Izrada web aplikacije za narudžbe u ugostiteljstvu preko QR koda

Škifić, Viktor

Sveučilište u Rijeci – Odjel za informatiku

Preddiplomski sveučilišni studij informatike (jednopredmetni)

Viktor Škifić

Izrada web aplikacije za narudžbe u ugostiteljstvu preko QR koda

Završni rad

Mentor: doc. dr. sc. Lucia Načinović Prskalo

Rijeka, rujan 2021.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 2.5.2021.

Zadatak za završni rad

Pristupnik: Viktor Škifić

Naziv završnog rada: Izrada web aplikacije za narudžbe u ugostiteljstvu preko QR kôda

Naziv završnog rada na eng. jeziku: Development of a web application for orders in the catering industry via QR code

Sadržaj zadatka: Glavni zadatak završnog rada je izraditi web aplikaciju za naručivanje proizvoda u ugostiteljskim objektima putem QR kôda i pregled narudžbi. Pritom će se koristiti odabrani alati za razvoj frontend i backend dijela aplikacije koji će se u radu i detaljno opisati. Također će se opisati i demonstrirati svi važni elementi i funkcionalnosti izrađene web aplikacije.

Mentor

Doc. dr. sc. Lucia Načinović Prskalo

Načinović Prskalo

Voditelj za završne radove

doc. dr. sc. Miran Pobar

Zadatak preuzet: 2.5.2021.

Viktor Škifić

(potpis pristupnika)

Sadržaj

Sažetak

U ovome radu predstaviti ću Web aplikaciju za narudžbe u ugostiteljskim objektima preko QR kodova. Aplikacija je sastavljena od dva dijela; stranice za goste kojemu se pristupa skeniranjem QR kodova i zaštićene stranice za zaposlenike lokala. Cilj aplikacije je olakšanje i ubrzanje procesa zaprimanja narudžbi, kao i lakši pregled i praćenje stanja pojedine narudžbe. Ovom aplikacijom pokušava se umanjiti i pojednostaviti obujam posla poslužitelja u ugostiteljskim objektima.

Za izradu aplikacije korištene su sljedeće tehnologije: Visual Studio Code kao razvojno okruženje, NextJS razvojni okvir za *frontend* dio aplikacije, Firebase platforma za *backend* dio aplikacije, Typescript kao programski jezik u kojem je aplikacija napisana i ChakraUI za dizajn aplikacije.

Svi navedeni alati i funkcionalnosti aplikacije su detaljnije opisani u nastavku rada.

Ključne riječi: Web aplikacija, NextJS, ReactJS, Firebase, Typescript, Visual Studio Code, ChakraUI, narudžbe

1. Uvod

Web aplikacije su aplikacije kojima se pristupa putem web-a koristeći Internet ili Intranet mrežu [1]. One su klijent-server programi što znači da su napravljene od klijentskoga i poslužiteljskoga dijela. Klijentski dio Web aplikacija odnosi na dio aplikacije koji se (najčešće) izvršava u pregledniku korisnika. Korisnik ima pristup aplikaciji te ju koristi za neki rad, na primjer stvaranje, brisanje i mijenjanje podataka. Poslužiteljski dio aplikacije odnosi se na dio aplikacije koji se izvršava na nekom udaljenom računalu. Ono prati rad korisnika te izvršava pripadajuće operacije koje ovise o korisnikovim akcijama.

Od posebne važnosti za ovaj rad su Web aplikacije koje spadaju pod domenu E-trgovina. To su aplikacije čija je svrha mogućnost naručivanja dobara putem interneta. Zbog jednostavnosti i manjka truda koji je potreban za obavljanje kupnje, ljudi se sve češće okreću online kupovini. U 2014. godini prihodi od online trgovina na svjetskoj razini iznosili su 1.3 bilijardi dolara, te neka predviđanja govore da će se do 2024. godine ta brojka popeti do 6.3 bilijardi dolara [2]. Na osnovu ovih vrijednosti lako je zaključiti da E-trgovine imaju ogroman potencijal, koji svake godine raste.

Druga stvar koja je značajna za ovaj rad su QR (Quick Response) kodovi. Oni su dvodimenzionalni barkodovi izumljeni 1994. godine od strane japanske automobilske tvrtke Denso Wave. Mogu se koristiti za različiti spektar primjena; proširena stvarnost, prikaz multimedijskih sadržaja te naručivanje u ugostiteljskim objektima. Upotreba QR kodova u ugostiteljskim objektima ubrzava proces naručivanja, smanjuje se obujam rada konobara i mogućnost greške prilikom zaprimanja narudžbi. U isto vrijeme, u doba COVID krize, bitna je činjenica da se smanjuje količina bliske interakcije između konobara i gostiju te se samim time smanjuje mogućnost prijenosa virusa.

2. Tehnologije

Tehnologije korištene pri izradi ovoga rada mogu se podijeliti na *Frontend* i *Backend* tehnologije. *Frontend* tehnologije se odnose na tehnologije korištene za izradu klijentskoga dijela aplikacije, a *Backend* tehnologije za izradu serverskoga dijela aplikacije.

Softverski okvir (*framework*) je skup implementacija često korištenih funkcija i procedura korištenih pri razvoju aplikacija. Upotreba dobro definiranog aplikacijskog programskog sučelja (API) unutar *framework-a* pojednostavljuje i ubrzava izradu aplikacije. Programer ne mora brinuti o implementaciji određenih funkcija, te se može usredotočiti na pružanje boljeg korisničkog iskustva [3].

2.1 Frontend tehnologije

1) NextJS

NextJS[4] je *open source* JavaScript *framework* koji se koristi za izradu statičkih web stranica, kao i za izradu web aplikacija pomoću React knjižnice. Najveća prednost korištenja Next-a je mogućnost korištenja *Server Side Rendering-a* [5] i *Static Site Generation-a* [6] „*out of the box*“ što ubrzava inicijalno učitavanje stranice te time poboljšava korisničko iskustvo i pomaže našoj stranici da bolje rangira na Google-ovim rezultatima pretraživanja [7]. Osim toga, NextJS pruža podršku za kreiranje stranica sa dinamičkim rutama, izradu rasporeda stranice za višekratnu uporabu, automatsku optimizaciju slika i više [8]. Kao dokaz kvalitete NextJS-a kao JavaScript *framework-a* dovoljno je samo pogledati koje kompanije ga koriste za svoje *Frontend* potrebe; samo neke od njih su Netflix, Twitch i TikTok.

2) ChakraUI

Malo ljudi na ovome svijetu voli pisati CSS, a ja sigurno ne pripadam u tu skupinu. Stoga sam se za potrebe dizajna aplikacije odlučio koristiti ChakraUI[9]. Chakra je knjižnica gotovih React komponenti koja se trudi biti što je više moguće fleksibilna, odnosno developeru daje veliku razinu slobode u implementaciji dizajna. S drugim knjižnicama komponenti, na primjer

MaterialUI, po samom izgledu stranice može se vidjeti da se za njenu izradu koristio MaterialUI. To ne mora nužno biti loša stvar, ali ako je cilj jedinstveniji izgled stranice onda je Chakra sigurno bolji izbor.

2.2 Backend tehnologije

1) Firebase

Firebase [10] je Google-ova platforma koja u sebi sadrži mnoštvo usluga koje su potrebne developerima za izradu serverskoga dijela aplikacija, ali koje radije ne bi razvijali od nule. To uključuje usluge kao što su autentikacija, spremište podataka (Storage), baza podataka (Firestore ili Realtime Database), analitika (Google Analytics) i slično. Sve usluge su poslužene u „*cloud-u*“, te im se pristupa iz klijentskoga dijela aplikacije preko client SDK-a [11] (Software Development Kit – kolekcija alata za razvoj softvera unutar jednog instalacijskog paketa). Upotrebom Firebase-a eliminira se potreba za izgradnjom tradicionalnog *backend* sustava čime se developer može u potpunosti posvetiti boljem korisničkom iskustvu i boljoj funkcionalnosti aplikacije. Još jedna prednost je ta što se skaliranje sustava događa s malo ili nimalo truda od strane developera, već se za to brinu ljudi iz Google-a.

2.3 Ostale tehnologije

Iako sam tehnologije za izradu aplikacije podijelio na *Frontend* i *Backend* tehnologije, one nisu jedine korištene tehnologije prilikom izrade. U ovome dijelu volio bih spomenuti još tri.

