Mobilna aplikacija za uslužne objekte

Klement, Goran

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

Sveučilišni studij

Mobilna aplikacija za uslužne objekte

Diplomski rad

Goran Klement

Osijek, 2023.

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

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 18.09.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Goran Klement
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1211R, 08.10.2021.
OIB studenta:	31661793256
Mentor:	izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Tomislav Keser
Član Povjerenstva 1:	izv. prof. dr. sc. Alfonzo Baumgartner
Član Povjerenstva 2:	doc. dr. sc. Tomislav Galba
Naslov diplomskog rada:	Mobilna aplikacija za uslužne objekte
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	[Rezervirano: Goran Klement] Napraviti mobilnu aplikaciju koja ce moci skenirati QR kod koji se nalazi na nekoj lokaciji, te pouniditi korisniku usluzne sadrzaje. Korisnik moze odabrati vise razlicitih proizvoda/usluga, te ih naručiti. Također napraviti i mobilnu aplikaciju za davatelja usluga koji moze provjeriti narudzbe i naručeno onda donijeti na traženu lokaciju.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	18.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 08.10.2023.

Ime i prezime studenta:	Goran Klement
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1211R, 08.10.2021.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Mobilna aplikacija za uslužne objekte**

izrađen pod vodstvom mentora izv. prof. dr. sc. Alfonzo Baumgartner

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 studenta:

SADRŽAJ

1. UVOD

Jedno od područja gdje digitalizacija posljednjih godina nije kročila je ugostiteljstvo. Posao djelatnika u uslužnim djelatnostima godinama je isti, s mnogo koraka koji nepotrebno utroše vrijeme ili mogu dovesti do pogrešaka. Pod utroškom vremena smatra se nekoliko dolazaka uslužnih djelatnika kako bi korisnici uspješno naručili i dobili željenu narudžbu. Osim toga novijim djelatnicima se zbog nedovoljno spremnosti ili stresa mogu dogoditi pogreške u smislu krivog razumijevanja ili zabune narudžbe. Takve pogreške mogu se dogoditi i iskusnijim djelatnicima uslijed preglasne glazbe, povećanog umora, povećanog opsega posla.

Osim prethodno navedenog problem mnogih uslužnih objekata je nedovoljna preglednost. Postoji mogućnost da objekt ima strukturu koja dovodi do toga da djelatnici ne mogu ugledati nove klijente. To posljedično može dovesti do povećanja nervoze klijenata koji samim time utječu na ostale prisutne. Osim toga može se dogoditi i da klijent više ne želi čekati dolazak djelatnika te napusti uslužni objekt. Takvi događaji nisu poželjni jer djelatnik može biti nepotrebno kritiziran iako ponekad nije kriv zbog same preglednosti prostorije. To dovodi i do smanjenja profita.

1.1. Zadatak diplomskog rada

Zadatak ovoga diplomskog rada je izraditi mobilnu aplikaciju koja će omogućiti korisnicima skeniranje QR koda (engl. *quick-response code*) koji se nalazi na nekoj lokaciji te ponuditi korisniku uslužne sadržaje. Korisnik može odabrati više različitih proizvoda/usluga, te ih naručiti. Također napraviti i mobilnu aplikaciju za davatelja usluga koji moze provjeriti narudžbe i naručeno onda donijeti na traženu lokaciju.

2. USPOREDBA SA SLIČNIM RJEŠENJIMA

2.1. WaiterBolt aplikacija

WaiterBolt aplikacija je digitalna bilježnica namijenjena uslužnom osoblju na koju se jednostavno mogu dodati i prikazati narudžbe. Moguće je kreirati odvojene prostorije s potrebnim brojem stolova, kreirati vlastiti meni, pogledati povijest narudžbi, promijeniti trenutnu narudžbu. Prije kreiranja narudžbe, potrebno je dodati prostorije i stolove koje je moguće poslužiti. U aplikaciji registrirani korisnici mogu mijenjati meni, dodavati ili uklanjati proizvode te mijenjati cijenu. Osim samostalnog unosa aplikacija nudi mogućnost učitavanja dokumenta u kojem se nalazi lista proizvoda. Glavne funkcionalnosti WaiterBolt aplikacije su:

1. Kreiranje vlastitog menija.
2. Upravljanje narudžbama
3. Brzo traženje proizvoda
4. Pregled prijašnjih narudžbi
5. Rad van mreže

Na slici Sl. 3.1.1. nalazi se prikaz narudžbi sa aplikacije.

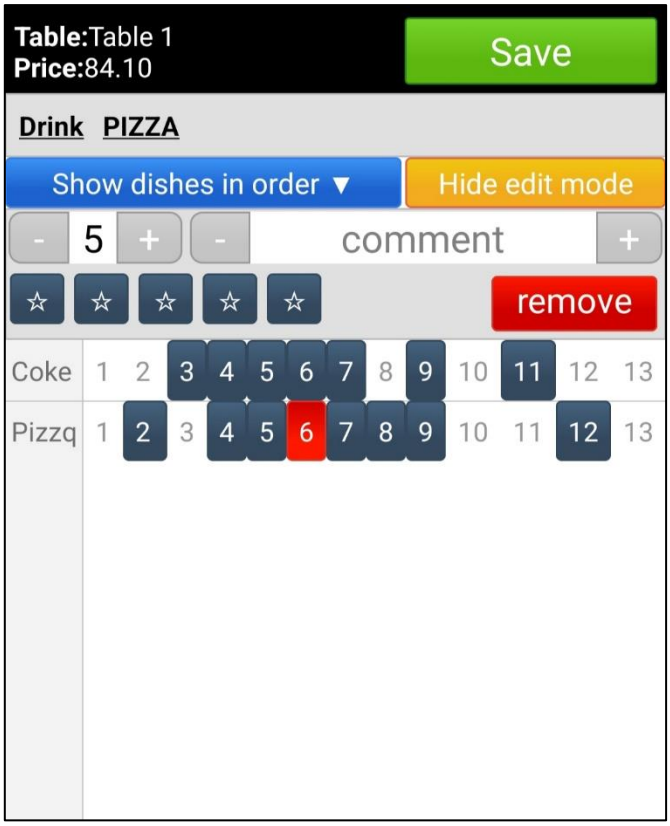
Sl. 2.1.1. *WaiterBolt aplikacija*

2.2. Waiter Fastoder

Aplikacija namijenjena spremanju narudžbi u svrhu unaprjeđenja posla. Unutar aplikacije moguće je kreirati zasebne prostorije te dodati broj stolova, moguće je dodati proizvode te ih spremiti u zasebne kategorije, kreirati novu narudžbu te ju prikazati. Korištenje same aplikacije vrlo je neintuitivno pošto se narudžbe mogu označiti brojevima od 1 do 13 no nije jasno koja je svrha toga. Osim toga korisničko sučelje može biti bolje realizirano. Glavne funkcionalnosti Waiter Fastoder aplikacije su:

1. Upravljanje narudžbama
2. Ispis računa
3. Jednostavna promjena menija

Na slici Sl. 3.2.1. prikazan je izgled Waiter Fastoder aplikacije.



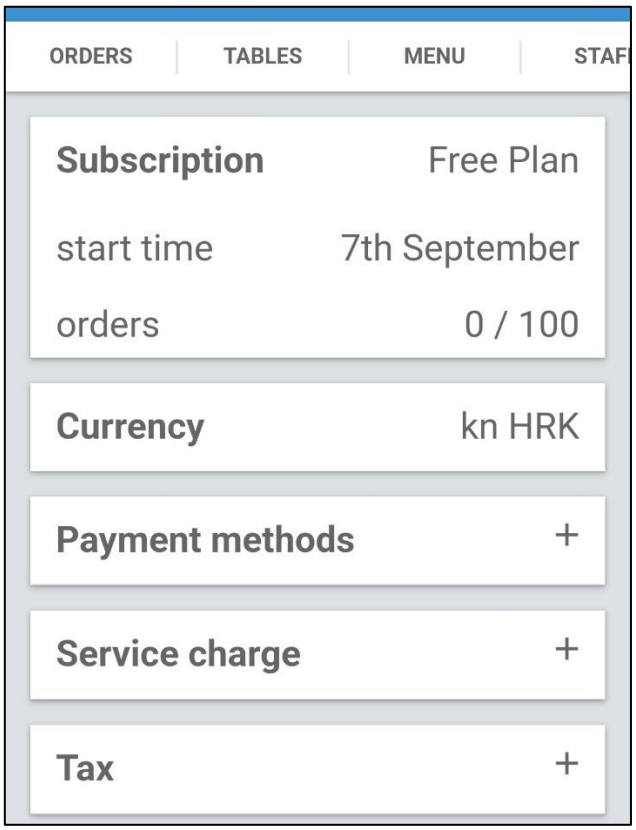
Sl. 2.2.1. *Waiter Fastoder aplikacija*

2.3. Waiterio

Waiterio je aplikacija namijenjena upravljanju restoranskih narudžbi. Njen kreator tvrdi da je aplikacija praktična i jednostavna za korištenje i da se preporuča svim uslužnim djelatnicima koji se ne žele zamarati papirom i olovkom. Nudi mogućnost kreiranja menija, pozivanja suradnika, primanja narudžbi, podjele računa. Funkcionalnosti Waiterio aplikacije su:

1. Upravljanje narudžbama
2. Brzo naručivanje
3. Kreiranje menija
4. Mogućnost plaćanja unutar aplikacije
5. Ispis narudžbe
6. Statistika za odabrana razdoblja

Aplikacija osobno ostavlja dojam nedovršenosti. Nudi mnogo funkcionalnosti no korisničko sučelje ne izgleda privlačno zbog količine teksta i brojnih izbornika. Izgled aplikacije Waiterio prikazan je na slici Sl. 3.3.1.



Sl. 2.3.1. *Waiterio aplikacija*

2.4. Pegasus waiter

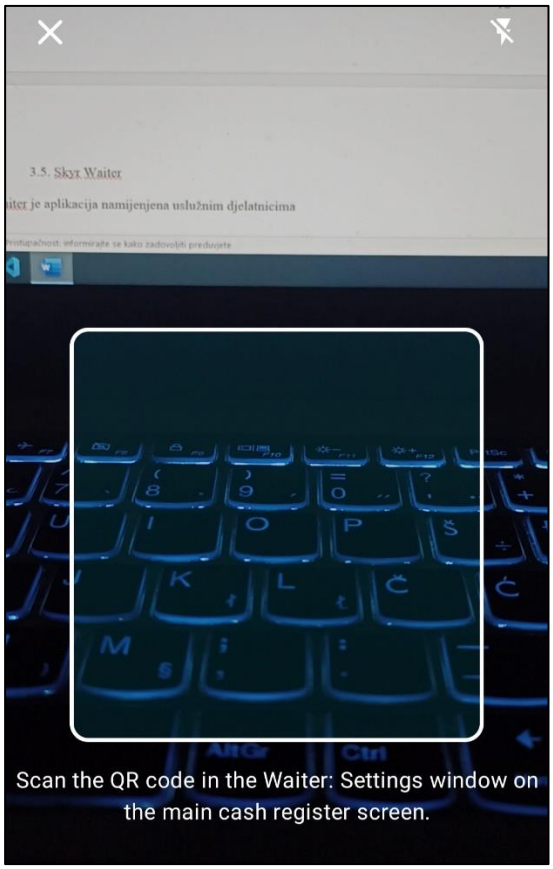
Pegaus waiter je aplikacija namijenjena upravljanju narudžbama u uslužnim objektima. Aplikacija nudi mogućnost odabira jezika na početnom zaslonu, no ta funkcionalnost zbog drukčijeg pisma koji nije latinica ponekad ne radi u skladu s očekivanjima. Nakon unosa imena djelatnika djelatniku se otvara zaslon sa kvadratima koji označavaju stolove. U njih je moguće unijeti narudžbu. Korisničko sučelje bolje je realizirano nego u aplikacijama pod 3.2. i 3.3., što je moguće vidjeti na slici Sl. 3.4.1.

