Razvoj mobilne aplikacije "Barbershop Boss"

Salopek, Teo

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija

Sveučilišni prijediplomski studij Informatika

Teo Salopek

Razvoj mobilne aplikacije „BarberShop Boss“

Završni rad

Mentor:izv.prof.dr.sc. Marija Brkić Bakarić

Rijeka, 23.rujan 2024.

Sažetak

Ovaj završni rad opisuje razvoj mobilne aplikacije **Barber Shop Boss** koja je namijenjena upravljanju rezervacijama i rasporedima u salonu. Aplikacija je razvijena koristeći Flutter, a cilj joj je olakšati vlasnicima salona organizaciju dnevnih aktivnosti, smanjiti administrativne troškove te poboljšati korisničko iskustvo klijenata. Rad detaljno obrađuje sve faze razvoja aplikacije, uključujući analizu zahtjeva, dizajn sučelja, implementaciju funkcionalnosti, te integraciju s bazom podataka. Također rad se bavi izazovima vezanim uz razvoj mobilnih aplikacija i predlaže poboljšanja za buduće verzije. Rezultati su pokazali da aplikacija značajno unapređuje efikasnost poslovanja i zadovoljstvo korisnika. Zaključno **Barber Shop Boss** predstavlja inovativno rješenje za digitalizaciju poslovnih procesa u salonu.

Ključne riječi: Barber Shop Boss; Flutter; mobilna aplikacija; rezervacija termina; upravljanje salonom; baza podataka.

Sadržaj

1. Uvod

U današnje vrijeme s brzim razvojem tehnologije i sve većim zahtjevima tržišta mnogi digitalni alati i rješenja postali su neizostavni dio poslovanja. U svakodnevnom životu koristimo pametne uređaje kao što su mobiteli, tableti i satovi. Svaki od tih uređaja koristi aplikacije koje omogućavaju korisnicima upravljanje i izvršavanje različitih zadataka. Te aplikacije su računalni programi kreirani kako bi korisnicima olakšali svakodnevne aktivnosti poput upravljanja rezervacijama u barber salonima što je upravo cilj aplikacije „*BarberShop Boss"*.

Kada govorimo o platformama na kojima se aplikacije razvijaju bitno je spomenuti dva vodeća operacijska sustava – Android i iOS. iOS koji je razvio Apple koristi se isključivo na Appleovim uređajima poput iPhonea i iPada. S druge strane Android koji je razvio Google slobodno je dostupan i koristi se na velikom broju uređaja različitih proizvođača što ga čini dominantnim operacijskim sustavom u svijetu mobilnih uređaja.

Za razvoj aplikacije „*BarberShop Boss*" izabrana je Android platforma, a korišten je programski jezik Dart u kombinaciji s Flutterom, okvirom za razvoj mobilnih aplikacija. Oboje je razvila tvrtka Google, a njihova integracija omogućava jednostavno i brzo razvijanje aplikacija za više platformi s fokusom na responzivnost i moderno korisničko sučelje. Flutter je poznat po mogućnosti kreiranja visokokvalitetnih aplikacija za Android i iOS što omogućava da aplikacija zadovolji različite korisnike i uređaje.