1) Typescript

Typescript [12] je snažno tipiran programski jezik izgrađen na Javascript-u od strane Microsoft-a. Kada sam prvi put čuo za Typescript pomislio sam „Zašto bi itko htio uvoditi tipove u Javascript?“. Odgovor na svoje pitanje dobio sam kada sam prvi put probao napraviti malo složeniju aplikaciju u Javascript-u. Kada aplikacija dosegne određenu veličinu teško je pamtiti koju vrijednost očekujemo od neke operacije i koji oblik podataka nam je potreban za izvođenje neke operacije, što postaje još gore ako uzmemo pauzu od nekog projekta. Uvođenjem tipova ti problemi su riješeni jer unaprijed znamo što nam je potrebno za operaciju i što ćemo dobiti kao rezultat. Usto, kako se Typescript mora kompilirati u Javascript prije izvršavanja, dobivamo mogućnost otkrivanja pogrešaka prije samog izvođenja programa.

2) Visual Studio Code

Za pisanje programskoga koda potreban nam je *code editor*. Visual Studio Code [13] je *open source code editor* razvijen od strane Microsofta. Ono što ga, po mojem mišljenju, razdvaja od ostalih *editora* je mogućnost proširivanja funkcionalnosti s proširenjima treće strane. Osim funkcionalnosti koje editor nudi od početka rada, moguće je preuzeti proširenja koja pomažu u radu ili čine cjelokupno iskustvo programiranja ugodnijim, jedno od kojih bi bio *formatter* koji prilikom svakog spremanja automatski formatira kod. Uz to, VS Code nudi skoro beskonačne mogućnosti prilagođavanja, dobro odrađenu mogućnost programiranja u paru i ugrađene Git naredbe.

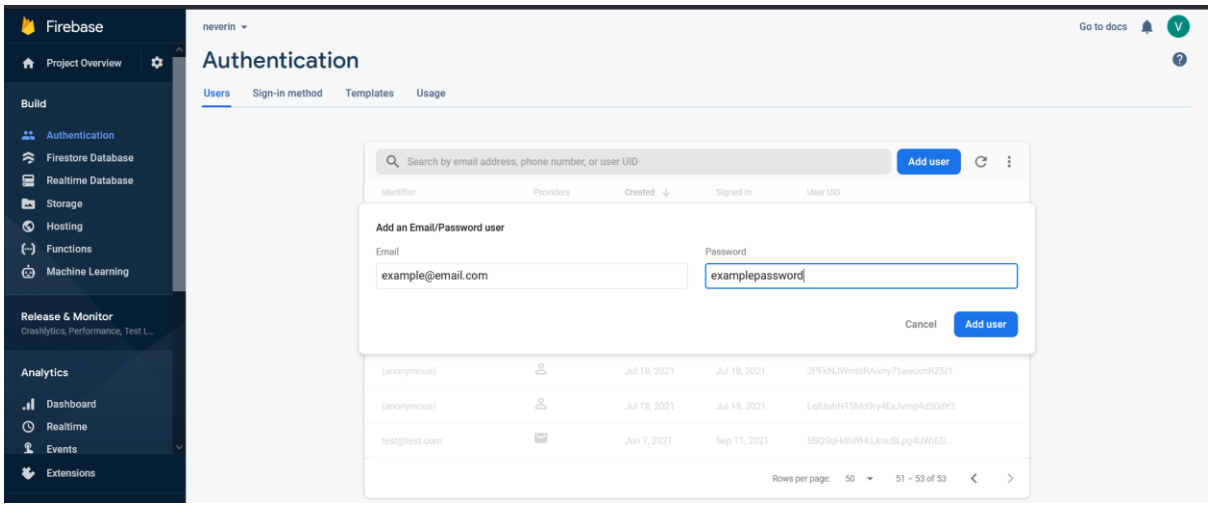
3) QR kodovi

Kao što sam spomenuo u uvodu, QR kodovi su ključni elementi za rad ove aplikacije. Gost lokala skenira QR kod na kojem se nalazi adresa dijela aplikacije namijenjenog za goste. Sama implementacija je bila prilično jednostavna jer postoji veliki broj usluga koje generiraju QR kodove sa željenim informacijama, te ih je potrebno samo generirati i ispisati.

3. Opis elemenata i funkcionalnosti aplikacije

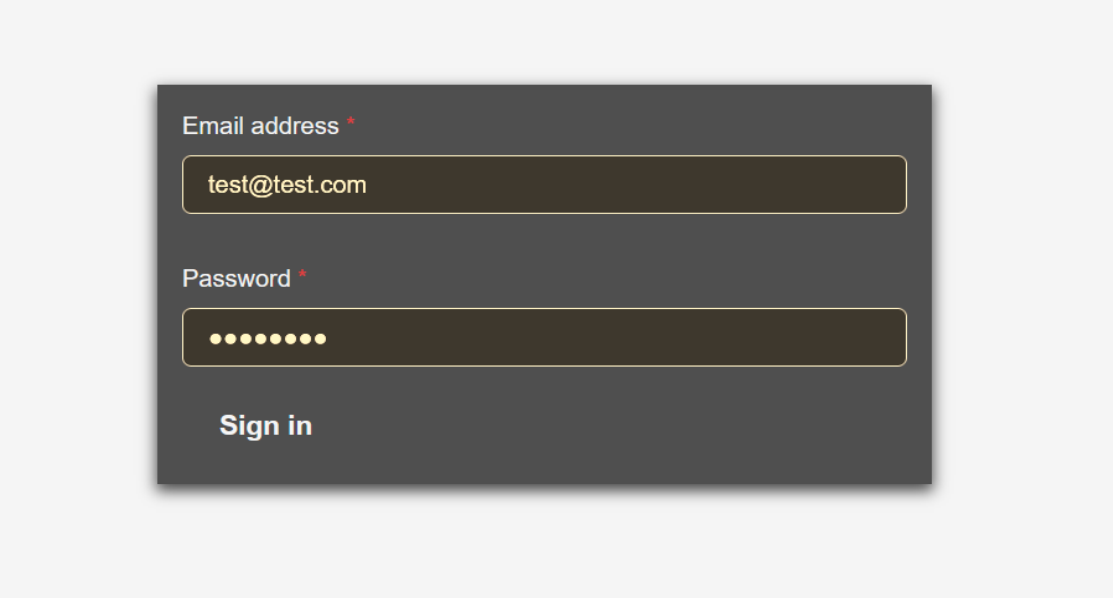
3.1 Autentikacija

Kako je ova aplikacija zamišljena primarno kao stvarni poslovni proizvod, ona nema mogućnost samostalne registracije korisnika. Nakon dogovora o kupnji usluge, vlasnik lokala mi daje E-mail adresu i lozinku koji će biti povezani s njegovim računom. E-mail adresa i lozinka se u Firebase konzoli unose u kartici *Authentication* te se tako stvara novi korisnik što je prikazano u Slici 1.



Slika 1: Dodavanje korisnika

Na *home page-u* aplikacije https://neverin.vercel.app nalazi se gumb *Login* koji navigira korisnika na *login page*. Tamo se nalazi jednostavna forma za unos E-mail adrese i lozinke koju korisnik mora ispuniti, te ako su uneseni podaci točni, on ulazi u dio aplikacije namijenjen zaposlenicima lokala. Navedena forma prikazana je u Slici 2.



Slika 2: Login forma

Implementacija autentikacije unutar aplikacije je odrađena kroz React-ov *Context*. U React aplikacijama je običaj da svaka komponenta brine o svom stanju (objekt koji u sebi sadrži komplet svojstava koje kontroliraju ponašanje komponente[14]) koje je napravljeno po principu *one-way data binding-a*, što znači da ono može teći samo niz stablo komponenti. Međutim, što ako više komponenti treba imati pristup istom stanju? Za takve slučajeve postoje knjižnice treće strane kao što su Redux i Mobx, ali radi jednostavnosti implementacije ja sam se odlučio za korištenje *context-a*. *Context* služi kao spremnik stanja kojemu se može pristupiti iz bilo kojeg dijela aplikacije koji u stablu komponenti dolazi nakon deklaracije *context-a*. U *context-u* vezanom za autentikaciju nalazi se varijabla *user* vezana za trenutno prijavljenog korisnika i funkcije *login*, *logout* i *loginAnonymously*.

