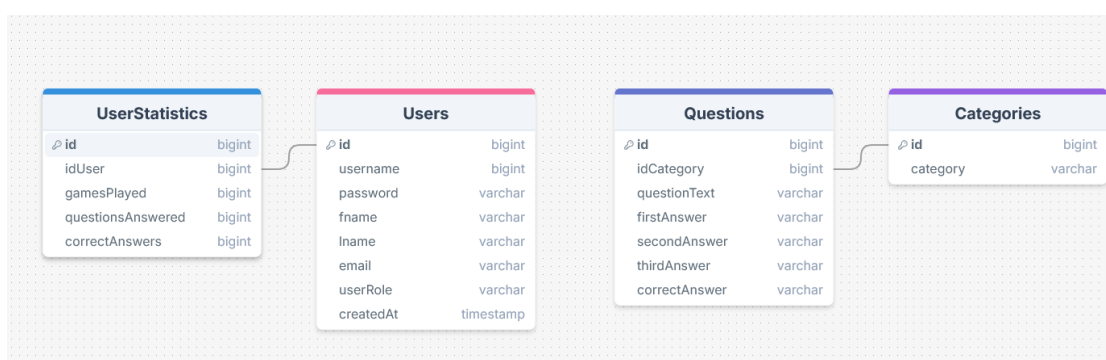
Nefunkcionalni zahtjevi koje obuhvaća izrađena aplikacija:

- Kompatibilnost
- Performanse
- Upotrebljivost
- Održavanje i nadogradnja
- Sigurnost

Kompatibilnost podrazumijeva da aplikacija mora biti kompatibilna sa različitim operacijskim sustavima i web preglednicima. Što se tiče performansi, aplikacija se treba učitavati što kraće. Upotrebljivost znači da aplikacija treba imati prilagodljiv dizajn za različite vrste uređaja, što omogućava dosljedno korisničko iskustvo na različitim uređajima. Održavanje i nadogradnja omogućeni su korištenjem komponenti za različite dijelove sučelja. Što se tiče sigurnosti, zaporka se šalje samo prilikom prijave ili registracije korisnika, te se ona sprema u bazu podataka nakon obrade s PHP-ovom funkcijom *password_hash*. Za provjeru ispravnosti zaporka prilikom prijave koristi se funkcija *password_verify*.

3.7. Projektiranje baze podataka

Na Slici 3.5. prikazan je dijagram baze podataka koja se koristi u aplikaciji kviza. Baza podataka sastoji se od četiri tablice. Tablica *Users* sadrži korisničko ime, zaporku, ime, prezime, e-mail, ulogu korisnika i vremensku oznaku kada je korisnički račun kreiran. Ako je tablica *Users* prazna, prvi kreirani račun je administratorski račun. Tablica *UserStatistics* sadrži strani ključ koji se povezuje sa tablicom *Users* i sadrži broj igranih igara, odgovorenih pitanja i točnih odgovora za računanje statistike korisnika. Tablica *Categories* sadrži ime kategorije. Tablica *Questions* sadrži strani ključ za kategoriju pod kojom se pitanje nalazi, tekst pitanja i odgovore.



Sl. 3.5. Prikaz baze podataka za aplikaciju kviz.

Za bazu podataka koristi se *SQL* (engl. Structured Query Language), a za lokalni poslužitelj koristi se program *XAMPP*. Baza podataka kreira se pokretanjem *PHP* koda. Na Slici 3.6. prikazan je programski kod za jednu od tablica.

```
$sql = 'CREATE TABLE IF NOT EXISTS categories(
    id INT AUTO_INCREMENT,
    category VARCHAR(255) NOT NULL,
    PRIMARY KEY (id)
)';

if(!$conn->query($sql)) {
    echo 'ERROR CREATING TABLE CATEGORIES';
}
```

Sl. 3.6. Prikaz programskog koda za kreiranje tablice.

4. IZRADA APLIKACIJE

U ovom poglavlju prikazan je postupak izrade aplikacije i način korištenja Material Design komponenti. Pojedino potpoglavlje predstavlja jednu od komponenti koje se koriste u aplikaciji.

4.1. Navigacijska komponenta

Na svakoj stranici unutar aplikacije prikazana je gornja navigacijska traka. Na njoj su postavljene tipke koje omogućavaju navigaciju na druge komponente. Na Slici 4.1. prikazan je programski kod funkcije *Navigation*.

```
function Navigation() {
  const navigate = useNavigate();
  const [activeTabIndex, setActiveTabIndex] = useState(0);
  const tabRef = useRef();

  useEffect(() => {
    const handleChange = (event) => {
      setActiveTabIndex(event.target.activeTabIndex);
      switch (event.target.activeTabIndex) {
        case 0:
          navigate('/');
          break;
        case 1:
          navigate('/play');
          break;
        case 2:
          navigate('/profile');
          break;
        default:
          break;
      }
    };

    const currentTabRef = tabRef.current;
    currentTabRef.addEventListener('change', handleChange);

    return () => {
      currentTabRef.removeEventListener('change', handleChange);
    };
  }, [navigate]);
}
```

Sl. 4.1. Prikaz programskog koda funkcije *Navigation*.

Na početku funkcije deklarirane su varijable *navigate*, *activeTabIndex*, *tabRef*, i funkcija *setActiveTabIndex*. Varijabla *navigate* sadrži funkciju *useNavigate()* koja se koristi za navigaciju između stranica. Varijabla *activeTabIndex* sadrži indeks aktivne kartice, odnosno kartice na kojoj se korisnik trenutno nalazi. Pritiskom na neku drugu karticu poziva se *useEffect* kuka (engl. *hook*). Kuka *useEffect* može se izvesti nakon prvog učitavanja stranice ili svaki puta kada se dogodi

promjena u nekoj varijabli. U ovom slučaju `useEffect` kuka izvodi svaki put kada se promijeni stanje varijable `navigate`. Unutar kuke definirana je funkcija `handleChange` koja promijeni vrijednost svaki put kada se promijeni kartica. Kuka postavi stanje varijable `activeTabIndex` na vrijednost koja se dobije od Material Design komponente `md-tabs`. Svaka od kartica ima svoj indeks koji kreće od nula. Ovisno o indeksu korisnik se proslijedi na odgovarajuću stranicu. Nakon prosljeđivanja napravi se varijabla `currentTabRef` koja sadrži referencu na trenutnu karticu. Na trenutnu karticu se zatim postavi slušatelj događaja koji prati ako se dogodi događaj promjene (engl. *change*) i ako se on dogodi poziva se funkcija `handleChange`. *Return* dio kuke izvršava se kada prilikom odspajanja komponente ili kada se kuka ponovno pozove. Unutar *return* dijela se briše slušatelj događaja promjene sa trenutne kartice.

```
return (
  <div className='container-fluid' id='container'>
    <div className='container-fluid'>
      <ul className='navbar-nav'>
        <md-tabs id='tabs' ref={tabRef}>
          <md-secondary-tab id='tabs' aria-controls='home-panel' active><p className='lead' id='text'>Home</p></md-secondary-tab>
          <md-secondary-tab id='tabs' aria-controls='play-panel'><p className='lead' id='text'>Play</p></md-secondary-tab>
          <md-secondary-tab id='tabs' aria-controls='profile-panel'><p className='lead' id='text'>Profile</p></md-secondary-tab>
        </md-tabs>
      </ul>
    </div>
    <Routes>
      <Route path="/play" element={<div className='container-fluid'><PlayPanel/></div>} />
      <Route path="/play/quiz" element={<div className='container-fluid'><Questions/></div>} />
      <Route path="/play/quiz/results" element={<div className='container-fluid'><Results/></div>} />
      <Route path="/profile" element={<div className='container-fluid'><ProfilePanel/></div>} />
      <Route path="/" element={<div className='container-fluid'><HomePanel/></div>} />
    </Routes>
  </div>
);
```

Sl. 4.2. Prikaz programskog koda sučelja gornje navigacijske trake.

Na Slici 4.2. prikazan je programski kod koji definira kartice gornje navigacijske trake. Komponenti `md-tabs` dodijeljen je identifikator koji povezuje kartice koje pripadaju `md-tabs` komponenti sa samom komponentom. Uz identifikator dodjeljuje se i referenca na trenutno aktivnu karticu `tabRef`. Unutar `md-tabs` komponente definirane su `md-secondary-tab` komponente koje predstavljaju kartice na koje je korisnik prosljeđen pritiskom. Nakon definiranja dizajna sučelja napisane su rute za pojedinu stranicu koje pozivaju odgovarajuće komponente. Navigacija unutar aplikacije odvija se korištenjem `react-router-dom` biblioteke za React.js.

4.2. Komponenta početne stranice

Prva stranica koja je vidljiva korisniku je početna stranica. Na njoj su napisane osnovne upute kako se koristi web aplikacija. Ako korisnik nije prijavljen ima dvije tipke koje otvaraju meni za prijavu ili registraciju. Za tipke koristi se komponenta *md-filled-tonal-button*. Na Slici 4.3. prikazane su deklaracije varijabli i funkcije za promjenu stanja unutar varijabli unutar funkcije *HomePanel*.

```
function HomePanel() {
  const sessionData = JSON.parse(localStorage.getItem('sessionData'));
  const [showLogin, setShowLogin] = useState(false);
  const [showRegister, setShowRegister] = useState(false);
  const [counter, setCounter] = useState(0);

  const handleLoginClick = () =>{
    setShowLogin(true);
  }

  const handleCloseLogin = () =>{
    setShowLogin(false);
  }

  const handleRegisterClick = () =>{
    setShowRegister(true);
  }

  const handleCloseRegister = () =>{
    setShowRegister(false);
  }

  const handleLogoutClick = () => {
    localStorage.clear();
    if(counter === 0){
      setCounter(1);
    }
    else{
      setCounter(0);
    }
  }
}
```

Sl. 4.3. Prikaz programskog koda funkcije *HomePanel*.