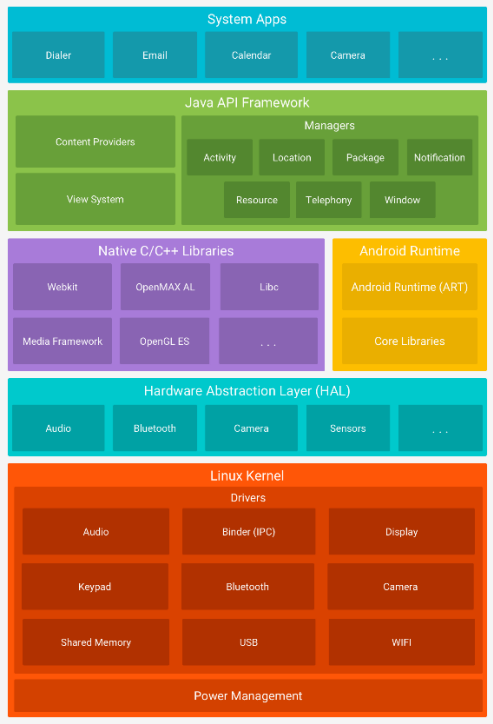
U daljnjim poglavljima rada bit će detaljno opisane korištene tehnologije i ključne funkcionalnosti aplikacije „BarberShop Boss". U drugom poglavlju obrađene su značajke Android operacijskog sustava te tehnologije korištene u razvoju aplikacije s posebnim naglaskom na prednosti korištenja Fluttera i Darta. Treće poglavlje fokusira se na funkcionalnosti aplikacije uključujući sustav za registraciju i prijavu korisnika, kao i upravljanje rezervacijama što je centralni dio aplikacije. U četvrtom poglavlju obrađena je integracija Firebasea za autentifikaciju korisnika i Cloud Firestorea za pohranu podataka. Također je objašnjeno kako ove tehnologije osiguravaju *real-time* sinkronizaciju i sigurnu pohranu podataka. Peto poglavlje opisuje postupak izrade logotipa dok se šesto poglavlje bavi asinkronim programiranjem u Dartu. Sedmo poglavlje fokusira se na testiranje flutter aplikacije s ciljem osiguranja stabilnosti i pouzdanosti u radu.

2. Pregled korištenih tehnologija

Ovo poglavlje daje sažet pregled tehnologija korištenih u izradi aplikacije. Detaljno je objašnjen razvoj i struktura Android operativnog sustava zajedno s opisom Flutter frameworka njegovom arhitekturom i načinom rada. Poseban naglasak stavljen je na mogućnosti Dart programskog jezika koji je korišten za implementaciju aplikacije. Na kraju prikazana je uloga Firestore baze podataka i modernih rješenja koja pruža za upravljanje podacima u mobilnim aplikacijama.

2.1 Android operacijski sustav

Android je danas jedan od najpopularnijih operativnih sustava za mobilne uređaje, a temelji se na Linux kernelu verzije 2.6. Prvotno ga je razvila kompanija Android Inc., koju je kasnije kupila kompanija Google. Ovaj operativni sustav široko je rasprostranjen i koristi se na raznim uređajima uključujući pametne telefone, tablete, pametne satove, pa čak i televizore. Google kontinuirano radi na razvoju Androida i nudi njegovu osnovnu verziju besplatno proizvođačima uređaja. Zbog toga je Android postao dominantna platforma na tržištu mobilnih uređaja što je dodatno potaknuto velikim brojem aplikacija dostupnih u Google Play trgovini [1]. Arhitektura Androida sastoji se od nekoliko slojeva koji uključuju osnovne sistemske komponente [2]. Android runtime okruženje omogućava rad različitih aplikacija i pružanje širokog spektra funkcionalnosti (slika 1).



Slika 1 Android arhitektura. Izvor: [1]

2.2. Tehnologije korištene u razvoju

Flutter je open-source framework za razvoj mobilnih aplikacija razvijen od strane Googlea [3]. Omogućava razvoj aplikacija za iOS, Android pa čak i za web i desktop platforme s jedinstvenim kodom što znatno pojednostavljuje proces razvoja i održavanja aplikacija. Flutter koristi programski jezik Dart koji je također razvijen od strane Googlea.

Prednosti Fluttera uključuju jedinstveni kod za više platformi što omogućuje korisničko iskustvo i uštedu vremena. Također Flutter nudi brzu izgradnju korisničkog sučelja zahvaljujući bogatom setu widgeta i mogućnosti "hot reload" što omogućava instant pregled promjena u sučelju aplikacije. Performanse su visoke jer se Flutter kompilira direktno u izvorni kod što ga čini bržim u odnosu na mnoga druga rješenja za razvoj aplikacija za više platformi.

Flutter je u ovoj aplikaciji korišten za razvoj svih korisničkih sučelja (UI) i interakcija s korisnicima. Svi navigacijski elementi poput ekrana za prijavu, registraciju, pregled recenzija i administratorski panel razvijeni su korištenjem Fluttera.