Prijavljivanje i odjavljivanje korisnika odrađuje komponenta *AuthButton* čiji kod je prikazan u Slici 3. Ono što će komponenta raditi ovisi o tome je li korisnik trenutno prijavljen.

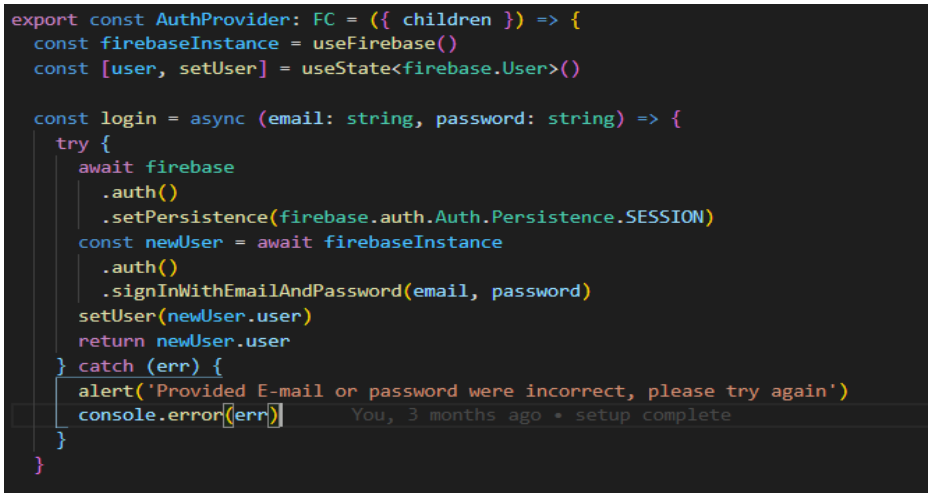
Ako korisnik nije prijavljen, na gumbu se ispisuje tekst „Sign in“. Pritiskom na gumb izvršava se *login* funkcija iz *auth* context-a koja je prikazana na Slici 4. *Login* funkcija postavlja *persistence* na opciju *session* što znači da će prilikom zatvaranja kartice u kojoj se nalazi aplikacija korisnik biti automatski odjavljen. Ovo je napravljeno iz sigurnosnih razloga. Nakon toga izvršava se Firebase-ova metoda *signInWithEmailAndPassword(email, password)*, te se nakon njenog uspješnog izvršavanja postavlja trenutno prijavljeni korisnik koji se zatim

preusmjerava na stranice lokala. Ako je izvršavanje funkcije neuspješno prikazuje se *alert* prozor, te se u konzolu ispisuje razlog pogreške.

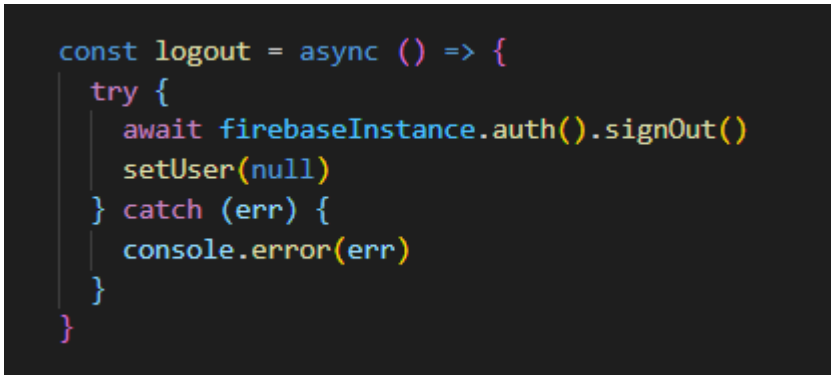
Ako je korisnik prijavljen, na gumbu se ispisuje tekst „Sign out“. Pritiskom na gumb izvršava se funkcija *logout* koja pomoću Firebase-ove *signOut()* metode odjavljuje korisnika i postavlja ga na *null* vrijednost i koja je prikazana na Slici 5. Korisnik će zatim biti preusmjeren na *root* stranicu aplikacije.

```
export const AuthButton: FC<Props> = ({ email, password, variant }) => {
  const [isLoading, setIsLoading] = useState<boolean>(false)
  const { login, logout, user } = useAuth()
  const router = useRouter()

  const handler = async () => {
    if (user) {
      setIsLoading(true)
      await logout()
      router.push('/')
    } else {
      setIsLoading(true)
      try {
        const user = await login(email, password)       You, 2 days ago • 
        router.push(`/shop/${user.uid}`)
      } catch (err) {
        setIsLoading(false)
        console.error(err)
      }
    }
  }

  useEffect(() => {
    setIsLoading(false)
  }, [])

  return (
    <Button
      bg='accent'
      size='lg'
      isLoading={isLoading}
      variant={variant}
      onClick={handler}
      color='white'
      _hover={{ bg: 'white', color: 'text' }}
    >
      {user ? 'Sign out' : 'Sign in'}
    </Button>
  )
}
```

Slika 3: AuthButton komponenta



Slika 4: Login funkcija

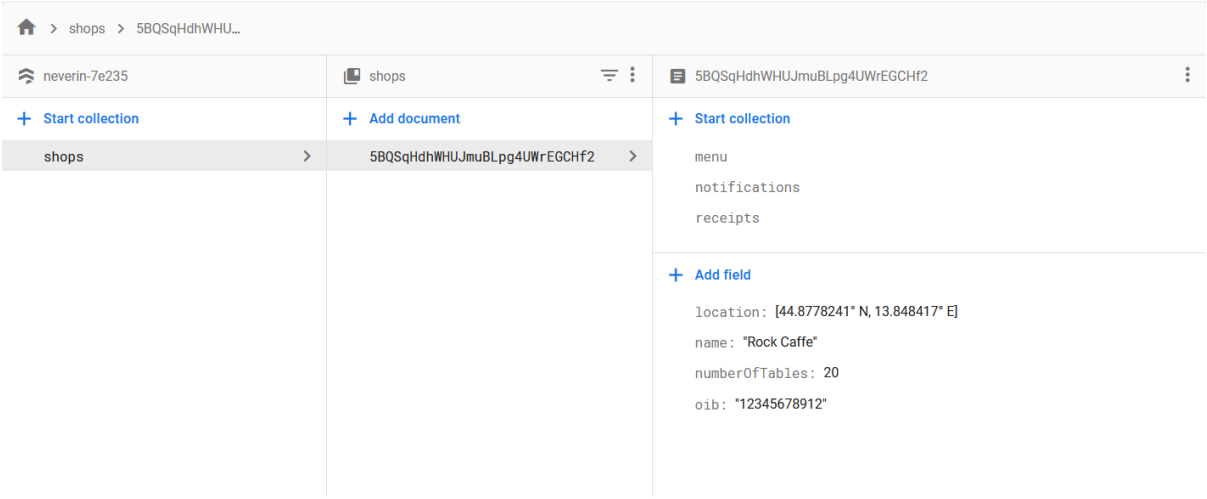


Slika 5: Logout funkcija

3.2 Struktura baze podataka

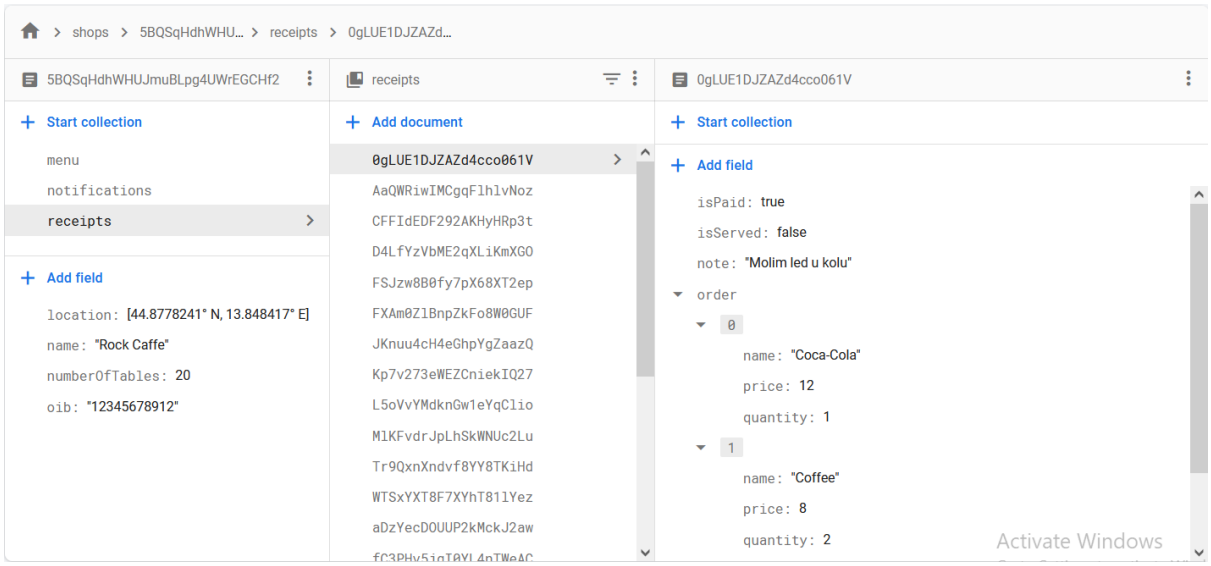
Nakon što je novi korisnik dodan u Firebase za njega se može kreirati dokument u bazi podataka.