Varijable *showLogin*, *showRegister* određuju hoće li se prikazivati skočni prozor za prijavu ili registraciju. Taj skočni prozor izveden je pomoću *modala*. *Modal* je komponenta koja je uvedena iz biblioteke *react-bootstrap*. Ako je pritisnuta tipka *Login* poziva se komponenta *LoginModal* koja prikazuje sučelje za prijavu, a ako je pritisnuta tipka *Register* poziva se komponenta *RegisterModal*. *LocalStorage* se koristi za spremanje podatka na uređaju korisnika. U ovom slučaju *LocalStorage* se koristi umjesto kolačića za pamćenje prijavljenog korisnika. Ako je korisnik prijavljen i varijabla *sessionData* je popunjena s podacima korisniku se prikazuje drugačije sučelje u odnosu na korisnika koji nije prijavljen. Ako je korisnik prijavljen postoji tipka *Logout*, te pritiskom na nju se poziva funkcija *handleLogoutClick* koja počisti *localStorage*

i promijeni vrijednost brojača kako bi se osvježila stranica, jer u React.js razvojnem okviru treba doći do neke promjene kako bi došlo do osvježavanja stranice.

```
if(sessionData != null){
  return (
    <>
    <div className='jumbotron jumbotron-fluid'>
      <h1
        className='display-4 text-center mt-5 w-100'
        id='text'
      >Welcome {sessionData['fname']} {sessionData['lname']} to the Quiz!</h1>
      <p
        className='lead text-center mt-3'
        id='text'
      >You are currently on the home page but if you want to start playing press the "Play" tab!</p>
      <p
        className='lead text-center mt-3'
        id='text'
      >Pressing the "Play" button will start the quiz and pressing the "Configure" button will allow you to adjust parameters</p>
      <p
        className='lead text-center mt-3'
        id='text'
      >You can also view your statistics on the "Profile" tab</p>
      <p className='lead text-center mt-3'
        id='text'
      >Or if you changed your mind you can log out by pressing the button below</p>

      <div className='container-fluid d-flex align-items-center justify-content-center'>
        <div className='d-flex flex-column text-center'>
          <md-filled-tonal-button
            id='primaryTonalButton'
            onClick={handleLogoutClick}
            class='mt-3'
          >Log out</md-filled-tonal-button>
        </div>
      </div>
    </div>
    </>
  );
}
```

Sl. 4.4. Prikaz programskog koda sučelja kada je korisnik prijavljen.

Na Slici 4.4. prikazan je programski kod sučelja ako je korisnik prijavljen. Ako varijabla *sessionData* nije prazna onda to znači da korisnik ima spremljene podatke na uređaju i da je prijavljen. Na Slici 4.5. prikazan je programski kod za sučelje kada korisnik nije prijavljen. Razlika između njih je to što se na stranici kod prijavljenog korisnika prikaže njegovo ime i prezime dobiveno prilikom registracije, a kod korisnika koji nije prijavljen prikažu se tipke koje omogućavaju registraciju ili prijavu. Na dnu programskog koda sučelja korisnika koji nije prijavljen pozivaju se komponente *LoginModal* i *RegisterModal* kojima se predaje varijabla *showLogin* i funkcije koje su odgovorne za zatvaranje skočnog prozora.

```

else{
  return (
    <>
    <div className='jumbotron jumbotron-fluid'>
      <h1
        className='display-4 text-center mt-5 w-100'
        id='text'
      >Welcome to the Quiz!</h1>
      <p className='lead text-center mt-3'
        id='text'
      >You are currently on the home page but if you want to start playing press the "Play" tab!</p>
      <p className='lead text-center mt-3'
        id='text'
      >Pressing the "Play" button will start the quiz and pressing the "Configure" button will allow you to adjust parameters</p>
      <p className='lead text-center mt-3 mb-3'
        id='text'
      >or you can also log in or register.</p>
    </div>

    <div className='container-fluid d-flex align-items-center justify-content-center'>
      <div className='d-flex flex-column text-center'>
        <md-filled-tonal-button id='primaryTonalButton'
          onClick={handleLoginClick}
          class='mt-3'
        >Log in</md-filled-tonal-button>
        <md-filled-tonal-button id='primaryTonalButton'
          class='mt-3'
          onClick={handleRegisterClick}
        >Register</md-filled-tonal-button>
      </div>
    </div>

    <LoginModal show={showLogin} handleClose={handleCloseLogin}/>
    <RegisterModal show={showRegister} handleClose={handleCloseRegister}/>
  </>
);
}

```

Sl. 4.5. Prikaz programskog koda sučelja kada je korisnik nije prijavljen.

4.3. Komponente skočnih prozora za prijavu i registraciju

Kod komponente za prijavu prvi puta se koristi komunikacija sa poslužiteljem. Poslužitelj je napravljen na lokalnom računalu putem programa *XAMPP*. Funkcije *LoginModal* i *RegisterModal* rade na vrlo sličan način. Obje za upis koriste komponente *md-filled-text-field* i na isti način se obavlja provjera jesu li podaci točno upisani u polja za upis. Najveća razlika između njih je što *LoginModal* samo ispisuje grešku ako korisnik upiše netočne podatke, a *RegisterModal* ima dodatnu provjeru za valjanost korisničkog imena. Na Slici 4.6. prikazan je programski kod gdje se deklariraju varijable u koje se spremaju podaci iz forme i varijable koje u sebi sadrže informacije postoji li negdje unutar formi greška. Također je definirana funkcija *checkEmpty* koja provjerava jesu li polja za upis prazna nakon što se korisnik prebaci s jednog polja na drugo.

```

const [formData, setFormData] = useState({
  username: '',
  password: '',
});

const [errors, setErrors] = useState({
  username: undefined,
  password: undefined,
});

const [incorrectError, setIncorrectError] = useState(undefined);

const handleInputChange = (e) => {
  const {name, value} = e.target;
  setFormData({ ...formData, [name]: value });
}

const checkEmpty = (e) => {
  const { name, value } = e.target;
  const isEmpty = value.trim() === '';

  if (isEmpty) {
    e.target.setCustomValidity(`${name} cannot be empty`);
  } else {
    e.target.setCustomValidity('');
  }
}

setErrors((prevErrors) => ({
  ...prevErrors,
  [name]: isEmpty ? `${name} cannot be empty` : undefined,
}));
};

```

Sl. 4.6. Prikaz programskog koda deklariranja varijabli u *LoginModal* funkciji.

Kako bi se na poljima za unos ispravno prikazalo stanje pogreške, odnosno kada je neko polje prazno ili je upis netočan, koristi se vrijednost *true*. Ako se sa polja treba maknuti stanje pogreške, varijabla *errors* se postavlja na *undefined* i pogreška se prestaje prikazivati korisniku.

```

const handleLogin = () => {
  const form = document.getElementById('loginForm');
  const elementsWithErrors = form.querySelectorAll(':invalid');

  if (elementsWithErrors.length > 0) {
    console.log("Form has errors. Cannot submit.");
    return;
  }

  try {
    const response = fetch('http://localhost/Zavrzni rad/login.php', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(formData),
    })

    .then(response => response.json())
    .then(responseData => {
      if (responseData.success) {
        console.log("Successful login");
        localStorage.setItem('sessionData', JSON.stringify(responseData.data));
        setIncorrectError(undefined);
        handleClose();
      } else {
        setIncorrectError(true);
      }
    })
  } catch (error) {
    console.error("ERROR WHILE LOGGING IN: ", error);
  }
}

```

Sl. 4.7. Prikaz programskog koda funkcije *handleLogin*.

Na Slici 4.7. prikazan je programski kod funkcije *handleLogin*. Pritiskom na tipku *Login* izvodi se funkcija *handleLogin*. Ako forma ima greške ne dozvoljava se slanje podataka poslužitelju. Ako forma nema greške pomoću funkcije *fetch* šalje se zahtjev poslužitelju i čeka se njegov odgovor. U odgovoru se nalaze korisnički podaci u JSON formatu koji se zatim spremaju u LocalStorage pod nazivom *sessionData*. Podacima u *LocalStorage* pristupa se na stranicama za provjeru je li korisnik prijavljen.

```

return (
  <>
  <div className='container-fluid align-items-center justify-content-center'
    id='modalStyle'>
    <Modal
      show={show}
      onHide={handleClose}
      id='modal-header'>
      <Modal.Header
        id='modalStyle'>
        <Modal.Title id='text'>Log in</Modal.Title>
      </Modal.Header>
      <form
        onSubmit={(e) => {e.preventDefault();}}
        id='loginForm'>
      <Modal.Body id='modalStyle'>
        <div className='container-fluid d-flex align-items-center justify-content-center'>
          <div className='d-flex flex-column'>
            <md-filled-text-field
              label='Username'
              type='text' id='textField'
              name='username'
              onChange={handleInputChange}
              onBlur={ (e) => {checkEmpty(e);}}
              error={errors.username || incorrectError}
              required></md-filled-text-field>
            <md-filled-text-field
              label='Password'
              type='password'
              id='textField'
              name='password'
              class='mt-2'
              onChange={handleInputChange}
              onBlur={ (e) => {checkEmpty(e);}}
              error={errors.password || incorrectError}
              error-text={"Incorrect username or password"}
              required></md-filled-text-field>
          </div>
        </div>
      </Modal.Body>
      <Modal.Footer id='modalStyle'>
        <md-filled-tonal-button id='secondaryTonalButton' onClick={handleClose}>Cancel</md-filled-tonal-button>
        <md-filled-button id='primaryButton' type='submit' onClick={ () => {handleLogin();}}>Log in</md-filled-button>
      </Modal.Footer>
    </form>
  </Modal>
  </div>
  </>
);

```

Sl. 4.8. Prikaz programskog koda sučelja *LoginModal* komponente.