Sl. 2.4.1. *Pegasus waiter aplikacija*

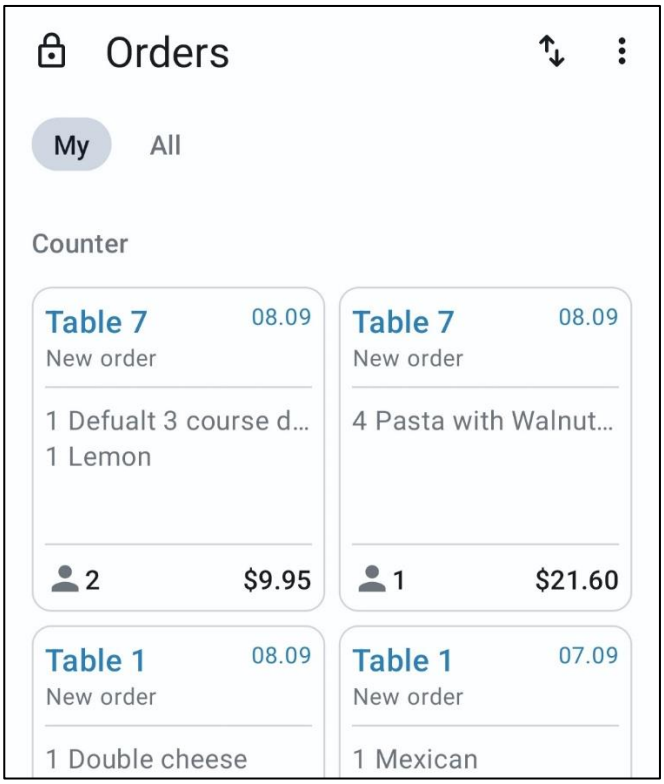
2.5. Syrve Waiter

Syrve Waiter je aplikacija namijenjena uslužnim djelatnicima. Aplikacija osim standardnih mogućnosti unosa i prikaza narudžbi nudi i mogućnosti skeniranja QR koda što je prikazano na slici Sl. 3.5.1. Aplikacija nudi i demo način rada zahvaljujući kojemu sam uspio vidjeti izgled

aplikacije. Aplikacija je super za korištenje, prikaz narudžbi je grafički i intuitivan. Generalno aplikacija ostavlja puno bolji dojam od ostalih.



Sl. 2.5.1. *QR čitač*



Sl. 2.5.2. *Syrve Waiter prikaz*

2.6. Generalna usporedba vlastite aplikacije sa navedenima

Od svih prethodno navedenih aplikacija, jedino aplikacija Syrve Waiter ima QR čitač kao i aplikacija koja se opisuje u ovom radu. Uloga QR čitača u takvim aplikacijama jest osigurati da samo osobe koje se fizički nalaze na lokaciji uslužnog objekta mogu napraviti narudžbu. Sve ostale aplikacije nemaju tu mogućnost, nego se narudžba pravi direktno u mobilnoj aplikaciji. Negdje nije dostupna ni registracija korisnika što dodatno olakšava mogućnost unosa lažnih podataka. Sučelja aplikacija nisu dorađena, nedostaje grafičkih elemenata koji bi ukrasili aplikacije. Jedino aplikacije navedene pod 3.1. i 3.5. zadovoljavaju potrebe korisnika. Niti jedna od navedenih aplikacije nema prikaz koliko je narudžba vremenski aktivna. Aplikacija napravljena za potrebe ovoga rada za svaku narudžbu prikazuje tajmer, a shodno sa otkucajem tajmera mijenja se i boja narudžbe od zelene do crvene. Još jedna od razlika između ove opisane u ovom radu i ostalih navedenih je što ostale nude funkcionalnosti samo za jedan uslužni objekt.

3. TEORIJSKA PODLOGA

Za potrebe lakšeg razumijevanja u ovom poglavlju bit će opisane korištene tehnologije. Krucijalni dio aplikacije jest programski paket Flutter, a korišteni su i Firebase baza podataka, Dart programski jezik, web tehnologije poput CSS-a (engl. *Cascading Style Sheets*) te JavaScript biblioteka React.

3.1. Flutter

Flutter je Googleov javno dostupan radni okvir čija je svrha razvoj aplikacija za više platformi pomoću istoga koda. Omogućen je razvoj za Android OS, iOS, macOS, Windows te web. Za izgradnju aplikacija koristi se Dart, programski jezik koji je također razvijen od strane Googlea, a nudi širok spektar widgeta i alata za izgradnju responzivnih korisničkih sučelja. Prvi put opisan je u 2015. godini, a pojavio se 2017. godine. Namjera je bila omogućiti prikazivanje (render) brzinom od 120 sličica u sekundi, a Flutter je iz godine u godinu napredovao. 2020. godine performanse na iOS uređajima poboljšale su se za prosječno pedeset posto. 2021. godina donijela je službenu podršku za web aplikacije pomoću Canvas renderera i widgeta koji su specifični za web. 2022. godine u verziji Flutter 3 proširio se ukupan broj podržanih platformi na 6, uključujući stabilnu podršku za Linux i macOS za Intelove i Appleove procesore. [1]

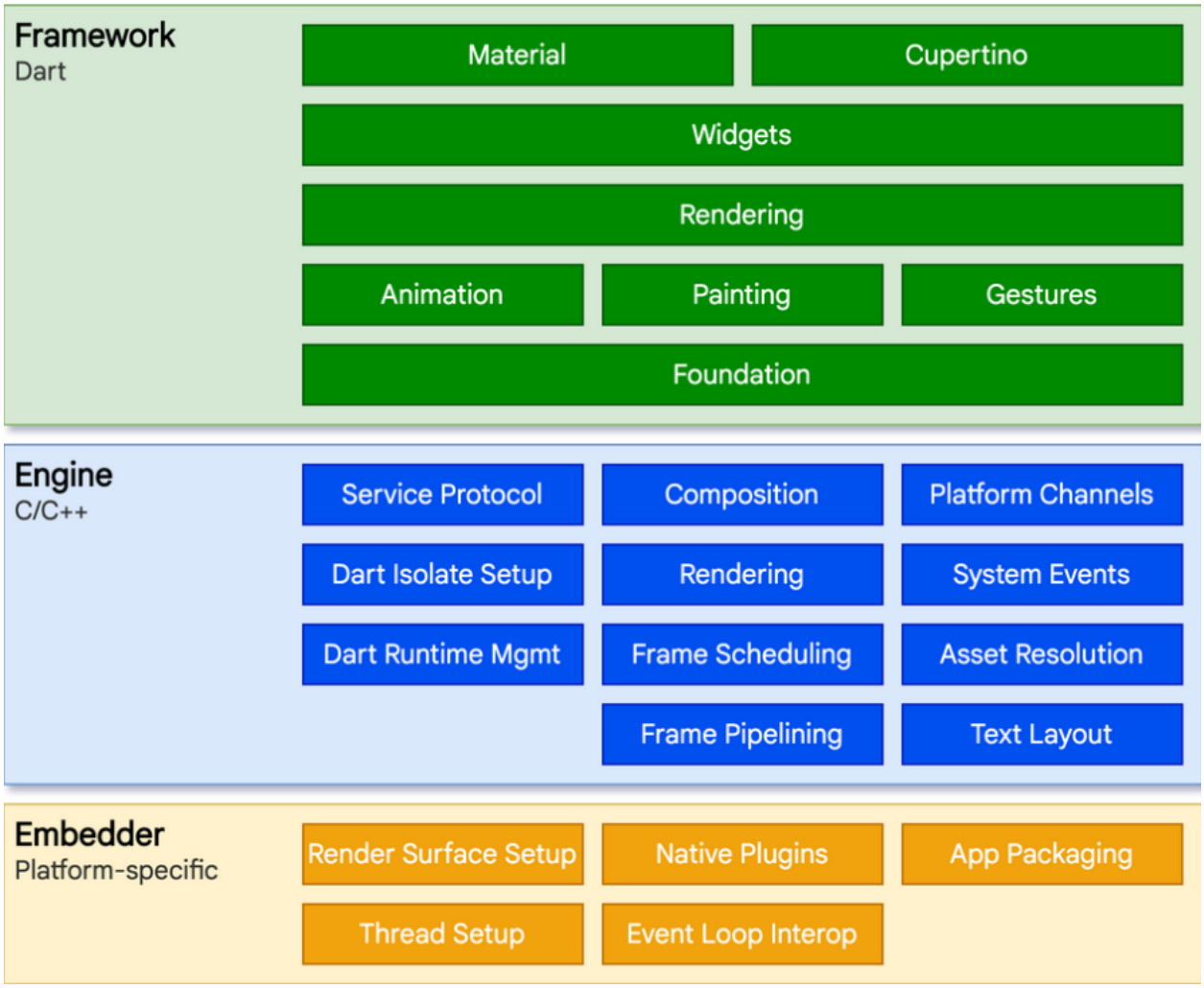
Velika prednost Fluttera jest brzo osvježavanje aplikacije. Ono omogućuje programerima trenutni prikaz promjena na pokrenutoj aplikaciji čim se napisani kod spremi. Standardni pristup u nativnom razvoju uključuje ponovnu kompilaciju koda i ponovno pokretanje aplikacije na mobilnom uređaju. Takav pristup značajno je sporiji i usporava razvoj. Osim brzog prikaza promjena, pozitivna strana je i očuvanje stanja u kojem se aplikacija nalazi. Dakle ukoliko je korisnik primjerice prijavom promijenio stanje aplikacije prilikom brzog osvježavanja mijenjaju se samo widgeti koji su promijenjeni, sva ostala stanja ostaju ista tako da će korisnik i dalje ostati prijavljen i izbjeći ponovno unošenje potrebnih podataka.

Da bi Flutter aplikacije imale performanse slične nativnima i glatke animacije koriste se specifični rendering enginei. Renderer je program koji prevodi UI kod u piksele koji se prikazuju na zaslonu mobilnog uređaja. Način na koji to radi je da svaki objekt sadrži instrukcije kako prikazati i postaviti widget. Te instrukcije daju se engineu i spremaju se u poredanu listu jednostavnih naredbi zvanih display list. Engine potom koristi render kako bi nacrtao display listu na mrežu piskela koji se prikazuju na zaslonu. [2] Do 2023. godine koristio se samo Skia renderer, a sada postoji i Impeller renderer koji se koristi samo za Android i iOS mobilne uređaje. Problem

kod korištenja Skia renderera je što nije dizajniran samo za Flutter. To je 2D grafička biblioteka koja omogućuje API-je koji se koriste na mnogo platformi pa tako služi kao grafički engine za Android, Flutter, Google Chrome, ChromeOS i druge proizvode [3]. Ima mnogo značajki napravljenih za širok spektar uređaja te zbog toga nije optimiziran za potrebe Fluttera.

3.2. Arhitektura Flutter radnog okvira

Cilj Fluttera je omogućiti programerima razvoj aplikacija visokih performansi koje se ponašaju prirodno na različitim platformama. Te aplikacije direktno se compileaju u strojni kod, ili u JavaScript u slučaju da se aplikacija pokreće na webu. Flutter je kreiran kao slojevit sustav sastavljen od niza biblioteka prikazanih na slici Sl. 2.2.1. [5]



Slika 3.2.1. *Strukturni slojevi [5]*

Flutter aplikacije pakiraju se na isti način kao i ostale nativne aplikacije prema temeljnom operacijskom sustavu. Embedder koji je specifičan za razne platforme omogućuje ulaznu točku odnosno koordinate za pristup uslugama poput renderanja površina, pristupnosti. Embedder je napisan u jeziku ovisno o platformi, što znači Java i C++ za Android uređaje, Objective-C/Objective C++ za macOS i iOS uređaje te C++ za Linux i Windows. Koristeći embedder, kod napisan u Flutteru može se integrirati kao modul u postojeću aplikaciju. [5]

Jezgra Fluttera je Flutter engine napisan uglavnom u C++ programskom jeziku i on pruža osnove podrške za sve Flutter aplikacije. Engine je odgovoran za rasteriziranje sastavljenih scena u trenutcima kada novi okvir treba biti prikazan. On omogućuje implementaciju Flutter APIa, uključujući grafiku, datoteke, mrežni ulaz/izlaz, Dart runetime i podršku pristupu. Flutter okviru engine je izložen pomoću dart:ui biblioteke koja je ključna biblioteka. Dart:ui izlaže najnižu razinu usluga koje Flutter okviri koriste za pokretanje aplikacija, poput klasa za upravljanje ulazom, grafikom, tekstom, rasporedom i podsustavima crtanja. [5]

3.3. Dart

Dart je programski jezik razvijen od strane Googlea kojem je prva verzija nastala 2013. godine. Koristi se za razvoj web i mobilnih aplikacija kao i server i desktop aplikacija [3]. Objektno je orijentiran, s jednostavnom sintaksom za visoku produktivnost, a dovoljno moćan za kreiranje širokog spektra aplikacija. Upravo zahvaljujući tome što je razvijen od strane Googlea omogućuje prilagodbe radnom okviru Flutter za izradu aplikacija. Dart koristi JIT (engl. *Just in time*) strategiju kompilacije. To znači da se kod dinamički prevodi prije pokretanja i da su promjene u kodu trenutno vidljive. JIT-om se uštedi na vremenu, posebno tijekom faze razvoja i debuggiranja. Ima veliku podršku od strane razvojnih alata (Android Studio, Visual Studio Code, IntelliJ Idea), automatsku nadopunu koda, bogatu standardnu biblioteku koja uključuje module za učestale zadatke. [3] Dart sadrži brojne biblioteke koje su napisane od strane Dart razvojnog tima, a još više biblioteka napisano je od strane motiviranih grupa i pojedinaca. Biblioteke omogućuju uštedu vremena korištenjem već gotovog programskog koda. Negativna strana je što previše uvezenih biblioteka može značajno usporiti rad aplikacije, a uz to rizik kod kreatora koji nisu članovi službenog tima može biti i puštanje malicioznog koda ili zapuštanja biblioteke s vremenom.