U bazi se nalazi jedna kolekcija naziva *shops* koja u sebi sadrži dokumente svih lokala. Svaki dokument u sebi sadži podatke o imenu lokala, broju stolova, OIB-u i lokaciji lokala. ID dokumenta pojedinog lokala jednak je ID-u korisnika kojemu lokal pripada. Unutar svakog lokala nalaze se podkolekcije *menu* i *receipts*. Na Slici 6 nalazi se prikaz kolekcije *shops* i osnovnih podataka o lokalu.



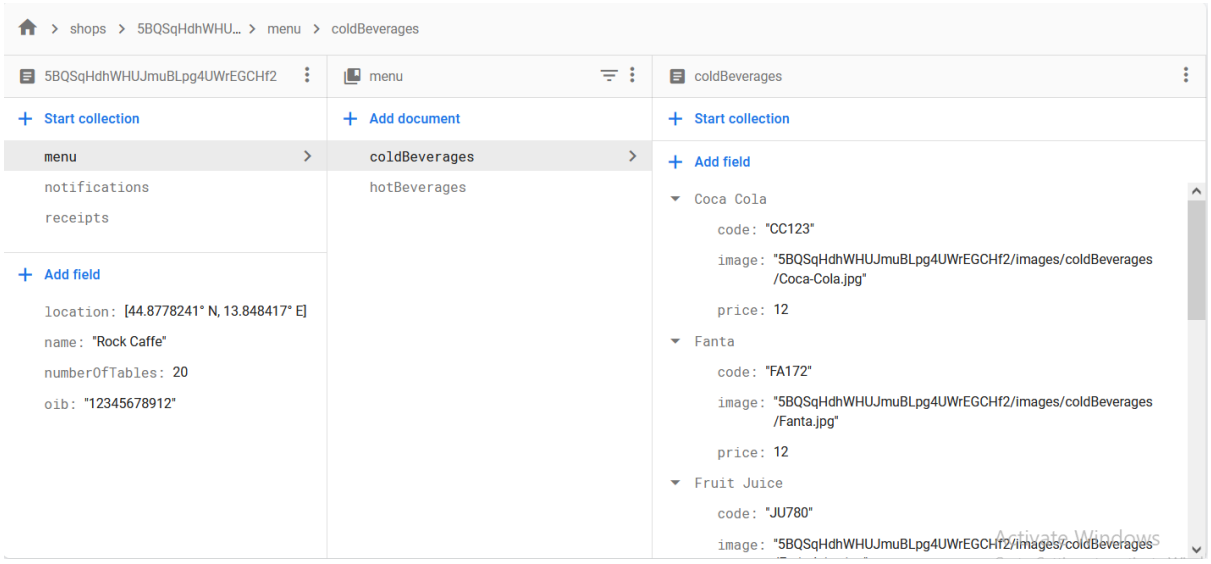
Slika 6: Baza podataka, kolekcija shops i osnovni podaci o lokalu

Podkolekcija *receipts* može biti viđena na Slici 7 i sadrži podatke o narudžbama, to jest, računima, koji se spremaju kao jedinstveni dokumenti. Računi su sastavljeni od polja *table* koje govori koji stol je napravio narudžbu, *timestamp* za praćenje vremena narudžbe, *note* za zamolbe konobarima, *isPaid* i *isServed* koji prate je li račun plaćen odnosno poslužen, *order* u kojeg se spremaju naručeni artikli koji je oblika *array*, te *total* sa konačnim iznosom računa.



Slika 7: Podkolekcija receipts

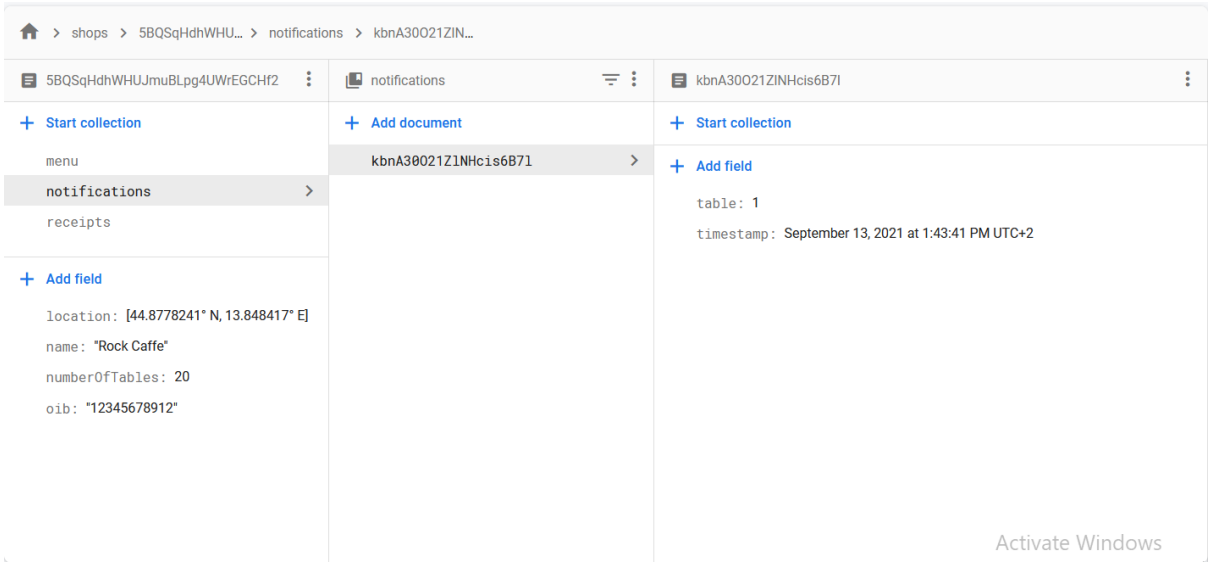
Podkolekcija *menu* prikazana na Slici 8 u sebi sadrže dokumente sa podacima pojedinih artikala. Naziv svakog dokumenta opisuje kategoriju artikla koji se u njemu nalazi, na primjer *Hot beverages* ili *Cold beverages*. Unutar dokumenata artikli su organizirani u objekte čiji su nazivi jednaki imenima artikala. Svaki artikal ima cijenu, šifru artikla i URL slike artikla ukoliko ona postoji. Slike artikala se spremaju u Firebase-ov Storage.



Slika 8: Podkolekcija menu

Tijekom rada uvjetno se stvara još jedna podkolekcija *notifications* prikazana u Slici 9. Ta podkolekcija se ne nalazi u bazi podataka u svakom trenutku jer Firebase automatski briše prazne kolekcije. Gost ima mogućnost pozvati konobara nakon čega se stvara podkolekcija *notifications* ako već ne postoji. Ona sadrži dokumente od kojih svaki predstavlja jedan zahtjev, a u sebi sadrži podatke o stolu koji je napravio zahtjev i vrijeme kada je zahtjev napravljen.

Zaposlenik može potvrditi da je zahtjev obavljen čime se dokument vezan za taj zahtjev briše iz podkolekcije.