Na Slici 4.8. prikazan je programski kod za sučelje *LoginModal* komponente. Skočni prozor koristi komponentu *Modal* koja u sebi sadrži *Modal.Header*, *Modal.Body* i *Modal.Footer*. Za svako polje za unos podataka postavljeni su isti atributi. Atribut *label* postavlja oznaku na polje za unos, *type* predstavlja tip podatka koji treba biti upisan u polje. Ako je *type* postavljen na *password* korisniku neće biti vidljiv tekst koji je upisan nego samo zvjezdice ili točke. Atribut *onInput* poziva se svaki put kada se upiše vrijednost u polje. U *onInput* poziva se funkcija koja mijenja vrijednosti varijabli koje trebaju sadržavati informacije iz polja. Atribut *onBlur* poziva funkciju *checkEmpty* svaki put

kada korisnik odluči prebaciti fokus sa jednog polja za unos na drugo. Atribut *error* određuje hoće li se na polju za unos prikazivati pogreška, odnosno hoće li polje za unos imati prikazanu grešku koja je upisana u *error-text* atribut, te hoće li polje za unos koristiti boje koje su ranije definirane za prikaz pogreške.

```
<?php
include 'connect.php';

header('Access-Control-Allow-Origin: http://localhost:3000');
header('Access-Control-Allow-Headers: Content-Type');
header('Content-Type: application/json');

$data = json_decode(file_get_contents('php://input'), true);

if (!isset($data['username']) || !isset($data['password'])) {
    echo json_encode(['success' => false, 'message' => 'Error logging in user']);
    exit;
}

$username = $data['username'];
$password = $data['password'];

$stmt = $conn->prepare("SELECT id, username, password, fname, lname, userRole, email FROM users WHERE username = ?");
$stmt->bind_param("s", $username);

if ($stmt->execute()) {
    $result = $stmt->get_result();
    if ($result->num_rows > 0) {
        $row = $result->fetch_assoc();
        if(password_verify($password, $row['password'])){
            $stmt = $conn->prepare("SELECT gamesPlayed, questionsAnswered, correctAnswers FROM userstatistics WHERE idUser = ?");
            $stmt->bind_param("s", $row['id']);
            $stmt->execute();
            $result = $stmt->get_result();
            $userStatisticsRow = $result->fetch_assoc();

            $stmt = $conn->prepare("UPDATE userLogs SET lastLogin = CURRENT_TIMESTAMP WHERE idUser = ?");
            $stmt->bind_param("s", $row['id']);
            $stmt->execute();

            $_SESSION['username'] = $username;
            $_SESSION['fname'] = $row['fname'];
            $_SESSION['lname'] = $row['lname'];
            $_SESSION['email'] = $row['email'];
            $_SESSION['userRole'] = $row['userRole'];
            $_SESSION['gamesPlayed'] = $userStatisticsRow['gamesPlayed'];
            $_SESSION['questionsAnswered'] = $userStatisticsRow['questionsAnswered'];
            $_SESSION['correctAnswers'] = $userStatisticsRow['correctAnswers'];

            echo json_encode(['success' => true, 'data' => $_SESSION]);
        }else {
            echo json_encode(['success' => false, 'message' => 'Invalid username or password']);
        }
    } else {
        echo json_encode(['success' => false, 'message' => 'Invalid username or password']);
    }
} else {
    echo json_encode(['success' => false, 'message' => 'Error logging in user']);
}

$stmt->close();
$conn->close();
?>
```

Sl. 4.9. Prikaz programskog koda poslužitelja za prijavu.

Na Slici 4.9. prikazan je programski kod poslužitelja za prijavu korisnika. Prvo se dozvoli pristup adresi gdje se nalazi web aplikacija, nakon toga se dozvoli *header* sa tipom *application/json*. Nakon toga se u varijablu *data* spremaju podaci koji su poslani prilikom prijave, te se odradi provjera je li neki od podataka prazan, te ako je nešto prazno vrati se greška. Ako nije prazno, naprave se

dvije nove varijable *username* i *password*. Pripremi se naredba za dohvaćanje podataka za korisničko ime koje je uneseno. Ako korisnik postoji verificira se zaporka pomoću funkcije *password_verify*. U *session* varijablu sprema se korisnički podaci te se ona kodira u JSON oblik i vraća nazad korisniku. Ako je zaporka netočna ili ako korisničko ime ne postoji pošalje se odgovarajuća pogreška.

```
const checkPasswordMatch = (e) => {
  const repeatPasswordField = e.target.form['repeatPassword'];
  if (formData.password !== formData.repeatPassword) {
    repeatPasswordField.setCustomValidity('Passwords do not match');
    setErrors((prevErrors) => ({
      ...prevErrors,
      repeatPassword: 'Passwords do not match',
    }));
  } else {
    repeatPasswordField.setCustomValidity('');
    setErrors((prevErrors) => ({
      ...prevErrors,
      repeatPassword: undefined,
    }));
  }
};

const checkUsernameAvailability = async (username) => {
  try {
    const response = await fetch('http://localhost/Zavrsni rad/checkUsername.php', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ username }),
    });
    return await response.json();
  } catch (error) {
    console.error("Error while checking username availability: ", error);
    return { success: false, error: "Error checking username availability" };
  }
};
```

Sl. 4.10. Prikaz programskog koda funkcija *checkPasswordMatch* i *checkUsernameAvailability*.

Komponenta za registraciju radi na jako sličan način kao komponenta za prijavu. Za razliku od komponente za prijavu, ona sadrži funkcije *checkPasswordMatch* koja provjerava podudaraju li se vrijednosti iz oba polja za unos zaporka, te funkciju *checkUsernameAvailability* koja šalje upit poslužitelju za provjeru je li korisničko ime zauzeto. Funkcije su prikazane na Slici 4.10.

4.4. Komponenta stranice za početak kviza

Slijedeća komponenta je komponenta stranice za početak kviza. Na toj stranici korisnik ima tipku *Configure* i tipku *Play*. Pritiskom na *Configure* korisniku se otvori skočni prozor u kojemu može birati kategoriju pitanja i broj pitanja na koje želi odgovarati. Kada se komponenta učita prvo se dohvati *sessionData* iz *LocalStoragea*, zatim se deklarira varijabla za navigaciju, te varijable i funkcije za prikazivanje skočnih prozora.

```
const handlePlayClick = () => {  
  if(sessionData == null){  
    setShowLoginAlert(true);  
  }  
  else{  
    fetchQuestions();  
  }  
}
```

Sl. 4.11. Prikaz programskog koda funkcije *handlePlayClick*.

Na Slici 4.11. prikazan je programski kod funkcije *handlePlayClick* koja pokreće kviz. Pritiskom na tipku obavi se provjera je li korisnik prijavljen, i ako nije, korisniku se prikazuje skočni prozor s porukom o obaveznoj prijavi. Ako je korisnik prijavljen poziva se funkcija *fetchQuestions*.

```
const fetchQuestions = () => {  
  try {  
    const response = fetch('http://localhost/Zavrsni rad/fetchQuestions.php', {  
      method: 'POST',  
      headers: {  
        'Content-Type': 'application/json',  
      },  
    });  
    .then(response => response.json())  
    .then(responseData => {  
      if(responseData.success){  
        localStorage.setItem("questions", JSON.stringify(responseData.data));  
        navigate('/play/quiz');  
      }else{  
        console.log("NO QUESTIONS IN DATABASE");  
      }  
    });  
  }catch (error) {  
    console.error("Error while grabbing questions: ", error);  
  }  
}
```

Sl. 4.12. Prikaz programskog koda funkcije *fetchQuestions*.

Funkcija *fetchQuestions* prikazana je na Slici 4.12. Funkcija dohvaća sva pitanja sa poslužitelja i sprema ih u *LocalStorage*. Nakon spremanja pitanja korisnik je proslijeđen na stranicu gdje se prikazuju pitanja i odgovori.

```
return (  
<>  
<div className='jumbotron jumbotron-fluid'>  
  <h1  
    className='display-4 text-center mt-5 w-100'  
    id='text'>Welcome to the Quiz!</h1>  
  <p className='lead text-center mt-3'  
    id='text'>Press the "Play" button to start playing</p>  
  <p className='lead text-center mt-3 mb-3'  
    id='text'>or press the "Configure" button to adjust settings!</p>  
</div>  
<div className='container-fluid d-flex align-items-center justify-content-center'>  
  <div className='d-flex flex-column text-center'>  
    <md-filled-button id='primaryButton' class='mt-3'  
      onClick={handlePlayClick}>Play</md-filled-button>  
    <md-filled-tonal-button id='secondaryTonalButton' class='mt-3'  
      onClick={handleConfigurationClick}>Configure</md-filled-tonal-button>  
  </div>  
</div>  
<ConfigureModal show={showConfiguration} handleClose={handleCloseConfiguration} />  
<LoginAlertModal show={showLoginAlert} handleClose={handleCloseLoginAlert} />  
</>  
>;
```

Sl. 4.14. Prikaz programskog koda sučelja stranice za pokretanje kviza.