Firebase

Firebase je sveobuhvatna platforma koju nudi Google, a koja se koristi za razvoj mobilnih i web aplikacija. Ova platforma pruža set backend usluga koje olakšavaju upravljanje ključnim funkcijama kao što su autentifikacija, pohrana podataka, analitika i druge komponente za izradu aplikacija. Ono što čini Firebase korisnim za razvoj aplikacija je činjenica da omogućava implementaciju backend funkcionalnosti bez potrebe za postavljanjem i upravljanjem vlastitim serverima. To smanjuje složenost razvoja i omogućava fokus na izgradnju aplikacijskih funkcionalnosti dok se Firebase brine za infrastrukturu.

Firebase se koristi za autentifikaciju korisnika putem emaila i lozinke pohranu korisničkih podataka i upravljanje rezervacijama. Kroz Firebase Auth aplikacija omogućava registraciju i prijavu korisnika dok Cloud Firestore služi za pohranu i dohvaćanje korisničkih informacija i rezervacija.

Prednosti

Firebase nudi širok spektar usluga koje se jednostavno integriraju u aplikacije putem svojih SDK-ova omogućavajući brzu i laku implementaciju funkcionalnosti. Firebase se automatski prilagođava potrebama aplikacije optimizirajući resurse kako bi učinkovito upravljao velikim količinama podataka i povećanim brojem korisnika [4].

Autentifikacija

Firebase **Authentication** je ključna usluga koja omogućava autentifikaciju korisnika pomoću emaila i lozinke, ali i putem drugih metoda prijave kao što su Google, Facebook i drugi pružatelji identiteta. Firebase Auth pruža jednostavan način za upravljanje korisničkim računima i pristupnim pravima unutar aplikacije.

Jedan od najvažnijih sigurnosnih aspekata ovog sustava je da lozinke korisnika nikada nisu pohranjene u svom izvornom obliku. Umjesto toga Firebase koristi tehnologiju **hashiranja** lozinki čime se osigurava da čak i u slučaju neovlaštenog pristupa bazi podataka lozinke ne mogu biti jednostavno pročitane ili zloupotrijebljene [5].

Autorizacija i sigurnosne mjere

Firebase također pruža mogućnosti definiranja sigurnosnih pravila unutar aplikacije. Kroz **Firebase Security Rules** developeri precizno definiraju tko ima pravo čitanja i mijenjanja određenih podataka. Ova pravila omogućavaju detaljnu kontrolu pristupa čime se osigurava da samo ovlašteni korisnici mogu pristupiti osjetljivim podacima kao što su osobne informacije i rezervacije korisnika.

Pored toga komunikacija između aplikacije i Firebase servera je zaštićena korištenjem **HTTPS protokola** što osigurava kriptiranje svih podataka tijekom prijenosa. Ovaj protokol sprječava neovlaštenu izmjenu podataka između korisničkih uređaja i servera. Također svi podaci pohranjeni u **Cloud Firestore** automatski su kriptirani pomoću naprednih kriptografskih algoritama što pruža dodatnu zaštitu podataka u mirovanju.

Cloud Firestore

Cloud Firestore je Firebaseova NoSQL baza podataka koja omogućava pohranu i sinkronizaciju podataka u stvarnom vremenu. Koristi strukturu dokumenata unutar kolekcija. Ključna prednost Firestorea je *real-time* update funkcionalnost koja automatski osvježava podatke na korisničkom sučelju bez potrebe za ručnim osvježavanjem. Ova značajka je izuzetno korisna u aplikacijama koje zahtijevaju ažuriranje podataka u stvarnom vremenu.

Firestore podržava rad offline omogućujući korisnicima pristup aplikaciji čak i kada nisu povezani na internet. Podaci se automatski sinkroniziraju kada veza ponovno postane dostupna čime se osigurava neprekidno korisničko iskustvo. Osim toga Firestore automatski prilagođava resurse rastu aplikacije i broju korisnika čime omogućava učinkovito upravljanje velikim količinama podataka bez potrebe za ručnim podešavanjem. Ključne značajke Cloud Firestore baze podataka sažeto su prikazane u tablici 1.