Flutter je zasigurno popularizirao Dart, no osim u Flutteru može se koristiti i u drugim primjenama:

- Izgradnja web aplikacija pomoću AngularDart okvira

- Izgradnja HTTP poslužitelja pomoću Aqueduct okvira
- Izrada IoT aplikacija

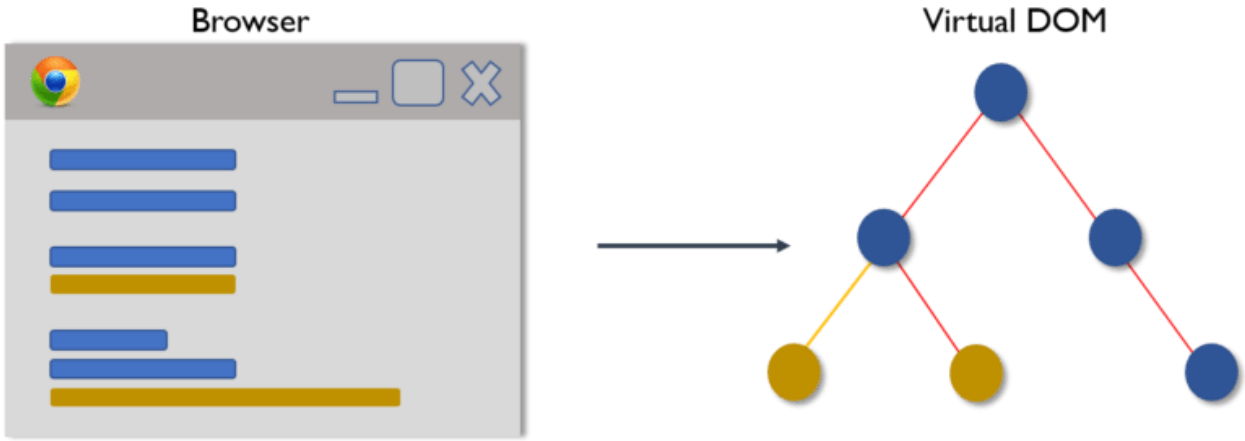
3.4. React

React je JavaScript biblioteka otvorenog koda čija se izgradnja korisničkih sučelja temelji na komponentama. React biblioteka razvijena je od strane Mete (nekadašnji Facebook), a prva upotreba bila je upravo na Facebookovoj naslovnoj stranici. Osim verzije za razvoj web stranica, 2015. godine u upotrebu je pušten React Native, inačica koja se uglavnom koristi za razvoj mobilnih (Android, iOS) nativnih aplikacija.[7]

Kostur svih React aplikacija su komponente. Komponente su manji dijelovi koda koji enkapsuliraju specifične funkcionalnosti i izgled, a zbog svoje ponovne upotrebljivosti koriste se pri izradi stranica koje imaju ponavljajuće elemente. Osim navedenog, komponente čine kod čitljivijim jer kompleksna sučelja rastavljaju na više manjih dijelova.

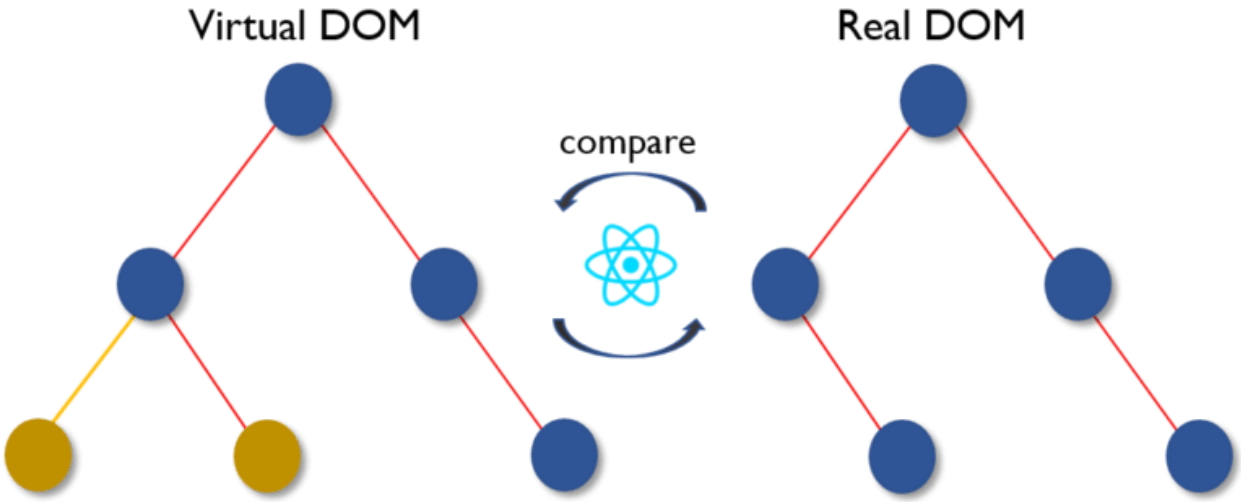
Brzina osvježavanja zaslona još je jedna od velikih prednosti korištenja Reacta. Ona se postiže virtualnim DOM-om (engl. *Document Object Model*). Virtualni DOM je memorijski prikaz stvarnog DOM-a. Prikaz korisničkog sučelja sprema se u memoriju te se sinkronizira sa stvarnim DOM-om, čime minimalnim nizom promjena dovodi do sinkronizacije zaslona. Način na koji virtualni DOM optimizira ažuriranje podijeljen je u tri jednostavna koraka: [8]

1. Kada god se podaci promijene, cijelo korisničko sučelje ponovno se stvara u virtualnom DOM-u, kako je prikazano na slici 2.4.1.



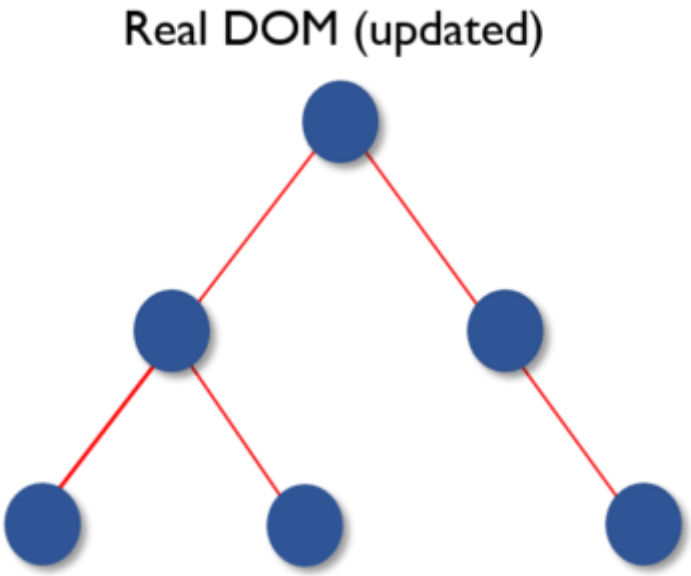
Sl. 3.4.1. *Stvaranje korisničkog sučelja u virtualnom DOM-u [8]*

2. Izračunava se razlika između prijašnjeg i novog DOM prikaza.



Sl. 3.4.2. *Izračun razlike između stvarnog i virtualnog DOM-a [8]*

3. Nakon odrađenih izračuna, stvarni DOM će se ažurirati samo s onim dijelovima koji su ažurirani.



Sl. 3.4.3. *Novi, ažurirani izgled DOM-a [8]*

3.5. Firebase

Za potrebe spremanja podataka korišten je Firebase baza podataka u stvarnom vremenu. Isto kao i Flutter, razvijen je od strane Googlea. Firebase pruža korisnicima brojne usluge koje bi inače morali sami implementirati. Među tim uslugama omogućena je autentifikacija, strojno učenje, mogućnost aktiviranja računa pomoću maila, izvješća o padu sustava. Što se tiče same Firebase i Flutter integracije, povezivanje je maksimalno olakšano. Prije svega potrebno je preuzeti Firebase CLI. Potom je potrebno kreirati novi Flutter projekt i potrebno je samo pomoću komandne linije dodati Firebase u projekt. Način na koji se to postiže je korištenjem Flutterfirea. Može se odabrati za koju vrstu aplikacije je potreban Firebase (iOS, web ili Android), aplikacija je registrirana i spremna je za korištenje. Primjer integracije prikazan je na slici 2.4.1.

```
Already logged in as goran.klement19@gmail.com

> dart pub global activate flutterfire_cli
Package flutterfire_cli is currently active at version 0.2.7.
The package flutterfire_cli is already activated at newest available version.
To recompile executables, first run `dart pub global deactivate flutterfire_cli`.
Installed executable flutterfire.
Activated flutterfire_cli 0.2.7.

> flutterfire configure
i Found 3 Firebase projects.
? Select a Firebase project to configure your Flutter application with · qrscannergklcl (qrscannergklcl)
? Which platforms should your configuration support (use arrow keys & space to select)? · android, ios, macos, web
i Firebase android app com.example.test_fb is not registered on Firebase project qrscannergklcl.
i Registered a new Firebase android app on Firebase project qrscannergklcl.
i Firebase ios app com.example.testFb is not registered on Firebase project qrscannergklcl.
i Registered a new Firebase ios app on Firebase project qrscannergklcl.
i Firebase macos app com.example.testFb registered.
i Firebase web app test_fb (web) registered.

Firebase configuration file lib\firebase_options.dart generated successfully with the following Firebase apps:
```

Sl. 3.5.1. *Integracija Firebasea u Flutter projekt*

3.6. Visual Studio Code

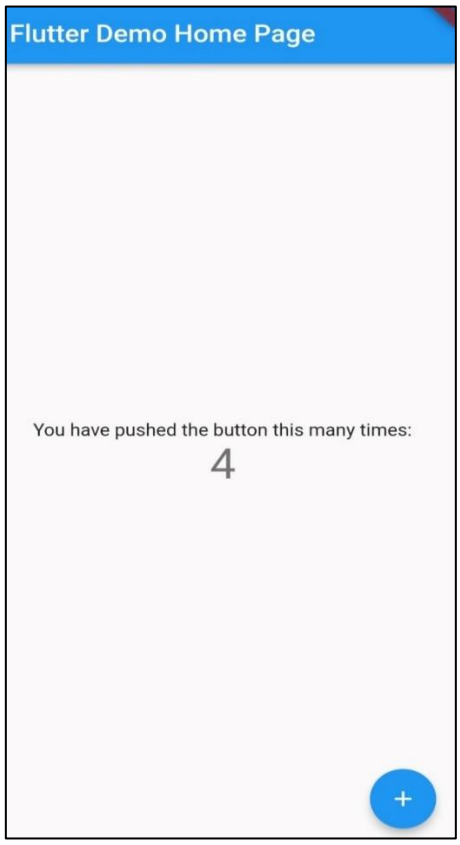
VS Code Microsofteov je uređivač koda. Prvi put najavljen je 2015. godine Može se koristiti na Windows, macOS i Linux platformama. Korisničko sučelje samog uređivača dobro je organizirano, može se personalizirati po potrebama, tako da je moguće mijenjati font, tamni ili svijetli način, prečace na tipkovnici, a ima i nekoliko tisuća ekstenzija koje omogućavaju ugodnije korištenje VS Codea u smislu testiranja, razvoja, otklanjanja pokreški. Podržava brojne jezike poput Jave, JavaScripta, C, C#, C++, Rust, Python, Julia, Fortran, Go. Podrška bi značila da naglašava dijelove sintakse, ima ugrađen IntelliSense za dovršavanje koda, naglašava poklapanje odgovarajućih zagrada, omogućava formatiranje koda. Po istraživanju Stack Overflowa iz 2022. godine, VS Code je najkorišteniji uređivač koda sa gotovo 75% aktivnih korisnika među programerskom populacijom. [6]

4. IZRADA MOBILNE APLIKACIJE ZA USLUŽNE OBJEKTE

Mobilna aplikacija za uslužne objekte mogla je biti izrađena u Java ili Kotlin programskom jeziku za Android sustave, u Swift programskom jeziku za iOS sustave no odlučeno je koristiti višeplatformni razvojni okvir Flutter i pripadni programski jezik Dart. Flutter aplikacije moguće je razvijati ili u Android Studio integriranom razvojnom okruženju, IntelliJu ili u VS Codeu koji je na kraju u uzet zbog jednostavnosti.