Slika 9: Podkolekcija notifications

3.3 Stranica za zaposlenike

Stranice lokala imaju adrese oblika shop/[id] gdje je id oznaka dokumenta lokala iz baze podataka. Ovakav oblik je implementiran kroz NextJS-ovu podršku za dinamičke rute stranica. Unutar *pages* direktorija u strukturi projekta deklariraju se stranice koje će aplikacija sadržavati. Kako bi se napravile stranice sa dinamičkim rutama potrebno je napraviti poddirektorije, u mom slučaju to su direktoriji *guest* i *shop*. Unutar njih nalaze se datoteke naziva [id].tsx što govori NextJS-u da stranice servira na rutama guest/[id], odnosno shop/[id].

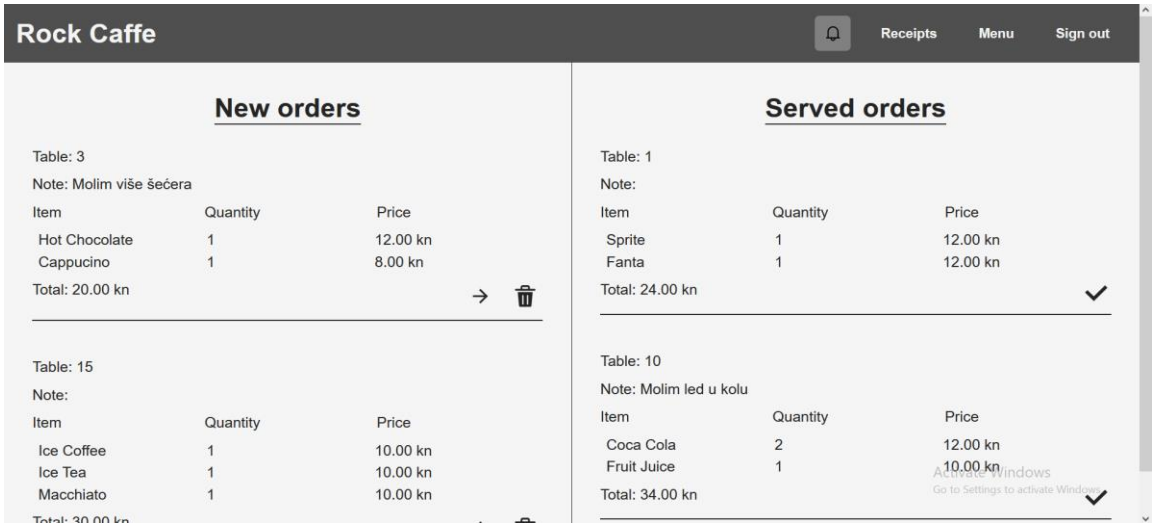
Prilikom učitavanja stranice izvršava se NextJS funkcija *getServerSideProps*. Ona nam omogućava *pre-render* stranice podacima koji su se u njoj dohvatili. Slika 10 prikazuje funkciju *getServerSideProps.*

```
export const getServerSideProps: GetServerSideProps<Props, Params> = async ({
  params,
}) => {
  const shopQueryResult = await firebaseInstance
    .firestore()       You, 3 months ago • server side props
    .collection('shops')
    .doc(params.id)
    .get()

  const newShop = shopQueryResult.data() as FShop

  return {
    props: {
      shop: {
        id: params.id,
        ...newShop,
        location: {
          lat: newShop.location.latitude,
          long: newShop.location.longitude,
        },
      },
    },
  }
}
```

Slika 10: getServerSideProps funkcija stranice za zaposlenike

getServerSideProps kao parametar prima *context* objekt koji u sebi sadržava ključeve *params, query, req, res* i još nekoliko njih. U ovom slučaju potreban nam je *params* ključ koji će u sebi sadržavati ID dokumenta iz baze koji se odnosi na neki lokal. Dobiveni ID se proslijeđuje Firebase *get* metodi i dobiveni podaci se spremaju u varijablu. Funkcija mora vraćati objekt oblika *props: {...}* u kojem se dobiveni podaci prosljeđuju komponenti od koje je stranica napravljena.

Dizajn stranice je prikazan u Slici 11, te je vrlo jednostavan jer mi je cilj bila preglednost narudžbi kako bi se zaposlenicima olakšao posao. U *Header* komponenti nalaze se naziv lokala, gumb za obavijesti, gumb *receipts* koji otvara *drawer* komponentu s popisom svih plaćenih računa, gumb *menu* koji otvara modalni prozor sa akcijama vezanim uz menu i *Sign out* gumb opisan ranije. Tijelo stranice čine popisi tek pristiglih narudžbi koje još nisu poslužene i posluženih narudžbi koje čekaju plaćanje.



Slika 11: DIzajn stranice za zaposlenike

```
const Shop: NextPage<Props> = ({ shop }) => {
  return (
    <>
      <ReceiptsProvider shopId={shop.id}>
        <Header shopName={shop.name} />
        <Orders />
      </ReceiptsProvider>
    </>
  )
}
```

Slika 12: Komponenta stranice za zaposlenike

Stranica je napravljena od dvije komponente, *Header* i *Orders*, omotane *ReceiptsProvider* komponentom koja komponente ispod sebe puni podacima iz *receipts contexta*. Ovaj *context* služi za čuvanje stanja o narudžbama i operacijama vezanim uz iste. U njega se prosljeđuje *prop* ID koji je potreban za izvršavanje akcija u bazi. U *Header* komponentu se prosljeđuje ime lokala za ispis. Prikaz koda stranice nalazi se na Slici 12.

Prilikom prvog *rendera Header* komponente postavlja se pretplata na *notifications* kolekciju u bazi podataka. Ova akcija izvršava se u React-ovom *useEffect hook-u* čiji primjer se nalazi na Slici 13. *useEffect hook* služi za nuspojave vezane uz promjene stanja aplikacije koje zahtijevaju *re-render*, međutim može se koristiti za dohvaćanje podataka ili postavljanje pretplati. Kao parametre prima funkciju koja će se izvršiti prilikom ažuriranja podataka i *dependency array*. *Dependency array* govori *useEffectu* na koje varijable treba obratiti pozornost prilikom ažuriranja, te će se izvršiti prilikom promjene tih varijabli. Postoji i opcija proslijeđivanja praznog *dependency array-a* što znači da će se *useEffect* izvršiti samo pri prvom *render-u* i prilikom *unmount-a* komponente. Još jedna opcija prilikom korištenja *useEffect-a* je *cleanup* funkcija. Ako je deklarirana, ona se izvršava prilikom *unmount-a* komponente i služi „pospremanju“ kada se komponenta više ne koristi, što je dobro za gašenje pretplata koje više nisu potrebne.

```
const { shopId } = useReceipts()
useEffect(() => {
  const unsubscribe = firebaseInstance
    .firestore()
    .collection('shops')
    .doc(shopId)
    .collection('notifications')
    .onSnapshot((notifications) =>
      notifications.docChanges().map((change) => {
        if (change.type === 'added') {
          setNotify(true)
        }
      })
    )
  return () => unsubscribe()
}, [])
```

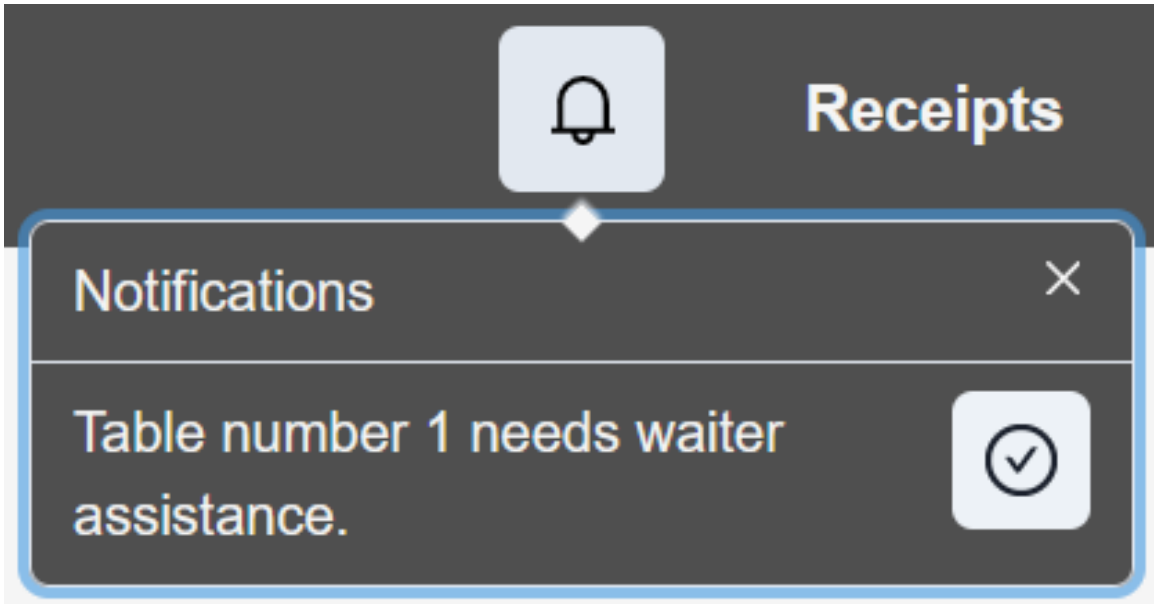
Slika 13: Primjer useEffect hook-a i onSnapshot metode

Za postavljanje same pretplate koristi se Firebase-ova *onSnapshot* metoda koja će na kolekciju *notifications* postaviti *event listener* i slati podatke o promjenama u kolekciji. Ako je tip

promjene *„changed“* stanje varijable *notify* se promijeni iz *false* u *true* zbog čega gumb za obavijesti svijetli crvenom bojom.