Na Slici 4.14. prikazan je programski kod sučelja stranice za pokretanje kviza. Na stranici postoje dvije tipke od kojih jedna pokreće kviz, a druga pokreće skočni prozor za konfiguraciju. Kako bi se korisniku dalo do znanja da tipke rade drugačije akcije jedna od njih koristi komponentu *md-filled-button*, a druga koristi komponentu *md-filled-tonal-button*. Pritiskom na tipku *Configure* korisniku se otvara skočni prozor koji omogućava korisniku izbor kategorije pitanja i broja pitanja na koje želi odgovarati prilikom igranja kviza. Ako korisnik odabere više pitanja nego što postoji za određenu kategoriju kviz završava kada dođe do zadnjeg pitanja.

4.5. Komponenta skočnog prozora za promjenu postavki kviza

Pozivom komponente *ConfigureModal* otvara se skočni prozor koji omogućuje odabir kategorije iz koje su postavljena pitanja i broj pitanja koji je postavljen. Nazivi kategorija dohvate se prilikom svakog pokretanja komponente i prikazu se pomoću *md-filled-select* komponente koja omogućuje odabir kategorije iz padajućeg izbornika. Ako je odabrana kategorija *mixed quiz* mogu se dobiti pitanja iz svih kategorija.

```
const handleSelectChange = (e) => {
  const selectedIndex = e.target.selectedIndex;
  const selectedCategory = categories[selectedIndex];
  setCategory(selectedCategory);
  console.log(selectedCategory);
};

const handleInputChange = (e) => {
  const {name, value} = e.target;
  setNumberOfQuestions({ ...numberOfQuestions, [name]: value });
  console.log(numberOfQuestions)
}

const checkIsNumber = (e) => {
  const numberField = e.target.form['numberOfQuestions'];
  if(Number.isInteger(parseInt(numberField.value))){
    numberField.setCustomValidity('');
    setErrors((prevErrors) => ({
      ...prevErrors,
      numberOfQuestions: undefined,
    }));
  }else{
    numberField.setCustomValidity('The input is not a number!');
    setErrors((prevErrors) => ({
      ...prevErrors,
      numberOfQuestions: 'The input is not a number',
    }));
  }
}
```

Sl. 4.15. Prikaz programskog koda funkcije *ConfigureModal*.

Na Slici 4.15. prikazan je programski kod funkcija *handleSelectChange*, *handleInputChange* i *checkIsNumber*. Funkcija *handleSelectChange* poziva se svaki put kada korisnik pritisne na padajući izbornik i odabere drugu vrijednost. Funkcija *handleInputChange* mijenja vrijednost varijable *numberOfQuestions* ovisno o tome koji broj korisnik unese u polje za unos. Funkcija *checkIsNumber* provjerava je li unesena vrijednost broj i ako nije postavlja pogrešku o spremanju neočekivane vrijednosti.

```

useEffect(() => {
  const selectElement = document.querySelector('#categorySelect');
  if (selectElement) {
    selectElement.addEventListener('change', handleSelectChange);
  }
  return () => {
    if (selectElement) {
      selectElement.removeEventListener('change', handleSelectChange);
    }
  };
});

const fetchCategories = async () => {
  try {
    const response = await fetch('http://localhost/Zavrsni rad/fetchCategories.php');
    const data = await response.json();
    console.log(data);
    return data.categories;
  } catch {
    console.error('Error fetching categories');
    return [];
  }
};

useEffect(() => {
  const fetchData = async () => {
    const fetchedCategories = await fetchCategories();
    setCategories(fetchedCategories);
  };
  if (show) {
    fetchData();
  }
}, [show]);

```

Sl. 4.16. Prikaz programskog koda funkcije *ConfigureModal*.

Na Slici 4.16. prikazan je programski kod u kojemu se definira funkcija *fetchCategories* i koriste dvije *useEffect* kuke. Prva *useEffect* kuka postavlja ili briše slušatelj događaja na odabrani element padajućeg izbornika. To je potrebno kako bi se znalo na koji element izbornika je promijenjen odabir i kada se to dogodi. Funkcija *fetchCategories* dohvaća sve kategorije iz baze podataka preko poslužitelja. Druga *useEffect* kuka poziva funkciju *fetchCategories* kada se pokrene komponenta *configureModal* kako bi se učitale kategorije i na vrijeme postavile unutar padajućeg izbornika.

```

return (
  <>
    <div className='container-fluid align-items-center justify-content-center' id='modalStyle'>
      <Modal show={show} onHide={handleClose} id='modal-header'>
        <Modal.Header id='modalStyle'>
          <Modal.Title id='text'>Configure the quiz</Modal.Title>
        </Modal.Header>
        <form onSubmit={(e) => { e.preventDefault(); }} id='configForm'>
          <Modal.Body id='modalStyle'>
            <div className='container-fluid d-flex align-items-center justify-content-center'>
              <div className='d-flex flex-column'>
                <md-filled-select label='Categories' id='categorySelect' name='categorySelect' class='mt-2' required>
                  {categories.map((category) => (
                    <md-select-option key={category.id} value={category.category}>
                      <div slot='headline'>{category.category}</div>
                    </md-select-option>
                  ))}
                </md-filled-select>
                <md-filled-text-field value='5' label='Number of questions'
                  type='number' id='textField' name='numberOfQuestions'
                  class='mt-2' onChange={handleInputChange} onBlur={ (e) => {checkIsNumber(e);}}
                  error={errors.numberOfQuestions}></md-filled-text-field>
              </div>
            </div>
          </Modal.Body>
          <Modal.Footer id='modalStyle'>
            <md-filled-tonal-button id='secondaryTonalButton' onClick={handleClose}>Cancel</md-filled-tonal-button>
            <md-filled-button id='primaryButton' onClick={handleConfig}>Save</md-filled-button>
          </Modal.Footer>
        </form>
      </Modal>
    </div>
  </>
);

```

Sl. 4.17. Prikaz programskog koda sučelja komponente *ConfigureModal*.

Na Slici 4.17. prikazan je programski kod sučelja komponente *ConfigureModal*. Komponenta radi na sličan način kao ostale komponente skočnih prozora, ali u ovom slučaju koristi se funkcija *map* koja omogućava dinamičko stvaranje opcija padajućeg izbora ovisno o sadržaju varijable *categories*. Odabrana kategorija spremi se u varijablu *category* i spremi se u *LocalStorage* pritiskom na tipku za spremanje.

4.6. Komponenta za igranje kviza

Slijedeća komponenta definira kako kviz radi i izgleda. Pitanje se prikazuje na velikoj tipki koja je isključena, te ima drugačiji dizajn od tipki za odgovor. Postoje četiri tipke za odgovor koje su postavljene ispod pitanja u dva retka. Tipka na kojoj je prikazano pitanje ne može se pritisnuti. Pritiskom na jedan od četiri ponuđena odgovora promjeni se pitanje i odgovori. Kada se odgovori na zadnje pitanje korisnik je prosljeđen na stranicu sa rezultatima kviza. Kako bi se kviz brojao, korisnik mora odgovoriti na sva pitanja. Ne postoji vremensko ograničenje.

```

const filteredQuestions = (!category || !category.id || category.category === "mixed quiz") ?
questions : questions.filter(question => Number(question.idCategory) === Number(category.id));

const [currentQuestionIndex, setCurrentQuestionIndex] = useState(0);
const [shuffledAnswers, setShuffledAnswers] = useState([]);
const correctAnswersCount = useRef(0);
const [answersShuffled, setAnswersShuffled] = useState(false);
const [shuffledQuestions, setShuffledQuestions] = useState([]);
const [questionsShuffled, setQuestionShuffled] = useState(false);
const selectedAnswers = useRef([]);

useEffect(() => {
  if (!questionsShuffled && filteredQuestions.length > 0) {
    setShuffledQuestions(shuffleArray(filteredQuestions).slice(0, numberOfQuestions));
    setQuestionShuffled(true);
  }
}, [filteredQuestions, numberOfQuestions, questionsShuffled]);

useEffect(() => {
  if (!answersShuffled && shuffledQuestions.length > 0) {
    const answers = [
      shuffledQuestions[currentQuestionIndex].firstAnswer,
      shuffledQuestions[currentQuestionIndex].secondAnswer,
      shuffledQuestions[currentQuestionIndex].correctAnswer,
      shuffledQuestions[currentQuestionIndex].thirdAnswer
    ];
    setShuffledAnswers(shuffleArray(answers));
    setAnswersShuffled(true);
  }
}, [currentQuestionIndex, shuffledQuestions, answersShuffled]);

```

Sl. 4.18. Prikaz programskog funkcije *Questions*.

Na Slici 4.24. prikazan je prvi dio programskog koda funkcije *Questions*. Iz *LocalStorage* se učitava kategorija koju je korisnik odlučio igrati i pitanja koja su dohvaćena u prethodnom koraku, prije prosljeđivanja na stranicu za igru. Isto tako je učitani broj pitanja koje je korisnik postavio ranije. Ako korisnik nije postavio broj pitanja ili se dogodila neka pogreška, broj pitanja se postavlja na pet. Prvo što se napravi je filtriranje pitanja na onu kategoriju koju je korisnik odabrao. Ako je odabrana kategorija za mješoviti kviz, korisnik može dobiti pitanja iz svake kategorije. Dodane su varijable koje prate redni broj pitanja, koliko korisnik ima točnih odgovora i koji su njegovi odgovori. Također postoje varijable za spremanje liste pomiješanih pitanja i liste sa pomiješanim odgovorima, što omogućava da njihov redoslijed nikada ne bude isti. Za pohranu odgovora i broja točnih odgovora koristi se funkcija *useRef* umjesto *useState* jer funkcija *useState* ažurira stanje unutar varijable samo prilikom ažuriranja prikazanog stanja cijele komponente. To nije poželjno ponašanje jer se prilikom odgovaranja na zadnje pitanje korisnik proslijedi na stranicu sa rezultatima i ne dođe do ažuriranja prikaza komponente i zadnji odgovor se izgubi. Funkcija *useRef* ažurira vrijednosti bez čekanja ažuriranje prikaza. Prva *useEffect* kuka provjerava jesu li pitanja već označena kao pomiješana i postoje li pitanja u listi filtriranih pitanja. Ako pitanja nisu pomiješana onda se izvodi funkcija *shuffleArray*. Kuka se izvodi ako se promijeni stanje varijabli

filteredQuestions, *numberOfQuestions* ili *questionsShuffled*. Druga kuka koristi se za promjenu redoslijeda odgovora za pojedino pitanje. Kada se promijeni pitanje izvodi se promjena redoslijeda odgovora i oni se onda prikazu na stranici.

```
function shuffleArray(array) {  
  for (let i = array.length - 1; i > 0; i--) {  
    const j = Math.floor(Math.random() * (i + 1));  
    [array[i], array[j]] = [array[j], array[i]];  
  }  
  return array;  
}
```

Sl. 4.19. Prikaz programskog koda funkcije za promjenu redoslijeda elemenata polja.