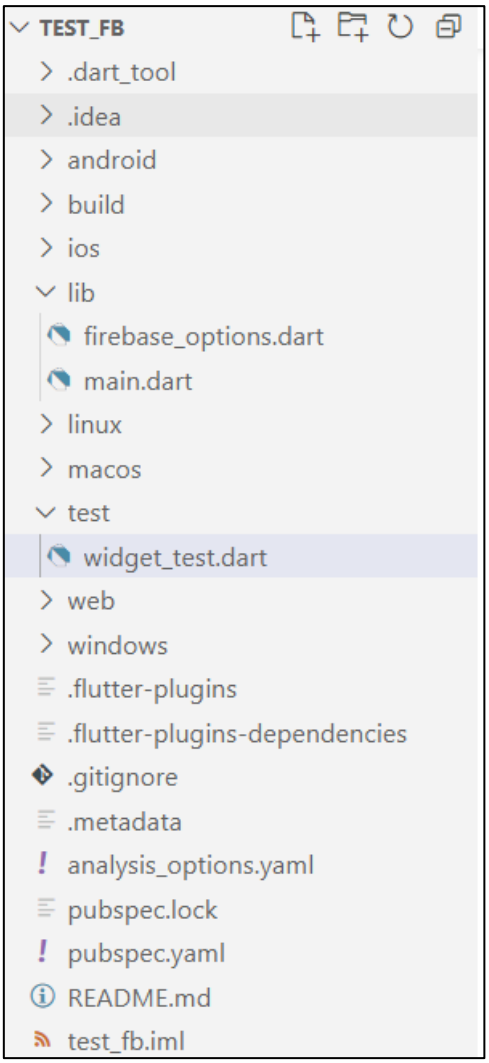
4.1. Uvod u aplikaciju

Pri kreiranju Flutter projekta generira se zadana osnovna aplikacija za brojanje pritisaka na gumb. Izgled te aplikacije prikazan je na slici 3.1.1.



Sl. 4.1.1. *Flutter demo aplikacija*

Takva aplikacija dolazi sa automatski generiranim kodom koji se može automatski pokrenuti na svim dostupnim platformama (Android, iOS, web, Windows, Linux, macOS). Ono što omogućava pokretanje su datoteke prikazane na slici Sl. 3.1.2.



Sl. 4.1.2. *Prikaz potrebnih datoteka*

Android, ios, web, macos i linux direktoriji omogućuju aplikaciji pokretanje na navedenim platformama. Sadrže specifične datoteke za konfiguraciju, poput recimo build.gradle i AndroidManifest.xml za Android, Podfile i Info.plist za iOS. Najvažniji direktorij jest lib. Ondje se pišu Dart datoteke u kojima se definira izgled zaslona i implementiraju njegove funkcionalnosti. Početna točka svake aplikacije je main.dart datoteka. Dart_tool direktorij sadrži datoteke koje koriste razni Dart alati, a sadrži i package_config.json datoteku koja specificira verziju i lokaciju paketa. Pubspec.yaml definira ovisnosti i ostale metapodatke o aplikaciji. Ovdje su definirane korištene slike, glazba i uvezeni paketi koji su korišteni prilikom razvoja. U test direktorij piše se kod kojim se želi provjeriti ispravan rad aplikacije. Navedeno se postiže pomoću test-caseova.

Flutter aplikacije sastoje se od velikog broja povezanih widgeta. Svaki widget ima ugrađena svojstva kojima se dodjeljuju vrijednosti, a promjenom tih vrijednosti utječe se na izgled i funkciju widgeta. Svojstva mogu biti obavezna i neobavezna, pa su tako na slici Sl. 3.1.3.

prikazana svojstva za ElevatedButton widget. Obavezna svojstva navedena su kao „required“, dakle ElevatedButton mora implementirati onPressed() funkciju i mora navesti svoje dijete widget koji označava tekst koji će biti napisan unutar gumba.

```
(new) ElevatedButton ElevatedButton({
  Key? key,
  required void Function()? onPressed,
  void Function()? onLongPress,
  void Function(bool)? onHover,
  void Function(bool)? onFocusChange,
  ButtonStyle? style,
  FocusNode? focusNode,
  bool autofocus = false,
  Clip clipBehavior = Clip.none,
  MaterialStatesController? statesController,
  required Widget? child,
})
```

Sl. 4.1.3. *ElevatedButton svojstva*

Nakon kratkog općenitog pojašnjenja Flutter aplikacija vrijeme je za opis same aplikacije za uslužne objekte. Ideja aplikacije bila je napraviti Flutter aplikaciju koja će olakšati narudžbe u uslužnim objektima i na taj način pomoći i klijentima i djelatnicima. Nakon učitavanja aplikacije, pojavljuje se mogućnost odabira prijave/registracije u sustav (za djelatnike) ili za otvaranje QR skenera za klijente. Klijenti pomoću skenera skeniraju QR kod koji ih dovodi do web stranice na kojoj su prikazani dostupni proizvodi. Klijenti odabiru željene proizvode i narudžba se sprema u bazu podataka. Sa druge strane, svaki djelatnik mora biti registriran kako ne bi došlo do problema u poslovanju. Nakon što se prijavi u sustav, djelatnik unosi ukupan broj stolova koji objekt posjeduje, a nakon toga bira brojeve stolova koje on poslužuje kako bi djelatnicima dolazile samo narudžbe za koje su oni odgovorni. Nakon odabira stolova djelatniku se kronološkim redom prikazuju narudžbe na čekanju. Prilikom dolaska nove narudžbe javlja se glasovni signal te se ona u stvarnom vremenu pojavljuje na dnu liste narudžbi. Djelatniku je omogućeno brisanje narudžbe nakon što klijenti budu posluženi.

4.2. Registracija i prijava djelatnika

Prilikom otvaranja aplikacije, korisniku se prikazuje učitavajući zaslon. Tijekom prikaza toga zaslona aplikacija obavlja nužne inicijalizacije poput povezivanja sa bazom podataka. Kada se otvori početni meni nudi se mogućnost registracije, prijave ili QR skeniranja. Prvo će biti

prikazane najvažnije funkcije koje omogućuju registraciju i prijavu djelatnika. Te funkcionalnosti nužne su za ovakav tip aplikacije zbog zaštite od neovlaštenog korištenja. Za te potrebe korištena je Firebase autentifikacija. Firebase nudi više mogućih opcija autentifikacije, od autentifikacije emailom i lozinkom (korištena u ovoj aplikaciji) do autentifikacije pomoću Google, Twitter, GitHub ili Facebook računa.

Prilikom registracije potrebno je unijeti email, lozinku, ponovljenu lozinku, inicijale (za označavanje prilikom uzimanja narudžbe) te je potrebno odabrati radno mjesto u padajućem izborniku kako bi narudžbe pristizale samo za taj uslužni objekt. Dio funkcije koji odrađuje registraciju prikazan je na slici Sl. 4.2.1.

```
Future register() async {
  Timer(const Duration(seconds: 10), () async => await checkEmailVerified());

  try {
    if (EmailValidator.validate(emailController.text) &&
        passwordController.text == repeatPwdController.text &&
        passwordController.text.length >= 6 &&
        initialsController.text != "" &&
        dropdownValue != "") {
      await FirebaseAuth.instance
          .createUserWithEmailAndPassword(
              email: emailController.text.trim(),
              password: passwordController.text.trim())
          .then((userCredential) async {
        User? user = userCredential.user;
        await user?.updateDisplayName(initialsController.text);
        await user?.updatePhotoURL(dropdownValue);
      });

      isVerified = FirebaseAuth.instance.currentUser!.emailVerified;
      if (!isVerified) {
        final user = FirebaseAuth.instance.currentUser;
        await user!.sendEmailVerification();
```

Sl. 4.2.1. *register() funkcija*

Jedna od posebnosti Dart programskoj jezika jest Future tip podatka. Da bi aplikacije bile responzivne i učinkovite, Google je uveo Future tip podatka za upravljanje asinkronim funkcijama. Taj tip podatka predstavlja potencijalnu vrijednost ili grešku u budućnosti bez blokiranja glavne niti čime se povećava responzivnost. U register() funkciji provjerava se jesu li potrebni unosi zadovoljili potrebne kriterije. Kriteriji koji trebaju biti ispunjeni su:

1. Unos u email upravljaču ne smije biti prazan
2. Lozinka mora sadržavati više od 6 znakova
3. Unos u upravljaču za inicijale ne smije biti prazan

4. Mora se odabrati vrijednost u padajućem izborniku
5. Lozinka i ponovljena lozinka moraju biti jednake

Ukoliko su uvjeti ispunjeni poziva se Firebaseova funkcija createUserWithEmailAndPassword() koja registrira novog korisnika u bazu podataka, a nakon toga korisniku još mijenjamo osobne podatke kako bi aplikacija očekivano radila. Ako korisnik nije verificiran, mora obaviti verifikaciju potvrđivanjem u mailu kojeg je naveo. Funkcija koja obavlja provjeru prikazana je na slici Sl. 3.2.2. Funkcija koristi Firebase za dohvat stanja korisnika, te mijenja stanje ovisno o rezultatu. Stanja su također jedna od posebnosti Dart programskog jezika, promjenom stanja dolazi do ponovnog ažuriranja zaslona.

```
Future<void> checkEmailVerified() async {
  await FirebaseAuth.instance.currentUser!.reload();
  setState(() {
    isVerified = FirebaseAuth.instance.currentUser!.emailVerified;
  });
}
```

Sl. 4.2.2. *checkEmailVerified() funkcija*

Nakon obavljene registracije, korisnici su u mogućnosti prijaviti se u aplikaciju. Prijava se također odvija pomoću Firebasea, točnije signInWithEmailAndPassword() funkcije.

4.3. Postavke djelatnika

Nakon uspješne prijave djelatnik postavlja uvjete koji se nalaze u uslužnom objektu. Ono što mora unijeti jest maksimalan broj stolova koje objekt ima (npr. ukoliko se u cijelom objektu nalaze 54 stola, unosi se brojka 54). Unos broja može se obaviti ili direktnim unosom ili klizačem. Dio koda Widgeta za direktan unos broja prikazan je na slici Sl. 4.3.1.

```
SizedBox(
  width: 90,
  child: TextField(
    controller: inputController,
    inputFormatters: [
      FilteringTextInputFormatter.allow(RegExp("[0-9]"))
    ],
    onChanged: (value) {
      double broj = double.parse(value);
      if (broj < 1 || broj > 100) {
        ScaffoldMessenger.of(context).showSnackBar(SnackBar(
            duration: const Duration(seconds: 3),
            backgroundColor: getColor(0xff18544b),
            content: const Text(
                "Number should be in range (1-100)"))); // Text
      } else {
        setState(() {
          _currentSliderValue = broj;
          addNumbers(broj.toStringAsFixed(0));
        });
      }
    },
```

Sl. 4.3.1. *TextField widget za unos broja*

SizedBox widget ovdje služi samo za dimenzioniranje TextFielda. TextField dopušta samo unos znamenki od 0-9, bez negativnih brojki. Ukoliko je unijeti broj van intervala od 1 do 100, djelatniku se ispisuje prikladna poruka koja ga navodi da broj mora biti unutar intervala. U slučaju da je uneseni broj prikladan, mijenja se stanje čijom promjenom dolazi do ponovnog osvježavanja zaslona i prikaza najnovijih informacija.