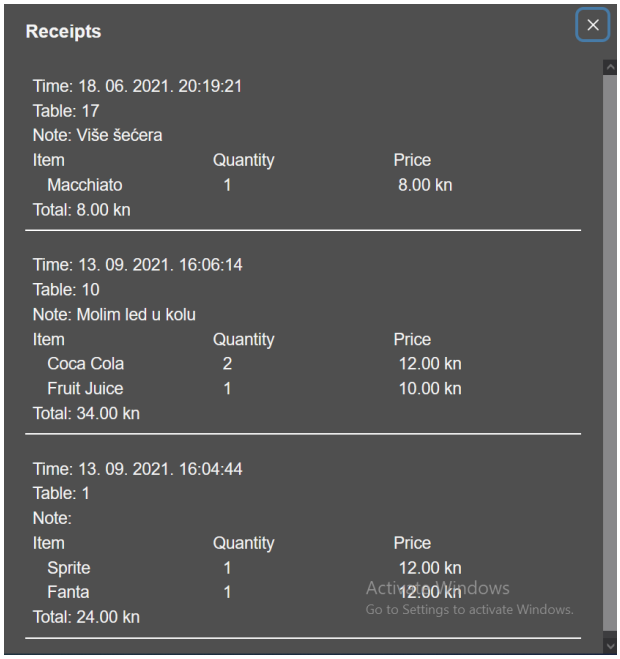
Klikom na gumb obavijesti otvara se *Popover* komponenta iz ChakraUI knjižnice komponenti koja u sebi sadrži *Notifications* komponentu. U njoj se ispisuju podaci o obavijestima, odnosno zahtjevima gostiju za pomoći konobara. Izgled komponente nakon pritiska gumba prikazan je na Slici 14.

Notifications komponenta u sebi također sadrži pretplatu na *notifications* kolekciju kako bi mogla ispisati nove zahtjeve. Pored svakog zahtjeva nalazi se gumb čijim se pritiskom potvrđuje da je obavijest viđena, te se ona briše iz kolekcije.



Slika 14: Obavijesti za zaposlenike

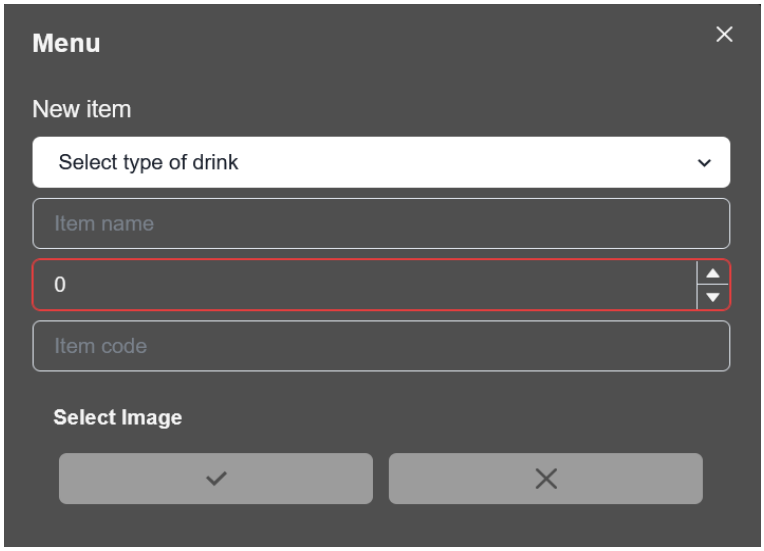
Pored obavijesti nalazi se gumb *Receipts* čijim se klikom otvara ChakraUI *Drawer* komponenta, sadržaj gumba je prikazan na Slici 15. Unutar *Drawer-a* nalazi se *Receipts* komponenta koja služi ispisivanju svih plaćenih računa. Računi se dohvaćaju iz *receipts context-a* i prilikom ispisa sortiraju u padajućem redoslijedu prema vremenu kada je narudžba zaprimljena.



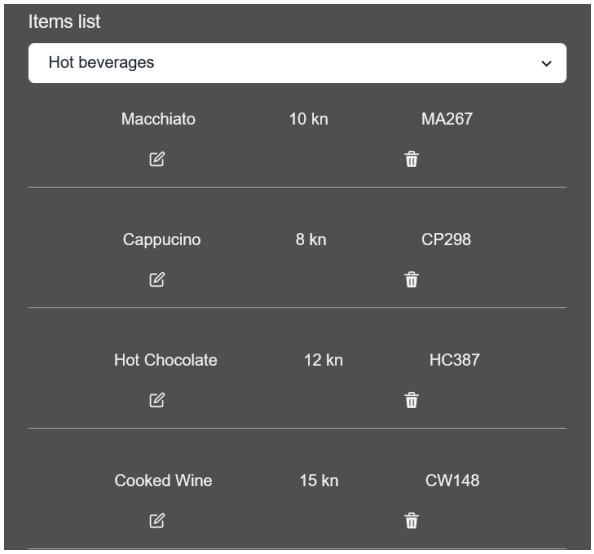
Slika 15: Receipts komponenta

Gumb *Menu* služi za pregled menu-a lokala i dodavanje novih artikala. Klikom na gumb otvara se modalni prozor sa ponuđenim radnjama. Na vrhu prozora nalazi se *New item* sekcija pomoću koje se u menu mogu dodati novi artikli. Prvo je potrebno odabrati kategoriju artikla, a zatim ispuniti polja o imenu, cijeni i šifri artikla, te postoji opcija dodavanja slike. Ukoliko nije odabrana kategorija ili jedno od polja nije ispunjeno, dodavanje artikla je onemogućeno. Prikaz sekcije za dodavanje artikala nalazi se na Slici 16.

Ispod toga nalazi se sekcija sa pregledom postojećeg menu-a. Za pregled je potrebno odabrati kategoriju artikala nakon čega se pojavljuje popis artikala koji spadaju pod tu kategoriju. Uz svaki artikal nalaze se gumbi za ažuriranje i brisanje. Pritiskom na gumb za ažuriranje pojavljuju se polja za unos podataka kojima će se artikal ažurirati koja su unaprijed popunjena već postojećim vrijednostima. Prikaz sekcije za pregled postojećih artikala nalazi se na Slici 17.



Slika 16: Menu komponenta, dodavanje novog artikla



Slika 17: Menu komponenta, listanje postojećih artikala

Glavni sadržaj stranice odnosi se na zaprimljene narudžbe. U dva stupca ispisuju se narudžbe koje tek treba poslužiti i narudžbe koje su posluženе i čekaju plaćanje.

Sadržaj je implementiran unutar *Orders* komponente koja dobiva stanje narudžbi iz *receipts context-a,* komponenta je prikazana na Slici 18. Narudžbe se dohvaćaju iz baze pomoću pretplate na kolekciju *receipts* i spremaju u varijablu *orders*. Unutar *Orders* komponente one se sortiraju rastućim redoslijedom prema vremenu zaprimljene narudžbe, te mapiraju u jedan od stupaca ovisno o stanju *isServed* varijable. Podaci o svakoj narudžbi se zatim šalju u *Order* komponentu koja te podatke ispisuje. Ukoliko narudžba još nije poslužena, u njoj se nalaze dva gumba. Pritiskom na prvi gumb mijenja se stanje *isServed* varijable te narudžbe te ona prelazi u stupac posluženih narudžbi. Drugi gumb briše narudžbu. Ako je narudžba poslužena gubi se mogućnost brisanja narudžbe, te se prikazuje samo gumb koji potvrđuje da je narudžba plaćena. Klikom na njega mijenja se stanje *isPaid* varijable, te se ona prestaje prikazivati u ijednom od dva stupca i ispisuje se u ranije opisanoj *Receipts* komponenti.