Na Slici 4.19. prikazan je programski kod funkcije *shuffleArray*. Ta funkcija omogućava promjenu redoslijeda elemenata polja koje joj se proslijedi. Ista funkcija se koristi za promjenu redoslijeda pitanja i odgovora.

```
const handleNextQuestion = (selectedAnswer) => {  
  selectedAnswers.current = [...selectedAnswers.current, selectedAnswer];  
  if (selectedAnswer === shuffledQuestions[currentQuestionIndex].correctAnswer) {  
    correctAnswersCount.current += 1;  
  }  
  if (currentQuestionIndex < shuffledQuestions.length - 1) {  
    setCurrentQuestionIndex(currentQuestionIndex + 1);  
    setAnswersShuffled(false);  
  } else {  
    endQuiz();  
  }  
}
```

Sl. 4.19. Prikaz programskog koda funkcija *handleNextQuestion*.

Na Slici 4.19. prikazan je programski kod funkcija *handleNextQuestion*. Funkcija *handleNextQuestion* poziva se kada korisnik pritisne na neki od ponuđenih odgovora u kvizu. Funkcija dodijeli zadnji pritisnut odgovor varijabli *selectedAnswers* i ako je odgovor točan poveća broj točnih odgovora za kviz koji se trenutno igra. Sljedeća provjera je provjera je li broj trenutnog pitanja prelazi broj svih pitanja i ako ne prelazi varijabla *answersShuffled* postavlja se na *false* kako bi se ponovno izvelo miješanje odgovora za nadolazeće pitanje, te se povećava broj trenutnog pitanja. Ako je broj trenutnog pitanja veći od sveukupnog broja pitanja poziva se funkcija *endQuiz* i završava se kviz.


```

const endQuiz = () => {
  sessionData.correctAnswers += correctAnswersCount.current;
  sessionData.questionsAnswered += currentQuestionIndex + 1;
  sessionData.gamesPlayed += 1;

  let lastQuiz = {
    "questions": shuffledQuestions,
    "answers": selectedAnswers.current,
  };

  localStorage.setItem("sessionData", JSON.stringify(sessionData));
  localStorage.setItem("lastQuiz", JSON.stringify(lastQuiz));
  updateUserStatistics()
    .then(() => navigate("/play/quiz/results"));
}

```

Sl. 4.20. Prikaz programskog koda funkcije *endQuiz*.

Na Slici 4.20. prikazan je programski kod funkcije *endQuiz*. Kada se odgovori na zadnje pitanje kviza poziva se funkcija *endQuiz*. Funkcija ažurira statistiku korisnika koja je dohvaćena iz *LocalStoragea*. U varijablu *lastQuiz* sprema pitanja koja su postavljena korisniku i odgovore koje je korisnik dao na pitanja. Zatim se varijabla *lastQuiz* sprema u *LocalStorage*.

```

if (!shuffledQuestions || shuffledQuestions.length === 0) {
  return (
    <div className='container-fluid text-center'>
      <p className='display-4 text-center mt-5' id='text'>THERE ARE NO QUESTIONS FOR THIS CATEGORY!</p>
    </div>
  );
}

return (
  <>
    <div className='container-fluid text-center'>
      <md-filled-button id='questionButton' class='mt-3 mb-5 w-100' disabled>
        <div className='container w-100 h-100'>
          <p className='questionText' id='questionText'>{shuffledQuestions[currentQuestionIndex].questionText}</p>
        </div>
      </md-filled-button>
    </div>

    <div className='container-fluid text-center'>
      <div className='row'>
        {shuffledAnswers.map((answer, index) => (
          <div className='col-md-6' key={index}>
            <md-filled-button id='answerButton' class='mt-3 w-100' onClick={() => handleNextQuestion(answer)}>
              <p id='questionText'>{answer}</p>
            </md-filled-button>
          </div>
        ))}
      </div>
    </div>
  </>
);

```

Sl. 4.21. Prikaz programskog koda sučelja za igru.

Na Slici 4.21. prikazan je programski kod sučelja komponente za igru kviza. Ako je varijabla s pitanjima prazna prikazuje se poruka kako pitanja za tu kategoriju ne postoje. Za prikazivanje pitanja koriste se varijable *shuffledQuestions* i *currentQuestionIndex*. Varijabla *currentQuestionIndex* koristi se za prikaz pitanja odgovarajućeg indeksa iz polja *shuffledQuestions*. Za prikaz pitanja ponovno se koristi funkcija *map*. Svaka tipka za odgovor postavljena je u odvojenom HTML elementu *div*. Svaki *div* dobije svoj jedinstveni ključ koji predstavlja njegov indeks. Svaka tipka poziva funkciju *handleNextQuestion* kojoj se proslijeđuje odgovor dodijeljen funkcijom *map*.

4.7. Komponenta stranice korisničkog profila

Svaki korisnik ima mogućnost vidjeti svoj profil. Stranica profila prikazuje statistike o igrama korisnika i tipku za odjavu. Korisnik ima mogućnost promijeniti svoje korisničke podatke, odnosno ime i prezime, adresu elektroničke pošte i korisničko ime. Ako je korisnik administrator aplikacije on isto tako ima mogućnost dodati nova pitanja, nove kategorije i promijeniti naziv kategorije pritiskom na odgovarajuće tipke. Ako korisnik koji nije prijavljen ode na stranicu profila naznačeno je kako se prvo mora registrirati ili prijaviti i prikazane su mu tipke za prijavu i registraciju. Deklariraju se varijable potrebne za korištenje skočnih prozora, osvježavanje zaslona i učitavanje korisničkih podataka iz *LocalStoragea*. Način rada sa skočnim prozorima isti je kao kod komponenti za registraciju i prijavu.

```

else{
  return (
    <>
    <div className='jumbotron jumbotron-fluid'>
      <h1 className='display-4 text-center mt-5 w-100' id='text'>Welcome!</h1>
      <p className='lead text-center mt-3' id='text'>We appreciate your enthusiasm but you need to log in or register first!</p>
      <p className='lead text-center mt-3 mb-3' id='text'>You can do that by pressing the appropriate button below!</p>
    </div>

    <div className='container-fluid d-flex align-items-center justify-content-center'>
      <div className='d-flex flex-column text-center'>
        <md-filled-tonal-button id='primaryTonalButton' onClick={handleLoginClick} class='mt-3'>Log in</md-filled-tonal-button>
        <md-filled-tonal-button id='primaryTonalButton' class='mt-3' onClick={handleRegisterClick}>Register</md-filled-tonal-button>
      </div>
    </div>

    <LoginModal show={showLogin} handleClose={handleCloseLogin}/>
    <RegisterModal show={showRegister} handleClose={handleCloseRegister}/>
  </>
  );
}

```

Sl. 4.22. Prikaz programskog koda sučelja stranice profila za korisnike koji nisu prijavljeni.

Prilikom učitavanja komponente obavi se provjera je li korisnik prijavljen i je li njegova uloga administrator. Ako je korisnik istovremeno i administrator prikazuje mu drugačije sučelje. Administratoru je omogućeno dodavanje novih pitanja i kategorija, brisanje pitanja i kategorija, te promjena naziva kategorija. Na vrhu stranice prikazane su tipke koje otvaraju skočne prozore pozivanjem odgovarajuće funkcije. Ispod administratorskih tipki prikazane su statistike administratora kao igrača. Na Slici 4.22. prikazan je programski kod sučelja kada korisnik nije prijavljen. Ako je korisnik prijavljen ima samo opciju za odjavu i promjenu korisničkih podataka, a ako nije prijavljen ima opciju za registraciju ili prijavu.

4.8. Komponenta stranice rezultata

Nakon što se odgovori na svako pitanje u kvizu korisnik je prosljeđen na stranicu s rezultatima kviza. Na stranici je prikazan njegov uspjeh na kvizu, pitanja koja su postavljena, točni odgovori na pojedino pitanje i odgovori koje je korisnik odabrao. Na Slici 4.33. prikazana je cijela funkcija *Results* koja učitava podatke o zadnjem igranom kvizu iz *LocalStoragea* i prikazuje ih korisniku.

```
function Results() {
  const sessionData = JSON.parse(localStorage.getItem('sessionData'));
  const lastQuiz = JSON.parse(localStorage.getItem('lastQuiz'));

  const totalQuestions = lastQuiz ? lastQuiz.answers.length : 0;
  let correctAnswers = 0;

  if (lastQuiz) {
    lastQuiz.questions.forEach((question, index) => {
      if (lastQuiz.answers[index] === question.correctAnswer) {
        correctAnswers++;
      }
    });
  }

  return (
    <>
    <div className='container-fluid text-center mt-5'>
      <h1 id="text">Quiz Results</h1>
      <h4 className='text-center mt-2' id='text'>Total questions: {totalQuestions}</h4>
      <h4 className='text-center mt-2' id='text'>Correct answers: {correctAnswers}/{totalQuestions}</h4>
    </div>
    <div className="container-fluid text-center mt-5 mb-5">
      <h2 id="text">Answer Sheet</h2>
      {lastQuiz && lastQuiz.answers.map((answer, index) => (
        <div key={index} className="question">
          <h3 id="text" className='text-center mt-3'>{index + 1}. {lastQuiz.questions[index].questionText}</h3>
          <h4 id="text" className='text-center mt-3'>Correct Answer: {lastQuiz.questions[index].correctAnswer}</h4>
          <h4 id="text" className='text-center mt-3'>Your Answer: {answer}</h4>
        </div>
      ))}
    </div>
  </>
);
}
```

Sl. 4.23. Prikaz programskog koda komponente *Results*.