Drugi već spomenuti način unosa ukupnog broja stolova u objektu jest pomoću klizača. Njegov kod prikazan je na slici Sl. 4.3.2. Slider widget kao obavezna svojstva ima vrijednost te onChanged() funkciju. Od neobaveznih svojstava dodana je maksimalna vrijednost, broj podjela koji je postavljen na vrijednost „100“ kako bi se svaki broj mogao odabrati. Da je ovdje postavljeno da broj podjela bude jednak 50, bilo bi moguće odabrati svaki drugi broj (0,2,4,6,….,98,100). Posljednje neobavezno svojstvo je labela koja prikazuje odabranu vrijednost klizača.

```
Slider(
  value: _currentSliderValue,
  max: 100,
  divisions: 100,
  label: _currentSliderValue.round().toString(),
  onChanged: (double value) {
    setState(() {
      _currentSliderValue = value;
      inputController.value = TextEditingValue(
          text: _currentSliderValue.toStringAsFixed(0)); // TextEditingValue
      addNumbers(_currentSliderValue.toStringAsFixed(0));
    });
  },
), // Slider
```

Sl. 4.3.2. *Klizač za unos broja*

Na istom zaslonu potrebno je potom odabrati koje sve brojeve stolova djelatnik poslužuje, pri čemu je maksimalna vrijednost ona koju je odabrao u klizaču ili unio u polje.Za potrebe odabira brojeva korišten je MultiSelectDialogField widget. Taj widget kao obavezno svojstvo ima listu čiji elementi se mogu odabrati i onConfirm() funkciju koja u ovom slučaju odabrane brojeve sprema u stanje. U listu elemenata su pomoću for petlje dodani brojevi od 1 do odabranog broja na klizaču. Kod za MultiSelectDialogField widget prikazan je na slici Sl. 4.3.3.

```
MultiSelectDialogField(
  searchable: true,
  items: _items,
  selectedColor: Colors.blue,
  decoration: BoxDecoration(
    color: Colors.blue.withOpacity(0.1),
    borderRadius: const BorderRadius.all(Radius.circular(40)),
    border: Border.all(
      color: Colors.blue,
      width: 2,
    ), // Border.all
  ), // BoxDecoration
  buttonIcon: const Icon(
    Icons.checklist_rounded,
    color: Colors.blue,
    size: 40,
  ), // Icon
  buttonText: Text(
    "Pick table nubers You serve",
    style: TextStyle(
      color: Colors.blue[800],
      fontSize: 16,
    ), // TextStyle
  ), // Text
  onConfirm: (results) {
    _selectedNumbers = results;
  },
), // MultiSelectDialogField
```

Sl. 4.3.3. *MultiSelectDialogField widget*

4.4. Prikaz narudžbi

Nakon što djelatnik odabere potrebne postavke, otvara mu se zaslon za prikaz narudžbi. Prijelaz sa jednog zaslona na drugi događa se pomoću Navigator widgeta kao što je prikazano na slici Sl.4.4.1.

```
onPressed: () {
  Navigator.push(
    context,
    MaterialPageRoute(
        builder: (context) => OrderView(
            numbers: _selectedNumbers,
        )), // OrderView // MaterialPageRoute
  );
},
```

Sl. 4.4.1. *Princip rada Navigator widgeta*

Push() funkcija radi na način da na stog ruta dodaje novu rutu, a uloga MaterialPageRouta je da animira prijelaz sa jednog zaslona na drugi. Ukoliko se želi otvoriti prethodan zaslon, moguće je to učiniti pomoću pop() funkcije koja vraća rutu sa stoga i otvara ju.

Kao što je vidljivo sa slike 4.4.1. u novi zaslon predajemo i odabrane brojeve sa prethodnog zaslona. Da bi se narudžbe prikazale potrebno ih je prvo dohvatiti iz baze podataka. Pomoću funkcije FirebaseDatabase.instance.ref() kreiramo referencu na željenu bazu podataka. Pošto je Firebase Googleov servis, a Flutter je također napravljen od strane Googlea, rad sa Firebaseom znatno je olakšan. Primjer toga je i FirebaseAnimatedList widget koji kreira spremnik za listu. Ta lista ima animacije pri dodavanju ili uklanjanju elemenata iz liste. Upravo se FirebaseAnimatedList koristi za prikaz podataka iz Firebase baze podataka, što je prikazano na slici Sl. 4.4.2.

```
body: Column(
  children: [
    Expanded(
      child: FirebaseAnimatedList(
        query: _ordersRef.orderByChild("time"),
        itemBuilder: (BuildContext context, DataSnapshot snapshot,
            Animation<double> animation, int index) {
          if (widget.numbers.contains(
              int.parse(snapshot.child("number").value.toString()))) {
            return OrderModel(
              tableNumber: snapshot.child("number").value.toString(),
              order: snapshot.child("order").value.toString(),
              time: snapshot.child("time").value.toString(),
              isTaken: snapshot.child("isTaken").value.toString(),
              ikona: IconButton(
                color: Colors.white,
                iconSize: 40,
                icon: const Icon(Icons.delete),
                onPressed: () => _ordersRef.child(snapshot.key!).remove(),
              ), // IconButton
            ); // OrderModel
          } else {
            return const SizedBox(
              width: 1,
            ); // SizedBox
          }
        },
      ), // FirebaseAnimatedList
    ), // Expanded
  ],
), // Column
```

Sl. 4.4.2. *FirebaseAnimatedList kod*

Obavezna svojstva tog widgeta su query te itemBuilder. Query je Firebase upit kojime se popunjava lista, u ovom slučaju taj upit je dodatno sortiran pomoću orderByChild(„time“) metode. Svaka narudžba ima time ključ, a vrijednost koja se upisuje jesu milisekunde protekle od 1.siječnja 1970. ItemBuilder je zadužen za izgradnju liste widgeta. Ima DataSnapshot parametar koji indicira na snapshot koji bi se trebao koristiti pri izgradnji. Pri svakoj izmjeni podataka izu baze podataka lista elemenata se osvježava tako da je prikaz stalno sinkroniziran s vrijednostima iz baze.

Izgradnja liste radi na način da se provjerava sadrži li lista s brojevima stolova koje je djelatnik odabrao broj stola narudžbe. Ukoliko sadrži, kreira se element liste OrderModel kojemu

predajemo podatke iz baze podataka vezane za tu narudžbu (je li preuzeta, broj stola, naručeni proizvodi i vrijeme u milisekundama). Te provjere rade se za svaki element iz upita (engl. *query*). OrderModel klasa zadužena je za dizajn narudžbe i funkcionalnosti kao što je pretvaranje vremena u hh:mm:ss oblik, promjena broje od zelene do crvene kako vrijeme protiče, preuzimanje narudžbe. Funkcija za izračun proteklog vremena prikazana je na slici Sl. 4.4.3.

```
String calculateElapsedTime(int time) {
  int current = DateTime.now().millisecondsSinceEpoch;
  int seconds = ((current - time) ~/ 1000);
  int minutes = seconds ~/ 60;
  int remainingSeconds = seconds % 60;
  return '$minutes:${remainingSeconds.toString().padLeft(2, '0')}';
}
```

Sl. 4.4.3. *Kod funkcije za izračun proteklog vremena*

Funkcija kao parametar prima milisekunde, a vraća String u obliku hh:mm:ss. Da bi se vrijeme osvježavalo svake sekunde, kreiran je tajmer koji svake sekunde poziva setState() funkciju koja ponovno osvježava zaslon. Ta funkcija prikazana je na slici Sl. 4.4.4.

```
_timer = Timer.periodic(const Duration(seconds: 1), (timer) {
  setState(() {});
}); // Timer.periodic
```

Sl. 4.4.4. *Kod funkcije za periodično osvježavanje zaslona*

Kako vrijeme narudžbe protiče, bitno je naznačiti da klijent već dugo čeka. Stoga svaka narudžba mijenja nijansu boje svake sekunde. Kada se tek pojavi boja je svijetlo zelena, vremenom prelazi u smeđu pa nakon 10 minuta čekanja u tamno crvenu. Time se želi upozoriti djelatnika da ukoliko je moguće preuzme narudžbu. Funkcija koja mijenja boju u zavisnosti o vremenu prikazana je na slici Sl. 4.4.5.

```
getColorFromTime(String time) {
  List<String> timeParts = time.split(':');
  int minutes = int.parse(timeParts[0]);
  int seconds = int.parse(timeParts[1]);

  Duration elapsedTime = Duration(minutes: minutes, seconds: seconds);

  Duration remainingTime = const Duration(minutes: 10) - elapsedTime;
  if (remainingTime.isNegative) {
    remainingTime = const Duration();
  }
  double remainingPercentage =
      remainingTime.inSeconds / const Duration(minutes: 10).inSeconds;

  int red = (200 * (1 - remainingPercentage)).toInt();
  int green = (200 * remainingPercentage).toInt();

  if (widget.isTaken != "0" && user?.displayName != widget.isTaken) {
    return Color.fromRGBO(red, green, 0, 0.2);
  }

  return Color.fromRGBO(red, green, 0, 1);
}
```

Sl. 4.4.5. *Kod funkcije za promjenu boje*

Funkcija radi na način da razdvoji dobiveni string na dva dijela, dio prije dvotočke pridaje minutama, dio iza dvotočke pridaje sekundama. U Flutteru postoji Duration widget koji izračunava vremensku razliku, tako da ovdje izračunava koje vrijeme je preostalo do 10 minuta, a nakon toga i postotak koliko još ima do 10 minuta. Taj postotak potom se koristi za izračun boje, pri čemu su krajnje boje crvena i zelena. Funkcija vraća Color widget koji se potom pridaje pozadinskoj boji narudžbe.

Da bi djelatnik preuzeo narudžbu, potrebno je napraviti dvostruki dodir na narudžbu. U tom trenutku na narudžbi se pojavljuju njegovi inicijali. Inicijale djelatnik unosi prilikom registracije. Ukoliko više djelatnika radi za istim stolom, u trenutku kada netko preuzme narudžbu ostalim djelatnicima boja narudžbe problijedi (poveća joj se prozirnost). Svaku narudžbu moguće je i obrisati klikom na  ikonu. Ukoliko je jedan od djelatnika preuzeo narudžbu, ostali djelatnici za tim stolom ne mogu obrisati tu narudžbu. Funkcija za preuzimanje narudžbe prikazana je na slici Sl. 4.4.5. Prvo se dohvati trenutni korisnik prijavljen u sustav. Ukoliko narudžba nije preuzeta

u bazi podataka mijenja se vrijednost „isTaken“ u inicijale. Ukoliko djelatnik zbog raznih okolnosti ipak ne može preuzeti narudžbu, ponovni dvostruki dodir vratit će narudžbu na početno stanje.

```
onDoubleTap: () async {
  User? currentUser = FirebaseAuth.instance.currentUser;
  if (widget.isTaken == "0") {
    _ordersRef.child(widget.tableNumber).update({
      "isTaken": currentUser?.displayName,
    });
  } else {
    _ordersRef.child(widget.tableNumber).update({
      "isTaken": "0",
    });
  }
},
```

Sl. 4.4.5. *Kod funkcije za preuzimanje narudžbe*

4.5. QR čitač

Nakon što je opisan dio vezan za djelatnike, vrijeme je za opis dijela pomoću kojega klijenti kreiraju svoje narudžbe. Klijenti na početnom zaslonu ne idu na registraciju ili prijavu nego odabiru opciju za čitanje QR koda. Svaki uslužni objekt na svakom bi stolu trebao imati isprintani QR kod koji vodi do web stranice za naručivanje i broj stola kojeg je potrebno unijeti pri narudžbi. Klijentu se otvara prikaz kamere, pri čemu se kamera može promijeniti u prednju ili stražnju. Moguće je upaliti i bljeskalicu. Kada klijent skenira QR kod, ukoliko je QR valjan na mobilnom se uređaju otvara web stranica. Ne postoji službeni widget od strane Flutterovog razvojnog tima, tako da je u ovom slučaju korišten neslužbeni widget, što može imati svoje mane jer Flutterov razvojni tim konstantno ažurira svoje pakete. Neslužbeni paketi koje ne održava službeni tim nego inspirirani individualci ili timovi mogu s vremenom postati zastarjeli.

Paket za QR čitač potrebno je prvo uvesti u aplikaciju da bi bilo moguće koristiti MobileScanner widget. Njegov kod prikazan je na slici Sl. 4.5.1.