Slika 18: Zaprimljene i poslužene narudžbe

3.4 Stranica za goste

Kao što je već spomenuto, stranice za goste imaju adrese oblika guest/[id], ali sa dodatkom *query* parametra *table*. Brojevi stolova su zapisani u adresi ispisanoj na svakom QR kodu kako bi se znalo koji stol je napravio koju narudžbu. Parametar *table* se pomoću *getServerSideProps* funkcije dohvaća iz adrese, te se uz ID lokala i lokaciju lokala šalje komponenti koja čini stranicu. Na Slici 19 nalazi se kod *getServerSideProps* funkcije stranice za goste.

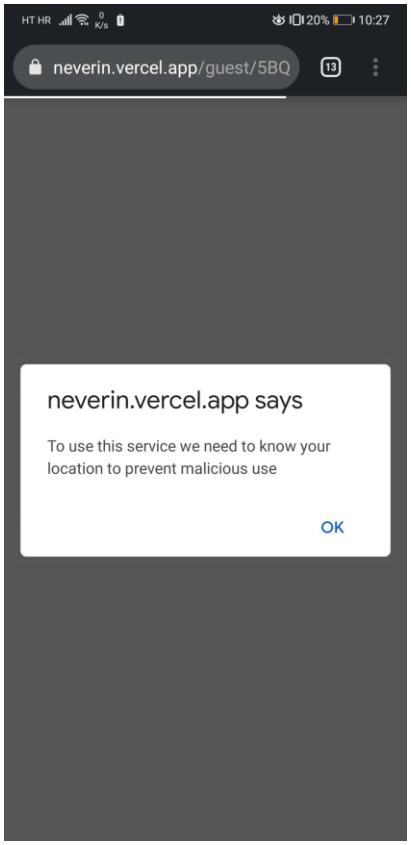
```
export const getServerSideProps: GetServerSideProps<Props, Params> = async ({
  params,
  query,
}) => {
  const { table } = query as Query

  const shop = await firebaseInstance
    .firestore()
    .collection('shops')
    .doc(params.id)
    .get()
  const location = {
    lat: shop.data().location.latitude,
    long: shop.data().location.longitude,
  }

  return {
    props: {
      shopId: params.id,
      table: parseInt(table),
      location,
    },
  }
}
```

Slika 19: getServerSideProps funkcija stranice za goste

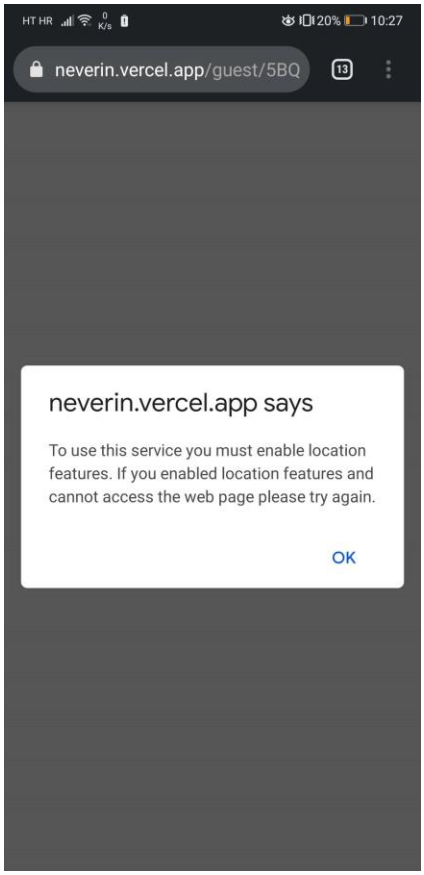
Pri učitavanju stranice korisniku se prikazuje poruka o provjeri njegove trenutne lokacije koja je prikazana na Slici 20. Ovo je implementirano kako bi se spriječile lažne narudžbe nekoga tko zna adresu stranice za goste, ali se ne nalazi u lokalu. Ako usluga lokacije nije uključena, prikazuje se *alert* prozor koji obavještava gosta da ne može koristiti aplikaciju bez provjere lokacije i preusmjeruje ga na *home page* aplikacije. Poruka je prikazana na Slici 21. Za provjeru nalazi li se gost u zadanom radijusu koristio sam Haversinovu formulu [15] koja se koristi za

mjerenje udaljenosti dvije točke na sferi. Stranica se ne učitava dok provjera nije uspješno obavljena. Implementacija formule prikazana je na Slici 22.

Ako se gost doista nalazi u lokalu, stvara se novi anonimni korisnik kroz Firebase metodu *signInAnonymously* i stranica se učitava.