4.9. Prilagodba komponenti

Jedan od najvećih razloga za implementaciju sučelja pomoću *Material Web* paketa jest mogućnost prilagodbe svake komponente. Svaka komponenta može se prilagoditi ovisno o tome što korisnik radi. Ima odvojene boje kada komponenta nije aktivna, korisnik pritisne i drži klik na komponenti, korisnik stavi pokazivač na komponentu. Svaka od komponenti ima definirane nazive stanja koje su prilagodljive *CSS-om* (engl. Cascading Style Sheets). Na Slici 4.24. prikazana je prilagodba boja komponente za unos teksta.

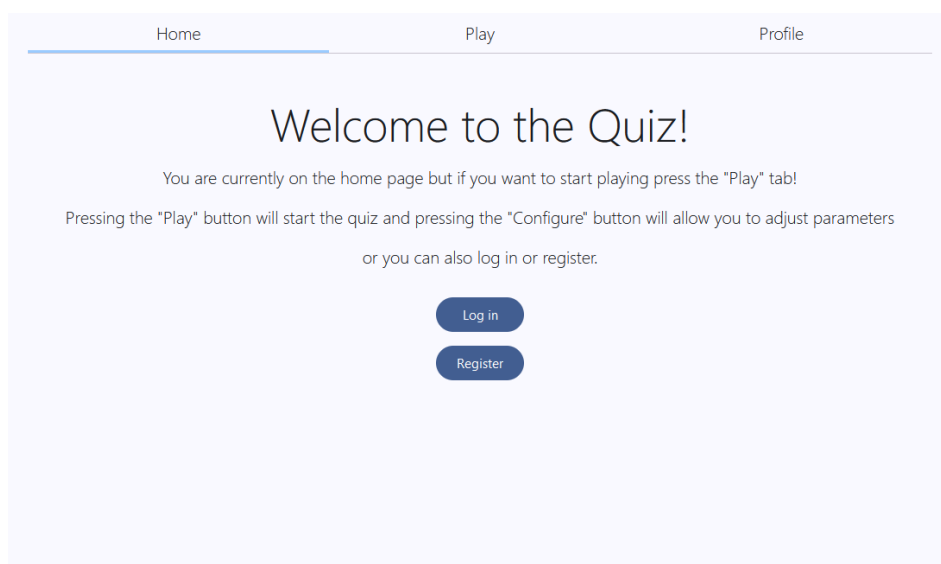
```
#textField{
  --md-sys-color-primary: #425e91;
  --md-sys-typescale-body-large-font: system-ui;

  --md-filled-text-field-container-color: #d7e3ff;
  --md-filled-text-field-focus-active-label-text-color: #425e91;
  --md-filled-text-field-focus-input-text-color: #425e91;
  --md-filled-text-field-input-text-color: #425e91;
  --md-filled-text-field-caret-color: #425e91;
  --md-filled-text-field-input-text-color: #243240;
  --md-filled-text-field-focus-input-text-color: #425e91;
  --md-filled-text-field-label-text-color: #243240;
  --md-filled-text-field-error-focus-active-input-text-color: #ba1a1a;
  --md-filled-text-field-error-focus-active-label-text-color: #ba1a1a;
  --md-filled-text-field-error-focus-active-indicator-color: #ba1a1a;
  --md-filled-text-field-error-focus-input-text-color: #ba1a1a;
  --md-filled-text-field-error-focus-label-text-color: #ba1a1a;
  --md-filled-text-field-error-label-text-color: #ba1a1a;
  --md-filled-text-field-error-hover-label-text-color: #ba1a1a;
  --md-filled-text-field-error-input-text-color: #ba1a1a;
  --md-filled-text-field-error-hover-input-text-color: #ba1a1a;
}
```

Sl. 4.24. Prikaz programskog koda prilagodbe boja komponente za unos teksta.

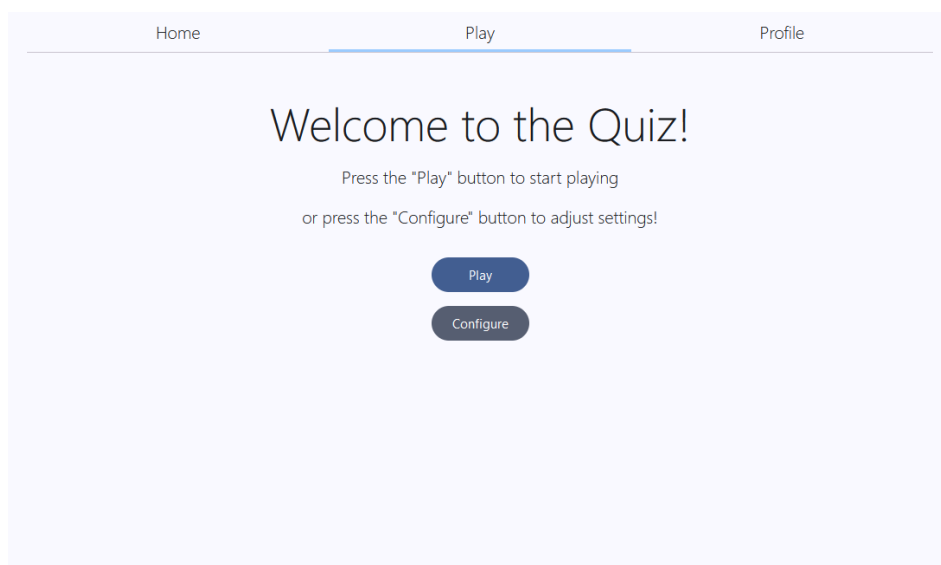
5. IZGLED IZRAĐENE APLIKACIJE

Na slijedećim slikama prikazano je konačno sučelje kviz web aplikacije. Boje su izabrane korištenjem alata „*Material Theme Builder*“. Stranica omogućava kreiranje palete boja za različite namjene unutar sučelja za tamnu i svijetlu temu aplikacije [10]. Na Slici 5.1. prikazana je početna stranica kada korisnik nije prijavljen.



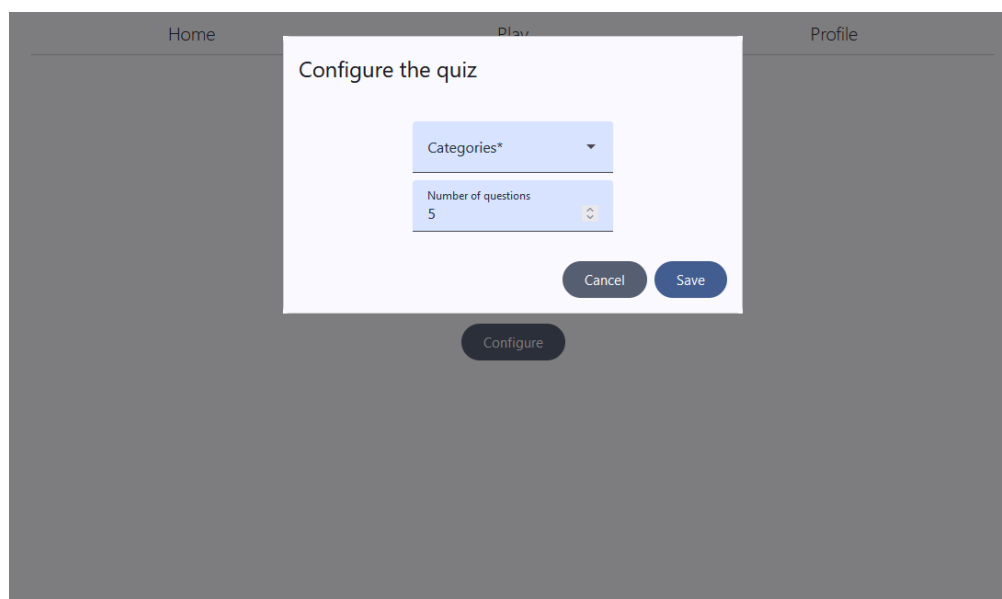
Sl. 5.1. Prikaz početne stranice.

Na Slici 5.2. prikazana je stranica za započinjanje i konfiguraciju kviza.