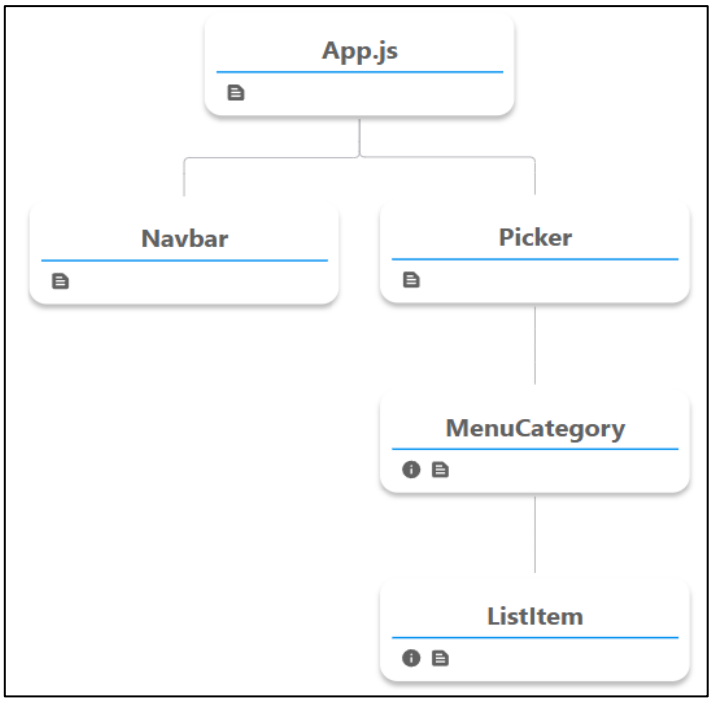
```
child: MobileScanner(
    allowDuplicates: false,
    controller: cameraController,
    onDetect: (barcode, args) {
      if (barcode.rawValue == null) {
        debugPrint('Failed to scan Barcode');
      } else {
        final String code = barcode.rawValue!;
        debugPrint('Barcode found! $code');
        launchUrlString(code);
      }
    }), // MobileScanner
```

Sl. 4.5.1. *Kod za čitač QR kodova*

Obavezno mora implementirati onDetect() funkciju koja se poziva kada se QR obradi i dekodira. U QR moguće je spremiti bilo što, primjerice URL, tekst, datum, čak i geografsku lokaciju.

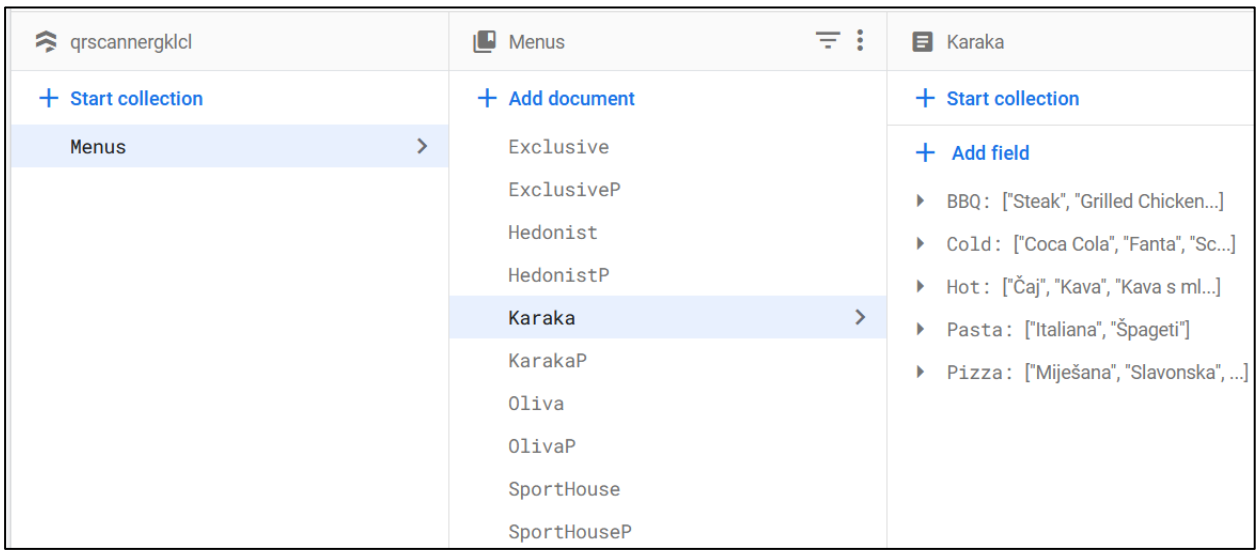
4.6. Web stranica za naručivanje

Web stranica napravljena je koristeći React biblioteku. Stablo komponenti web stranice prikazano je na slici Sl. 4.6.1. Navbar komponenta konstantno je na zaslonu, a Picker (za odabir restorana), MenuCategory(za prikaz kategorije proizvoda) i ListItem(za prikaz pojedinog proizvoda) izmjenjuju se ovisno o uvjetima. Picker komponenta sadrži stanje o svim MenuCategory komponentama koje kreira, a MenuCategory komponenta sadrži stanje o svakoj ListItem komponenti koju kreira, dakle svi odabrani proizvodi iz svih ListItem komponenti šalju se nazad do Picker komponente. Razlog tome je da bi se na jednom mjestu moglo imati sve odabrane proizvode.



Sl. 4.6.1. *Stablo komponenti*

Nakon otvaranja stranice korisnik odabire uslužni objekt u kojemu se nalazi pomoću padajućeg izbornika te mu se otvaraju kategorije proizvoda koje taj objekt nudi (topli napitci, hladna pića, pizze, roštilj, tjestenina). Ponuđeni proizvodi učitavaju se iz Firebase Firestore baze podataka, koja je prikazana na slici Sl. 4.6.2.



Sl. 4.6.2. *Prikaz Firebase Firestore za proizvode*

Firestore funkcionira na način da se kreiraju kolekcije, od kojih svaka može imati više dokumenata koji imaju više polja. U dokumente koji kao ime imaju naziv restorana spremljena su jela, a ukoliko je u naziv dodano slovo „P“ onda se u tom dokumentu spremaju cijene.

Funkcija koja dohvaća ponuđene proizvode iz Firebase Firestorea prikazana je na slici Sl. 4.6.3.

```
const getMenu = async () => {
  const docRef = doc(db, "Menus", props.name);
  const docSnap = await getDoc(docRef);

  if (docSnap.exists()) {
    setMenuData(docSnap.data());
  } else {
    console.log("No such document!");
  }
};
```

Sl. 4.6.3. *Funkcija za dohvaćanje menija*

Funkcija dohvaća vrijednosti iz „Menus“ kolekcije, iz određenog uslužnog objekta koji je predan kao prop dječjoj komponenti. Dakle u menuData se spremaju kategorije koje svaki objekt ima (hladna pića, topli napitci, pizze…). Ukoliko nijedna kategorija nije dostupna korisniku će se prikazati poruka, a ukoliko ima kategorija, za svaku od njih kreira se nova komponenta. Navedeno se postiže map() funkcijom koja iz listu elemenata uzima jedan po jedan element te koristeći njega komponenti koja će se kreirati predaje potrebne podatke. Bitno je naglasiti da originalna lista ostaje nepromijenjena. Navedeno je prikazano na slici Sl. 4.6.4.

```
const renderCategories = () => {
  const categories = menuData[selectedRestaurant?.name] || [];

  if (categories.length === 0) {
    return <p>No menu categories available.</p>;
  } else {
    return categories.map((category, index) => (
      <MenuCategory
        key={index}
        index={index}
        category={category}
        name={selectedRestaurant?.name}
        onUpdateState={updateChildState}
        onUpdatePrice={updateChildPrice}
      />
    ));
  }
};
```

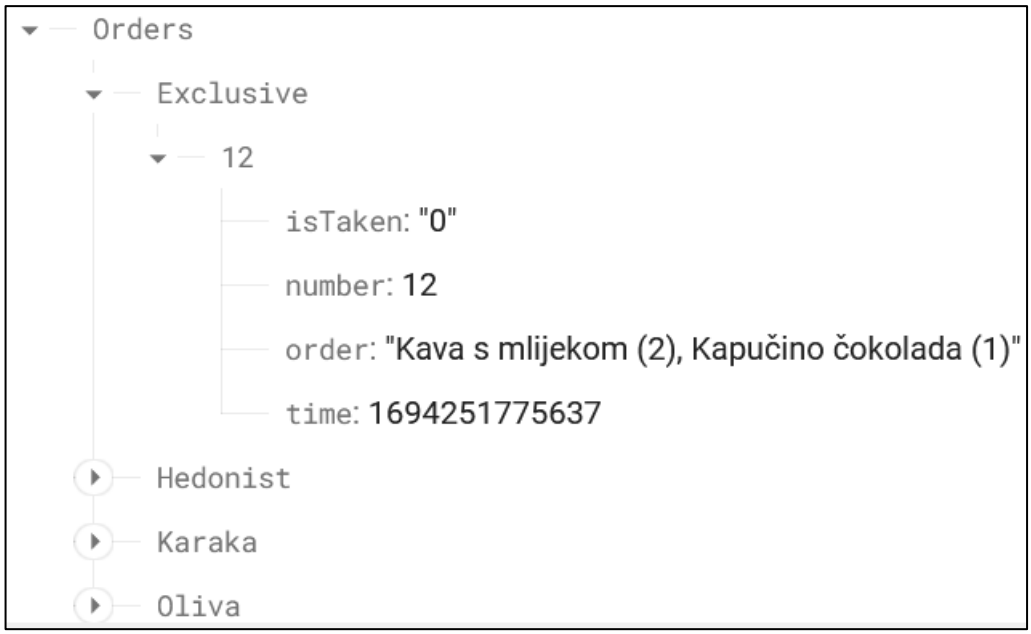
Sl. 4.6.4. *Funkcija za dohvaćanje menija*

Klikom na kategoriju prikazuju se svi proizvodi iz te kategorije i njihova cijena. Korisnik pomoću forme za unos broja klikom na „+“ ili „-„ znak može odabrati željenu količinu. U trenutku kada odabere neki od proizvoda, izbacuje se Toast poruka u kojoj piše što je korisnik naručio, u kojoj količini i kolika je ukupna cijena odabranih proizvoda. Kada se željeni proizvodi iz menija odaberu, unosi se broj stola i potvrđuje narudžba koja se šalje u bazu podataka. Funkcija je prikazana na slici Sl. 4.6.5.

```
const handleSaveOrder = async () => {
  const filteredChildStates = childStates.filter((state) => state !== "");
  if (selectedItems != {}) {
    set(
      ref(database, "Orders/" + selectedRestaurant.name + "/" + tableNumber),
      {
        number: tableNumber,
        order: filteredChildStates.toString(),
        time: Date.now(),
        isTaken: "0",
      }
    );
    showSuccess("Your order is sent!");
  } else {
    showWarn("Your order is empty!");
  }
};
```

Sl. 4.6.5. *Funkcija za spremanje narudžbe*

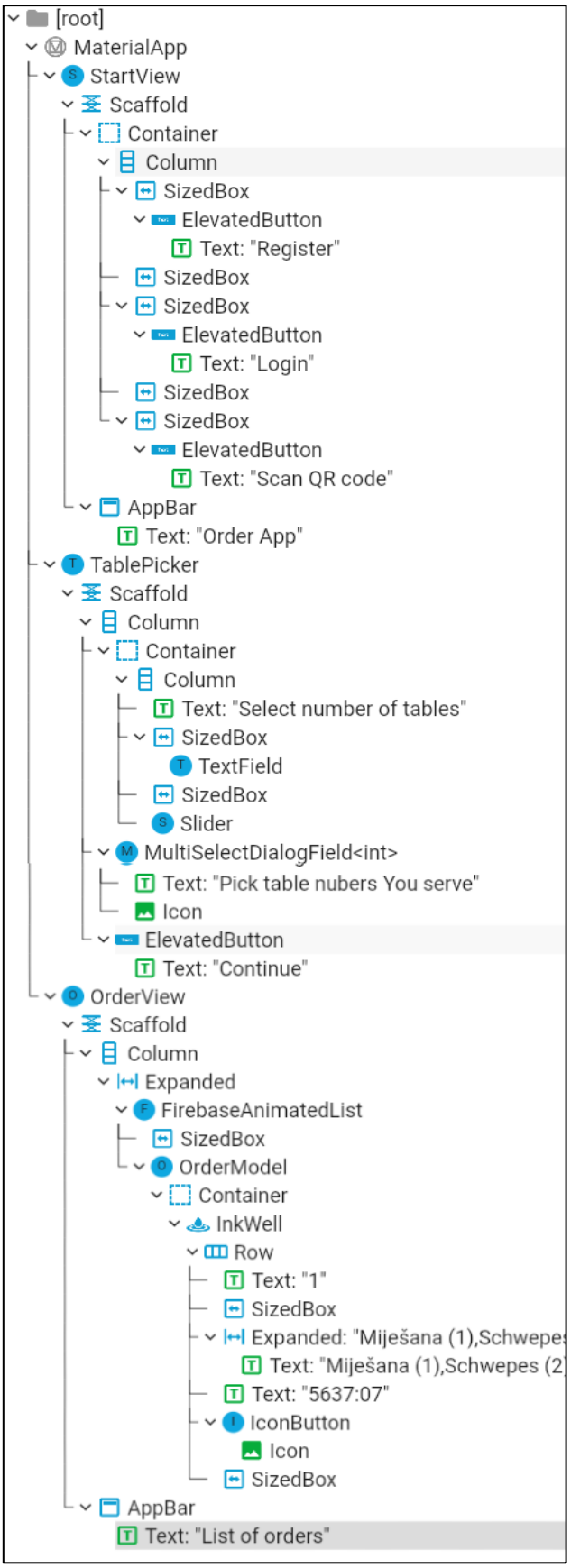
Funkcija se okida na klik gumba. ChildStates stanje nosi podatke o svim odabranim proizvodima. Pošto ne moraju iz svih kategorija proizvodi biti odabrani, childStates za neke komponente bude prazan tako da ga je potrebno filtrirati. Nakon filtriranja u bazu podataka se za odabrani restoran i uneseni broj stola unosi nova vrijednost. Mobilna aplikacija osluškuje istu tu bazu podataka te dodaje novu narudžbu koju djelatnici vide. Prikaz baze podataka nalazi se na slici Sl. 4.6.6.