Slika 20: Poruka o uporabi lokacije

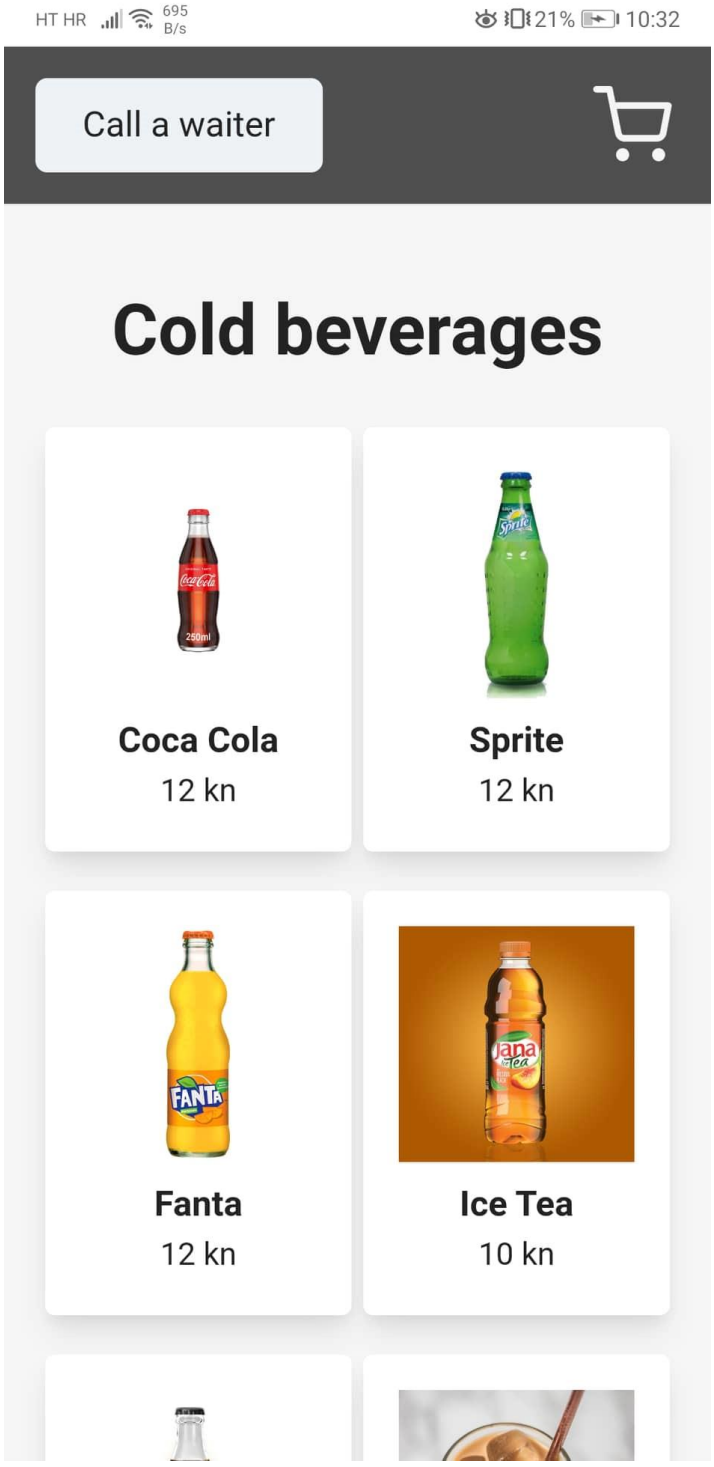


Slika 21: Neuspjeh dohvaćanja lokacije

```
const checkLocation = () => {
  alert(
    'To use this service we need to know your location to prevent malicious use'
  )
  const onSuccess: PositionCallback = (e) => {
    function getDistanceFromLatLonInM(lat1, lon1, lat2, lon2) {
      const R = 6371 // Radius of the earth in km
      const dLat = deg2rad(lat2 - lat1) // deg2rad below
      const dLon = deg2rad(lon2 - lon1)
      const a =
        Math.sin(dLat / 2) * Math.sin(dLat / 2) +
        Math.cos(deg2rad(lat1)) *
          Math.cos(deg2rad(lat2)) *
          Math.sin(dLon / 2) *
          Math.sin(dLon / 2)
      const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a))
      const d = R * c * 1000 // Distance in m
      return d
    }

    function deg2rad(deg) {
      return deg * (Math.PI / 180)      You, a month ago • uncomment location check and anonymous sign in
    }
    if (
      getDistanceFromLatLonInM(
        location.lat,
        location.long,
        e.coords.latitude,
        e.coords.longitude
      ) <= 200000
    ) {
      signIn()
    } else {
      alert('You must be in the shop to use this service')
      router.push('/')
    }
  }

  const onError = () => {
    alert(
      'To use this service you must enable location features. If you enabled location features and cannot access the web page please try again.'
    )
    router.push('/')
  }
  navigator.geolocation.getCurrentPosition(onSuccess, onError)
}
```

Slika 22: Kod za provjeru lokacije korisnika

Stranica se sastoji od *Header* komponente na kojoj se nalaze gumb *Call a waiter* i gumb u obliku košarice za pregled narudžbe. U tijelu stranice prikazan je menu lokala, sa sekcijama odvojenim ovisno o kategoriji artikala, na primjer *Hot beverages* ili *Cold beverages*. Unutar sekcija nalaze se kartice sa imenom artikla i cijenom, koje mogu, ali ne moraju sadržavati i sliku artikla. Prikaz dizajna stranice nalazi se na Slici 23.

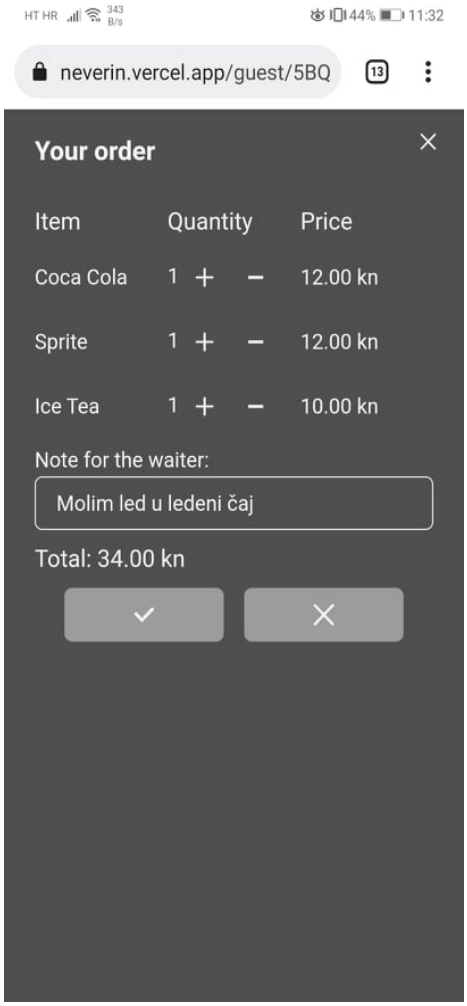


Slika 23: Dizajn stranice za goste

Pritiskom na gumb *Call a waiter* dodaje se novi dokument u kolekciju *notifications* sa podacima o broju stola koji je napravio zahtjev i vremenom zahtjeva.

Pritiskom na kartice sa artiklima, podaci o artiklu se šalju u košaricu te se na taj način stvara narudžba. Količina nekog artikla može se povećavati ponovnim pritiskom iste kartice.

Pritiskom na košaricu otvara se *Drawer* komponenta unutar koje se nalazi *Cart* komponenta koja je prikazana u Slici 24. U košarici se nalazi tablični prikaz trenutne narudžbe sa podacima o imenu, cijeni i količini naručenih artikala, te ukupnoj cijeni. Ovdje gost može dodatno namještati količinu artikala koje želi kupiti, a ako je količina jednaka nuli artikal se briše iz narudžbe. Također postoji opcija ostavljanja poruke konobaru, ali ona nije nužna. Ako gost želi odustati od narudžbe u potpunosti, to može pritiskom na gumb sa znakom X što će resetirati stanje košarice i zatvoriti košaricu. Ako je zadovoljan svojom narudžbom, pritiskom gumba sa znakom kvačice narudžbu šalje u kolekciju *receipts* kao novi dokument. Nakon izvršene narudžbe prikazuje se poruka da je narudžba uspješno provedena.



Slika 24: Cart komponenta, pregled narudžbe

4. Zaključak

Ideja za ovu aplikaciju došla je iz stvarnog zahtjeva vlasnika jednog kafića za izgradnju ove aplikacije. Možemo zaključiti da na tržištu već postoji interes za sličnim stvarima, interes koji je u doba pandemije zasigurno porastao i ostat će ovdje nakon pandemije. S obzirom na brojke o porastu internet kupovine iz uvoda ovoga rada i činjenicu da virtualno svaka osoba u džepu nosi pametni telefon, smatram da ovaj proizvod ima veliki potencijal za komercijalni uspjeh.

Imajući na umu jednostavnost aplikacije, krivulja učenja korištenja je praktički ravna crta zbog čega bi adopcija kod zaposlenika i gostiju, vjerujem, prošla prilično bezbolno. S navalom turista svakoga ljeta povećava se i obujam rada zaposlenika u čemu bi ova aplikacija mogla barem malo pomoći, što dovodi do većeg zadovoljstva svih uključenih.

Naravno, ovo nije primjer gotovoga proizvoda, već samo prikaz nečega što bi on jednoga dana mogao postati. Uz dodatne funkcionalnosti kao što su prijevodi, razne vrste filtriranja i sortiranja, te mogućnosti posebnih ponuda na određene artikle, stvarno vjerujem da bi jednoga dana ovaj proizvod mogao biti sastavni dio svakog ugostiteljskog objekta.

5. Popis literature

[1] http://www.mathos.unios.hr/wp/wp2009-10/P14_Web_aplikacije.pdf [05. rujna 2021.]

[2] D. Coppola (2021), *E-commerce worldwide – statistics and facts* dostupno na: https://www.statista.com/topics/871/online-shopping/#dossier-chapter1 [06. rujna 2021.]

[3] https://www.techopedia.com/definition/14384/software-framework [06. rujna. 2021.]

[4] D. Niżyński (2020), *What is Next JS and why you should use it in 2021?* dostupno na: https://pagepro.co/blog/what-is-nextjs/ [07. rujna 2021.]

[5] N. Showvhick, *What is SSR and Why You Need it* dostupno na: https://appradius.co/blog/what-is-ssr-react-ssr-next-js-zeit [07. rujna 2021.]

[6] https://www.cloudflare.com/learning/performance/static-site-generator/ [07. rujna 2021.]

[7] The Botify Team, *Page speed & SEO: How load time affects bots and humans differently* dostupno na: https://www.botify.com/blog/page-speed-seo [07. rujna 2021.]

[8] https://nextjs.org/docs/getting-started [08. rujna 2021.]

[9] https://chakra-ui.com/ [08. rujna. 2021.]

[10] D. Stevenson (2018), *What is Firebase? The complete story, abridged* dostupno na: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0 [08. rujna 2021.]

[11] L. Valdellon (2020), *What is an SDK? Everything you need to know* dostupno na: https://clevertap.com/blog/what-is-an-sdk/ [08. rujna 2021.]

[12] https://www.typescriptlang.org/ [09. rujna. 2021.]

[13] https://code.visualstudio.com/ [10. rujna 2021.]

[14] https://www.geeksforgeeks.org/reactjs-state-react/ [20. rujna 2021.]

[15] S. Kettle (2017), *Distance on a sphere: The Haversine formula* dostupno na: https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128 [14. rujna 2021.]

6. Popis slika