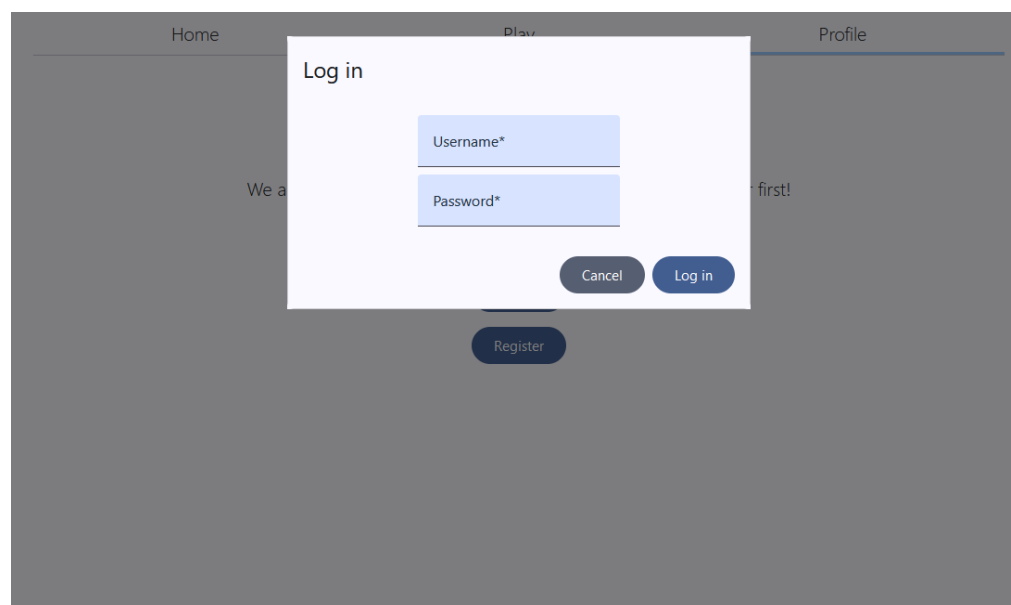
Sl. 5.2. Prikaz stranice za početak i konfiguraciju.

Na Slici 5.3. prikazan je skočni prozor za konfiguraciju kviza.



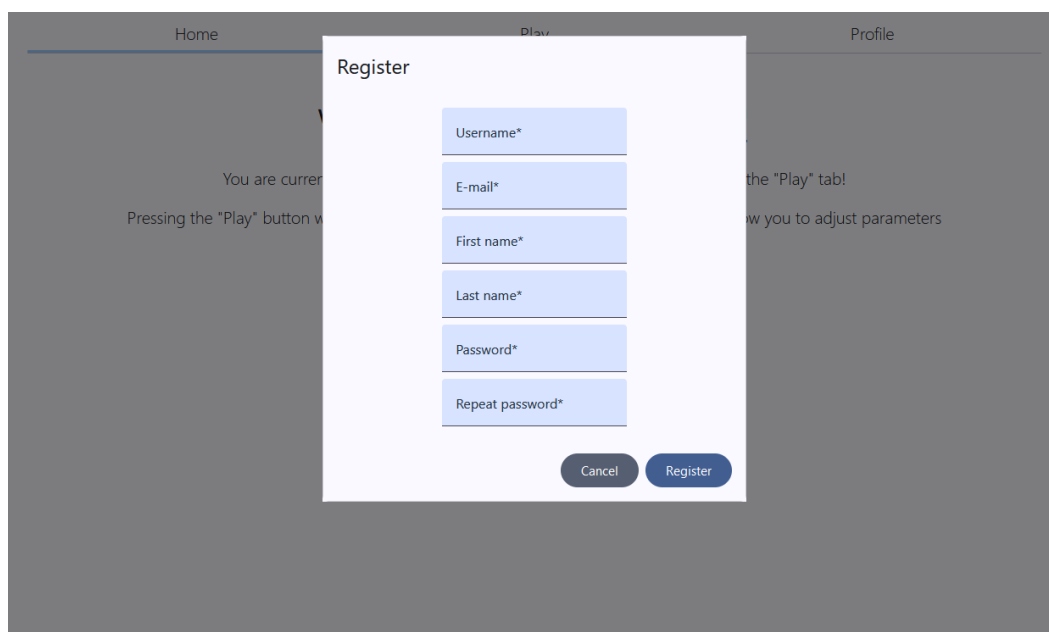
Sl. 5.3. Prikaz skočnog prozora za konfiguraciju kviza.

Na Slici 5.4. prikazan je skočni prozor za prijavu korisnika.



Sl. 5.4. Prikaz skočnog prozora za prijavu.

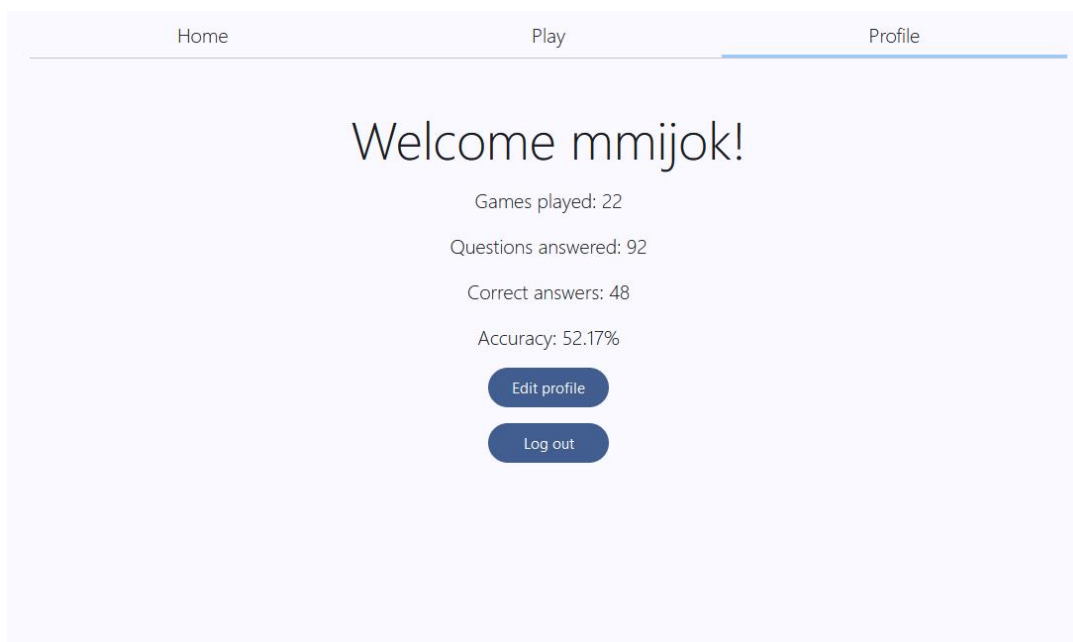
Na Slici 5.5. prikazan je skočni prozor za registraciju korisnika.



The image shows a 'Register' modal window overlaid on a dark background. The modal has a title 'Register' at the top. Below the title are six input fields, each with an asterisk indicating it is required: 'Username*', 'E-mail*', 'First name*', 'Last name*', 'Password*', and 'Repeat password*'. At the bottom right of the modal are two buttons: 'Cancel' and 'Register'.

Sl. 5.5. Prikaz skočnog prozora za registraciju.

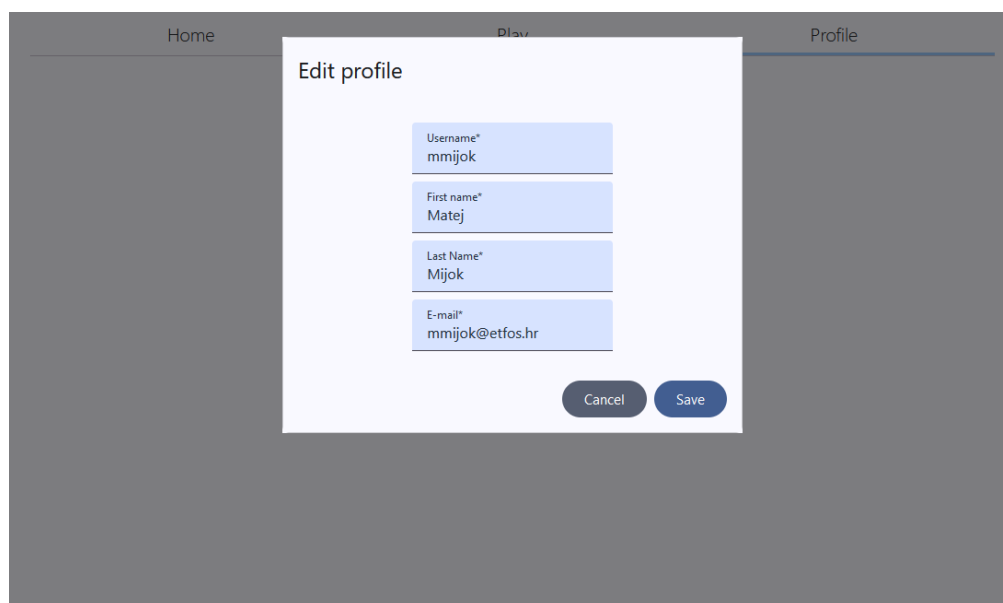
Na Slici 5.6. prikazana je stranica profila korisnika koji nije administrator.



The image shows a user profile page with a light purple background. At the top, there is a navigation bar with three tabs: 'Home', 'Play', and 'Profile', with 'Profile' being the active tab. The main content area displays a large heading 'Welcome mmijok!'. Below this, several statistics are listed: 'Games played: 22', 'Questions answered: 92', 'Correct answers: 48', and 'Accuracy: 52.17%'. At the bottom of the statistics section are two buttons: 'Edit profile' and 'Log out'.

Sl. 5.6. Prikaz stranice profila.