Sl. 4.6.6. *Prikaz aktivnih narudžbi*

4.7. Stablo Widgeta

Svaka Flutter aplikacija sastoji se od widgeta. U Flutteru se praktički sve smatra widgetom, njime se omogućuje izgled, funkcionalnost. Koristi se stablo widgeta u kojemu svaki widget može imati svoje widgete djecu, potom ti widgeti mogu imati svoju djecu te takva hijerarhijska struktura definira čitavo korisničko sučelje. U Flutteru stablo widgeta možemo pronaći pomoću VS Code programa. Aplikacija se pokrene u načinu za otklanjanje pogrešaka na uređaju kojeg sami odaberemo. Nakon pokretanja aplikacije pojavi se opcija za otvaranje DevTools Widget inspektora. Potom na uređaju dolazimo do zaslona čije nas stablo widgeta zanima te se pritisne refresh opcija kako bi se stablo generiralo do trenutno odabranog zaslona. Izgled stabla za zaslon u kojemu se prikazuju narudžbe prikazan je na slici Sl. 4.3.1. Kao što se može vidjeti na slici, stablo je jako razgranato upravo zbog toga što je sve u Flutteru widget, čak i običan tekst koji se prikazuje unutar gumbova ili u gornjoj traci za prikaz naslova. Razlog tome je konzistencija, jer svi koriste iste widgete kako bi programerima bilo lakše učiti i kreirati aplikacije. Još jedan od razloga jest potreba za brzim osvježavanjem prilikom promjena u korisničkom sučelju.



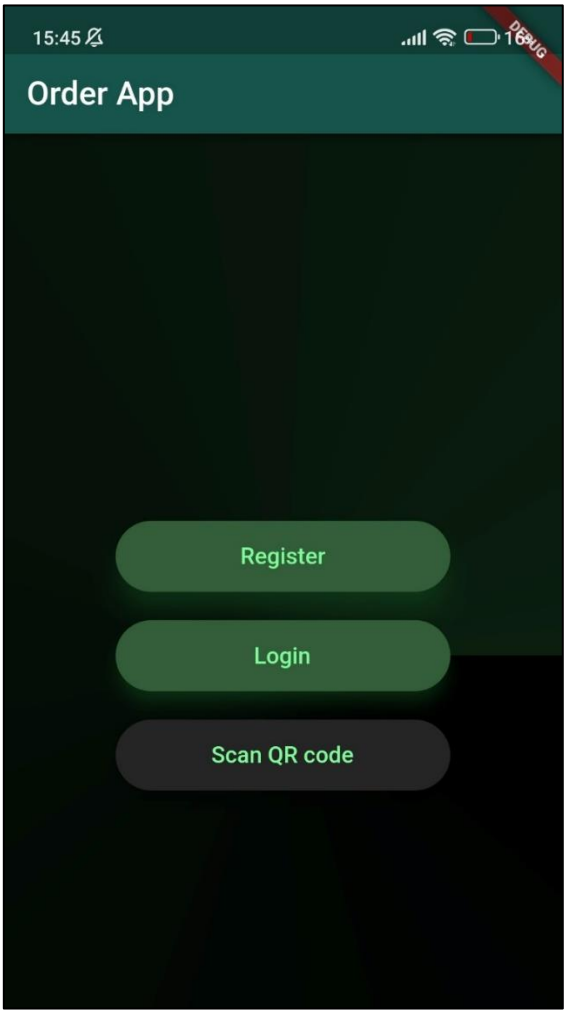
Sl. 4.7.1. *Stablo widgeta kod prikaza narudžbi*

5. IZGLED I KORIŠTENJE APLIKACIJE

U ovom dijelu rada prikazan je izgled pojedinih zaslona sa mobilnog uređaja.

5.1. Početni zaslon

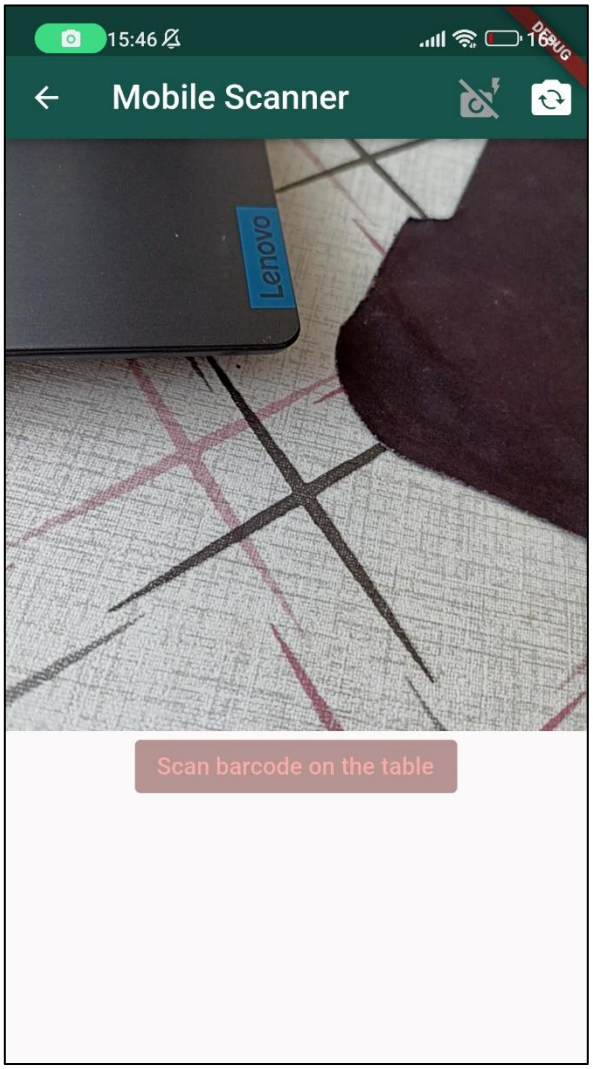
Uloga početnog zaslona jest odabiranje registracije ili prijave u slučaju da se radi o djelatniku ili skeniranje QR koda u slučaju da se radi o klijentu. Prije nego se učita ovaj zaslon, obavljaju se važne inicijalizacije kako bi sustav ispravno radio, poput spajanja na bazu podataka.



Sl. 5.1.1. *Početni zaslon aplikacije*

5.2. QR čitač

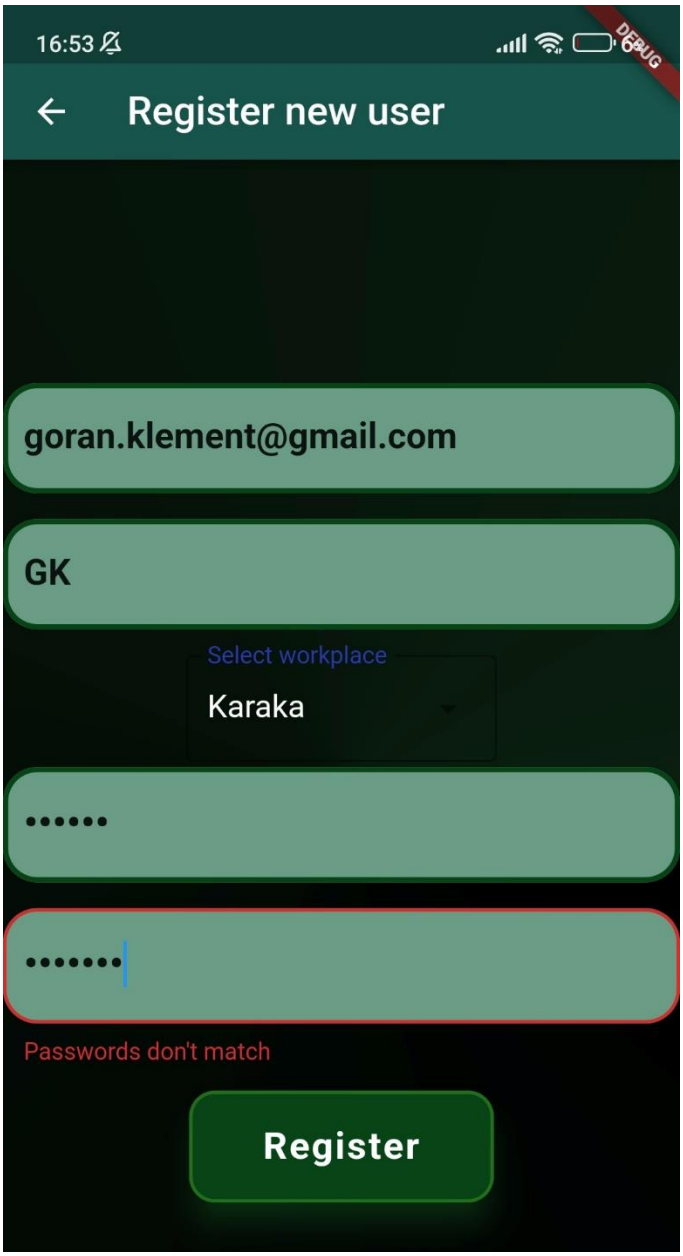
Kada klijent odabere gumb za QR skeniranje, otvara mu se prikaz kao na slici SL. 5.2.1. Obrada QR koda iznimno je brza, traje manje od pola sekunde. Korisnik može birati želi li koristiti prednju ili stražnju kameru te u slučaju noćnih uvjeta može upaliti bljeskalicu klikom na ikonu.



Sl. 5.2.1. *Prikaz QR čitača*

5.3. Registracija djelatnika

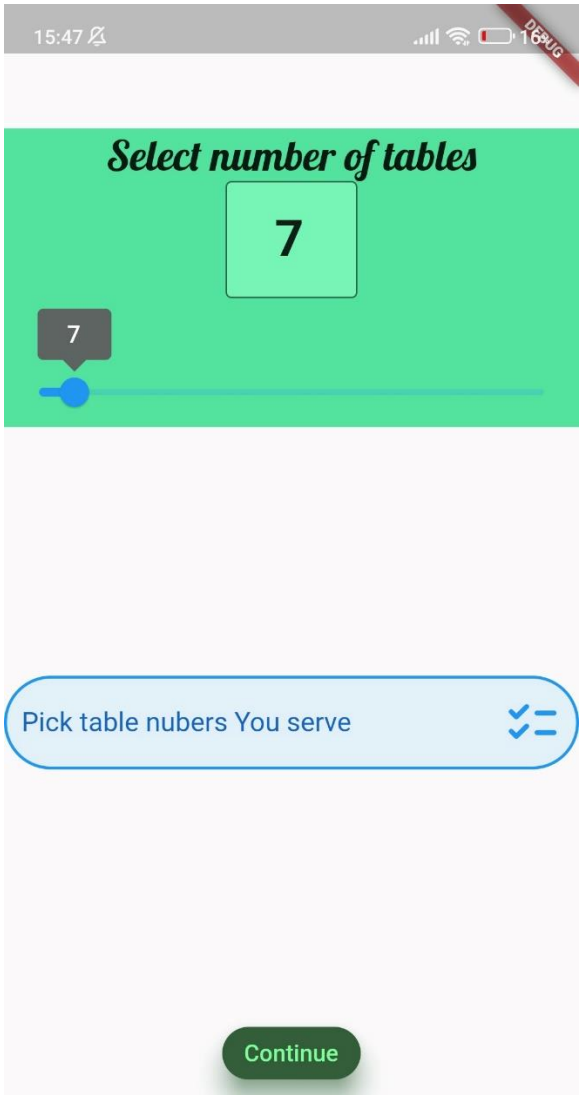
Registracija djelatnika prikazana je na slici Sl. 5.3.1. Potrebno je unijeti email, inicijale kako bi se mogle preuzete narudžbe označiti, radno mjesto djelatnika, i lozinku koja mora biti duža od šest znakova. Lozinka se mora ponoviti da bi se spriječile slučajne greške. Djelatniku u slučaju ispravnog unosa na mail stiže poruka za verifikaciju.



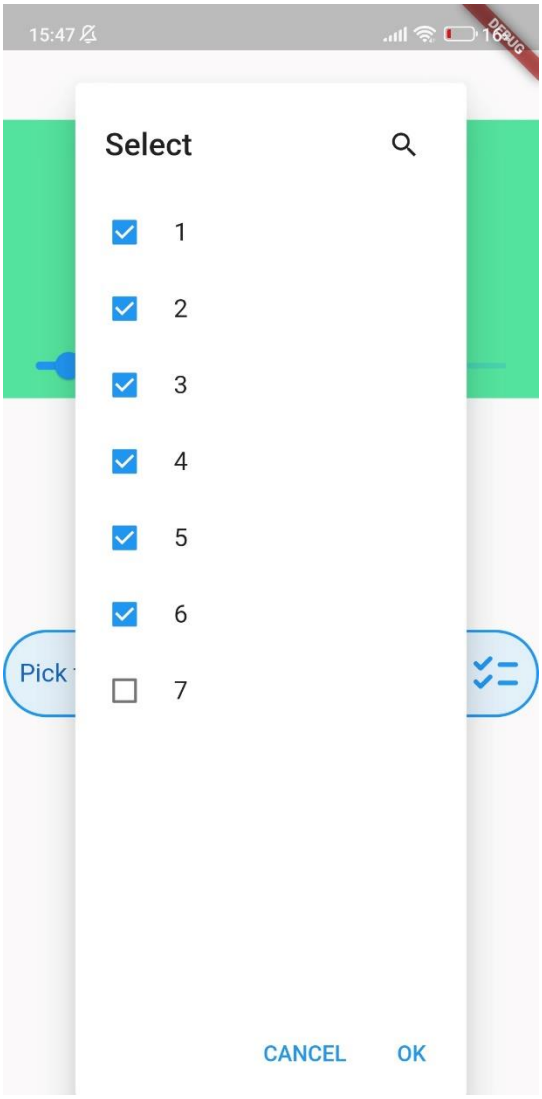
Sl. 5.3.1. *Registracijska forma za djelatnike*

5.4. Postavke za stolove

Da bi aplikacija radila očekivano potrebno je unijeti maksimalan broj stolova koji se nalazi u objektu. Navedeno se može učiniti pomoću klizača ili ručnim unosom brojke. Brojke su sinkronizirane tako da se nakon pomaka klizača brojka u unosu broja također promijeni. Nakon toga potrebno je odabrati koje sve stolove djelatnik poslužuje, od broja 1 do broja koji je odabran u prethodnom koraku. Navedeno je prikazano na slikama Sl. 5.4.1. i Sl. 5.4.2.



Sl. 5.4.1. *Widgeti za odabir broja stolova*



Sl. 5.4.2. Odabir stolova djelatnika

5.5. Izgled narudžbe

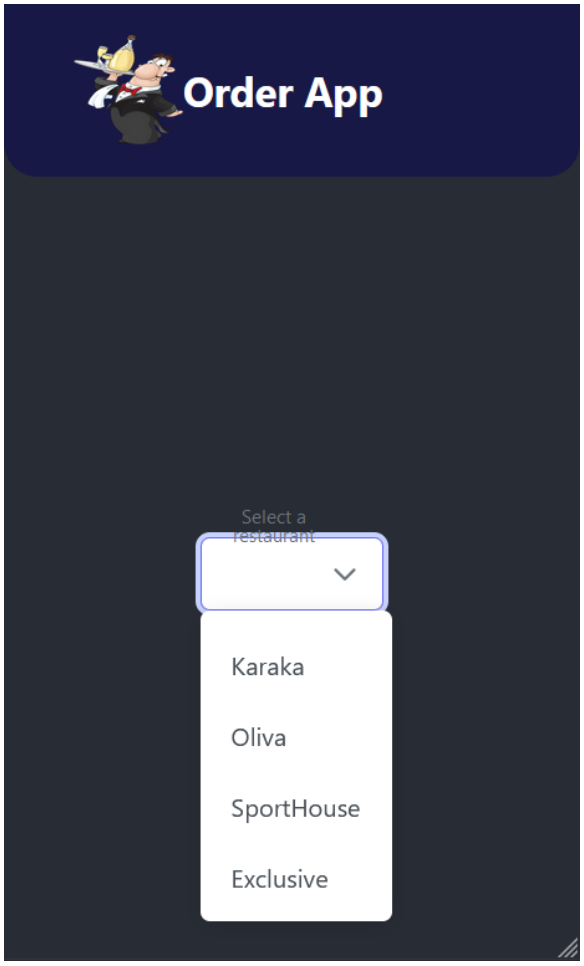
Najvažniji dio mobilne aplikacije jest prikaz spremljenih narudžbi. Narudžbe su prikazane kronološki, dakle redom kojim su zaprimljene se prikazuju na zaslonu. Na lijevoj strani narudžbe nalazi se brojka koja označava broj stola na kojega ju je potrebno poslužiti. Svaka narudžba ima tajmer koji se svake sekunde osvježava i za nijansu mijenja boju narudžbe kao što je vidljivo na slici Sl. 5.5.1. Na slici Sl. 5.5.2. obrisane su prve dvije narudžbe kako bi se pokazala funkcionalnost ikone za brisanje. Osim toga, kreiran je još jedan djelatnik sa inicijalima „MM“ koji je dvostrukim dodirom na narudžbu rezervirao tu narudžbu. Narudžba će se, kao što je vidljivo, prikazati ostalim djelatnicima zaduženima za taj stol, no ona će biti providna i neće ju moći obrisati. Kada djelatnik zauzme narudžbu, on prikaz vidi normalno, samo se pojavljuju njegovi inicijali („GK“ inicijali na narudžbi za stol broj 2).

Sl. 5.5.1. *Prikaz narudžbi*

Sl. 5.5.2. *Prikaz narudžbi nakon zauzimanja i brisanja*

5.6. Web stranica za naručivanje

React stranice sastavljene su od komponenti. Komponenta za odabir uslužnog objekta prikazana je na slici Sl. 5.6.1.



Sl. 5.6.1. *Komponenta za odabir uslužnog objekta*

Nakon odabira objekta, prikazuju se dostupne kategorije proizvoda koje taj objekt nudi. Primjerice, kafići će imati samo tople napitke i hladna pića kao moguće kategorije, dok restorani imaju nešto širi izbor. Klikom na određenu kategoriju otvaraju se proizvodi iz te kategorije. Klikom na „+“ proizvod se sprema, i klijentu se izbaci poruka koja prikazuje koje proizvode je uzeo, u kojoj količini i kolika je ukupna cijena trenutne narudžbe. Sve ovo prikazano je na slici Sl. 5.6.2.

Sl. 5.6.2. *Komponenta za odabir proizvoda i prikaz ukupne narudžbe*

6. ZAKLJUČAK

Mobilni uređaji postali su neizostavna stavka ljudskih života. Vrijeme aktivnog korištenja dnevno mjeri se u satima. Starije generacije također su počele vješto se služiti mobilnim uređajima. Ova aplikacija osmišljena je kako bi olakšala tok informacija od korisnika do djelatnika u uslužnim objektima. Jednostavna je za korištenje, tako da bi se korisnici trebali znati bez problema služiti se aplikacijom. Korištenjem aplikacije smanjuju se moguće greške koje se svim djelatnicima ponekad dogode poput zaboravljanja narudžbe, neprimjećivanja novih klijenata ili grešaka prilikom komunikacije. Osim toga smanjuje se vrijeme čekanja jer djelatnici imaju nešto manje posla u odnosu na standardan tok.

Mogućnost unaprjeđenja aplikacije svakako postoji, svakako bi bilo super dodati mogućnost plaćanja putem mobilne aplikacije. Još jedna ideja bila bi automatski ispis računa pomoću printera na pritisak gumba djelatnika.

Kreiranje Flutter aplikacije i web stranice za naručivanje bilo je poučno, pogotovo jer se autor nikada prije izrade ovoga rada nije susreo sa Flutterom. U učenju je puno pomogla službena dokumentacija koja zbilja detaljno, tekstualno i često videozapisom pojasni korištenje nekog widgeta. Postoje brojni widgeti koje je jednostavno uvesti u program i koristiti za potrebe razvoja aplikacije. Smatram da je aplikacija kvalitetno napravljena i da bi se mogla koristiti u nekom uslužnom objektu.

LITERATURA

[1] Flutter [online], dostupno na https://en.wikipedia.org/wiki/Flutter_(software) [pristupljeno 23. lipnja 2023.]

[2] Impeller rendering engine [online], dostupno na https://docs.flutter.dev/perf/impeller [pristupljeno 23. lipnja 2023.]

[3] Welcome to Skia [online], dostupno na https://skia.org/ [pristupljeno 23. lipnja 2023.]

[4] Dart (programming language) [online], dostupno na https://en.wikipedia.org/wiki/Dart_(programming_language) [pristupljeno 23. lipnja 2023.]

[5] Flutter architectural overview [online], dostupno na https://docs.flutter.dev/resources/architectural-overview [pristupljeno 24. lipnja 2023.]

[6] Visual Studio Code [online], dostupno na https://en.wikipedia.org/wiki/Visual_Studio_Code [pristupljeno 24. lipnja 2023.]

[7] React (software) [online], dostupno na https://en.wikipedia.org/wiki/React_(software) [pristupljeno 10.rujna 2023.]

[8] How Virtual DOM works? [online], dostupno na https://github.com/sudheerj/reactjs-interview-questions#what-is-virtual-dom [pristupljeno 10. rujna 2023.]

SAŽETAK

U ovom radu kreirana je i opisana Flutter aplikacija za lakše izvršavanje narudžbi u uslužnim objektima. Osim toga kreirana je i web stranica pomoću koje se kreiraju narudžbe. Svaki djelatnik može vidjeti samo narudžbe vezane za stolove koje je on preuzeo, može rezervirati narudžbu kako ju ostali djelatnici ne bi zabunom uzeli, te pri izvršenom poslu može obrisati narudžbu. Svaka narudžba ima tajmer koji kreće otkucavati kada klijent spremi svoju narudžbu. Svake sekunde pozadinska boja narudžbe se mijenja te tako od zelene boje nakon deset minuta dolazi do crvene što ukazuje djelatnicima da ukoliko je ikako moguće ranije izvrše tu narudžbu. Web stranica napravljena pomoću React biblioteke, klijent za nekoliko navedenih restorana može otvoriti meni i odabrati željene proizvode. Klijenti do web stranice dolaze skeniranjem QR koda koji se nalazi na stolu.

Ključne riječi: Flutter, dart, QR kod, narudžba, djelatnik

SUMMARY

In this paper, a Flutter application was created and described for easier execution of orders in service facilities. In addition, a website has been created to create orders. Each employee can see only the orders related to the tables that he has taken over, he can reserve the order so that other employees do not take it by mistake, and he can delete the order when the work is done. Each order has a timer that starts ticking when the client saves his order. Every second, the background color of the order changes from green to red after ten minutes, which indicates to the employees that, if possible, they should complete the order earlier. A website built using the React library, the client for several listed restaurants can open a menu and select the desired products. Clients get to the website by scanning the QR code on the table.

Keywords: Flutter, dart, QR code, order, employee

ŽIVOTOPIS

Goran Klement rođen je u mjestu Virovitica, Republika Hrvatska, dana 05.08.1998. godine. Nakon pohađanja opće gimnazije u Slatini upisuje preddiplomski studij na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, smjer računarstvo. Na istom fakultetu upisuje i diplomski studij, smjer informacijske i podatkovne znanosti.

_____

Potpis autora