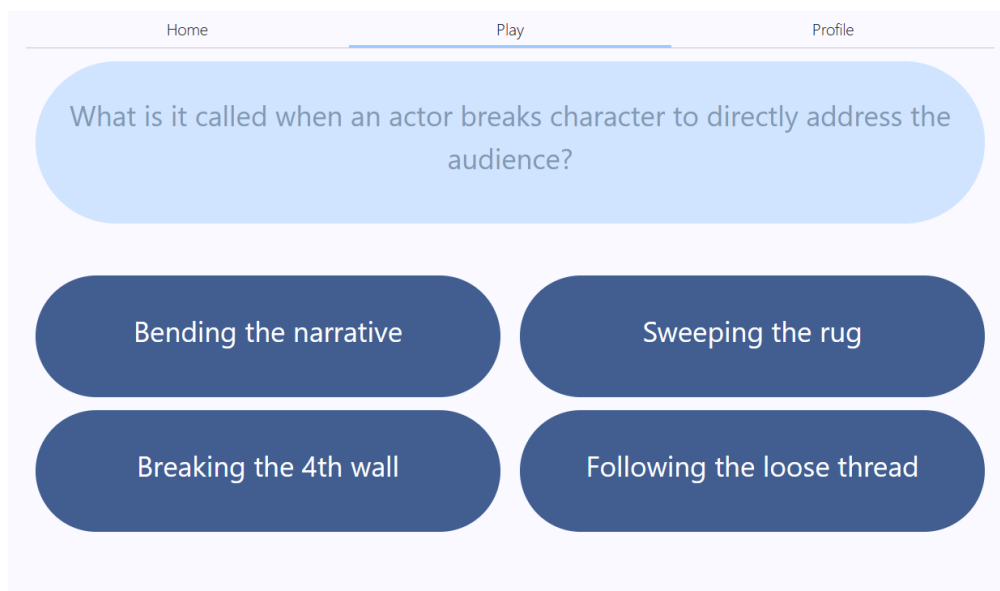
Na Slici 5.7. prikazan je skočni prozor za promjenu korisničkih podataka.



The image shows a web application interface with a dark grey background. At the top, there is a navigation bar with three tabs: 'Home', 'Play', and 'Profile'. The 'Profile' tab is currently selected. In the center, a white modal window titled 'Edit profile' is open. Inside the modal, there are four text input fields, each with a label and a value: 'Username*' with 'mmijok', 'First name*' with 'Matej', 'Last Name*' with 'Mijok', and 'E-mail*' with 'mmijok@etfos.hr'. At the bottom right of the modal, there are two buttons: 'Cancel' and 'Save'.

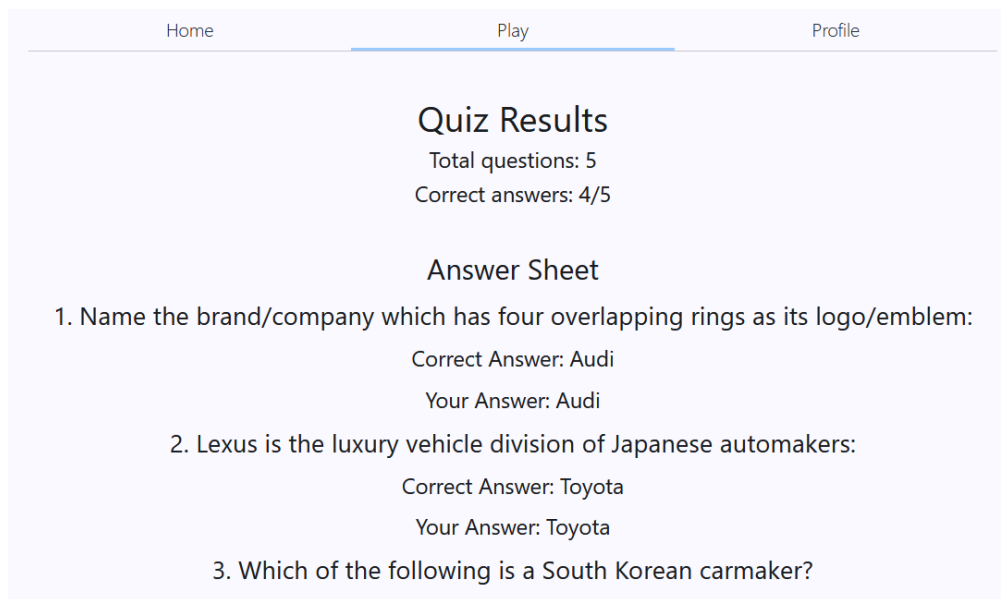
Sl. 5.7. Prikaz skočnog prozora za promjenu korisničkih podataka.

Na Slici 5.8. prikazana je stranica kviza.



The image shows a web application interface with a light purple background. At the top, there is a navigation bar with three tabs: 'Home', 'Play', and 'Profile'. The 'Play' tab is currently selected. In the center, there is a large light blue rounded rectangle containing the question: 'What is it called when an actor breaks character to directly address the audience?'. Below the question, there are four dark blue rounded rectangles arranged in a 2x2 grid, each containing a possible answer: 'Bending the narrative', 'Sweeping the rug', 'Breaking the 4th wall', and 'Following the loose thread'.

Sl. 5.8. Prikaz izgleda kviza.



Sl. 5.9. Prikaz stranice rezultata.

Na Slici 5.9. prikazan je izgled stranice rezultata. Na stranici je omogućeno pomjeranje prema dolje kako bi se vidjela sva odgovorena pitanja.

6. ZAKLJUČAK

Cilj ovog završnog rada bio je primijeniti *Material Design 3* smjernice pri izradi web aplikacije kviza. Komponente iz *Material Web* paketa programerima pružaju veliku slobodu u dizajnu sučelja zahvaljujući mogućnosti prilagodbe. Time se omogućava dosljednost u vizualnom identitetu aplikacije, bilo da se izrađuje za pojedinca ili tvrtku.

Međutim, rad s *Material Web* komponentama složeniji je u odnosu na rad s običnim *HTML* komponentama. Uporaba *Material Web* komponenti zahtjeva mnogo prilagodbi kako bi se postigla dosljednost sučelja jer je potrebno napraviti prilagodbe boje za svako od mogućih stanja. Komponente isto tako zahtijevaju slušatelje događaja za praćenje stanja komponenti kako bi se mogla pohraniti i koristiti njihova vrijednost. Potrebna je i uporaba varijabli za praćenje pogreški kako bi se korisniku moglo naznačiti gdje se pogreška nalazi, što čini uporabu *Material Web* komponenti složenijom.

Primjenom *Material Web* komponenti postignuto je kvalitetnije korisničko iskustvo. Animacije i različita stanja komponenti doprinose boljoj interakciji s korisnicima, čime aplikacija postaje intuitivnija i estetski ugodnija. Zbog svoje fleksibilnosti i mogućnosti prilagodbe, *Material Design* može se koristiti u raznim vrstama aplikacija, pružajući rješenja za moderan, jednostavan i funkcionalan dizajn.

LITERATURA

- [1] Kahoot službena web stranica, <https://kahoot.com/> 16.6.2024.
- [2] Quizzland aplikacija, <https://play.google.com/store/apps/details?id=com.xmonetize.quizzland> 16.6.2024.
- [3] Brittanica službena web stranica, <https://www.britannica.com/quiz/browse> 17.6.2024.
- [4] TriviaCrack aplikacija, <https://play.google.com/store/apps/details?id=com.etermax.preguntados.lite> 17.6.2024.
- [5] Službena Material Design 3 dokumentacija, <https://m3.material.io/> 18.6.2024
- [6] A. Murtaza, Material Design 1 vs Material Design 2 vs Material Design 3, Creative Tim, 2023., dostupno na: <https://www.creative-tim.com/blog/web-development/material-design-comparison/> 18.6.2024.
- [7] R. Wieruch, The Road to React: Your journey to master plain yet pragmatic React.js, Nezavisno objavljeno, 2024.
- [8] J. Duckett, PHP & MySQL: Server-side Web Development, Wiley, 2024.
- [9] J. S. Jensen, The Missing Bootstrap 5 Guide: Customize and extend Bootstrap 5 with Sass and JavaScript to create unique website designs, Packt Publishing, 2022.
- [10] Material Theme Builder, <https://material-foundation.github.io/material-theme-builder/> 10.9.2024.

SAŽETAK

U ovom završnom radu opisan je postupak izrade kviz aplikacije pomoću paketa *Material Web*. Aplikacija je uspoređena sa drugim dostupnim aplikacijama, te su zatim ukratko opisane tehnologije koje su se koristile pri izradi aplikacije. Također je napravljena usporedba sa prethodnom inačicom *Material Design* smjernica. Definirani su funkcionalni i nefunkcionalni zahtjevi aplikacije i baza podataka koju koristi aplikacija. Klijentski dio aplikacije izrađen je pomoću *React.js* razvojnog okvira koji omogućuje uporabu komponenti za dijelove korisničkog sučelja, a poslužiteljski dio aplikacije izrađen je pomoću programskog jezika *PHP* koji omogućuje komunikaciju s bazom podataka. Komponente korištene u aplikaciji su opisane po poglavljima. Nakon opisa komponenti prikazan je primjer prilagodbe izgleda *Material Web* komponenti. Nakon toga prikazan je izgled gotove aplikacije.

Ključne riječi: kviz aplikacija, material web, responzivni dizajn, web aplikacija

ABSTRACT

Creating a web quiz using the Material Design 3 package

This thesis describes the process of developing a quiz application using the Material Web package. The application is compared with other available applications, and the technologies used in the development process are briefly described. A comparison with the previous version of the Material Design guidelines is also made. The functional and non-functional requirements of the application, as well as the database it uses, are defined. The client-side of the application was developed using the React.js framework, which allows the use of components for parts of the user interface, while the server-side was developed using the PHP programming language, which is used for communication with the database. The components used in the application are described chapter by chapter. After the component descriptions, an example of customizing the appearance of Material Web components is presented. Finally, the appearance of the finished application is shown.

Key words: quiz application, material web, responsive design, web application