	Opis	Prednosti
Real-time update	Automatsko osvježavanje podataka u stvarnom vremenu.	Osigurava uvijek ažurne informacije za korisnike.
NoSQL struktura	Fleksibilna shema podataka bez potrebe za unaprijed definiriranim shemama	Laka prilagodba strukture podataka
Dokument-oblak model	Pohranjuje podatke u dokumente unutar kolekcija.	Logički organizirana pohrana podataka.
Offline podrška	Podrška za rad s podacima i kada je uređaj offline.	Poboljšava korisničko iskustvo.
Sigurnosna pravila	Kontrola pristupa na osnovu korisničkih pravila.	Osigurava zaštitu podataka.
Skalabilnost	Automatsko skaliranje prema potrebama aplikacije.	Rješava probleme s velikim količinama podataka.

Tablica 1 Ključne značajke Cloud Firestore

2.2. Frontend

Frontend aplikacije se odnosi na sve vizualne i interaktivne elemente s kojima korisnici izravno komuniciraju. Frontend je razvijen koristeći Flutter.

Ekrani za Registraciju i Prijavu: Omogućavaju korisnicima stvaranje novih računa i prijavu na postojeće.

Ekran za Upravljanje Rezervacijama: Omogućava korisnicima pregled i upravljanje njihovim rezervacijama.

Administratorski Panel: Omogućava administratorima pregled i upravljanje rezervacijama korisnika.

Sve vizualne komponente aplikacije, uključujući forme, gumbe, liste i obrasce, razvijene su koristeći widgete. Flutterova sposobnost za brzu izradu korisničkog sučelja omogućava lako prilagođavanje i iteraciju dizajna aplikacije.

2.3 Backend

Backend aplikacije obuhvaća svu logiku koja je neophodna za njezin rad. U ovoj aplikaciji backend se temelji na Firebase platformi koja pruža funkcionalnosti za autentifikaciju korisnika i pohranu podataka.

Firebase Auth predstavlja komponentu koja upravlja procesom autentifikacije. Ona omogućava korisnicima registraciju i prijavu
Cloud Firestore služi kao baza podataka u kojoj se pohranjuju svi podaci aplikacije uključujući informacije o korisnicima i rezervacijama. Firestore omogućava brzo i efikasno dohvaćanje podataka.

Implementacija backend sustava koristi Firebase kako bi se učinkovito upravljalo svim funkcionalnostima pohrane podataka i autentifikacije. Sve operacije vezane uz podatke kao što su dodavanje novih korisnika, dohvaćanje rezervacija i upravljanje postojećim zapisima obavljaju se putem Cloud Firestorea.

3. Funkcionalnosti i korisničko sučelje

3.1 Početna stranica

Na početnoj stranici korisnicima se prikazuje logo aplikacije te gumb s natpisom "Rezerviraj Frizuru". Cijeli dizajn je fokusiran na brzu interakciju bez suvišnih elemenata koji bi mogli ometati korisnika. Struktura stranice implementirana je pomoću Flutterovog Scaffold widgeta koji pruža osnovni okvir za vizualni izgled stranice dok je Container widget korišten za pozicioniranje logotipa i gumba. Pozadinske boje i elementi prilagođeni su estetici aplikacije (slika 2).

Slika 2 Početna Stranica

3.2. Korisničko sučelje

3.2.1. Signup (registracija)

Ekran za registraciju omogućava novim korisnicima da kreiraju svoj račun u aplikaciji. Ovaj ekran služi kao prvi korak za pristup svim funkcionalnostima aplikacije. Korisnici moraju unijeti svoje osnovne podatke kako bi se uspješno registrirali i pristupili aplikaciji.

Funkcionalnosti:

Korisnici unose svoje ime što je važno za prilagodbu njihove identifikacije u aplikaciji. Email adresa djeluje kao jedinstveni identifikator prilikom prijave, ali i kao sredstvo za komunikaciju s korisnikom. Lozinka je ključna za zaštitu korisničkog računa, a njezin prikaz je skriven kako bi se očuvala privatnost.Ekran za registraciju obuhvaća polja za unos imena, emaila i lozinke te uključuje gumbe za registraciju i navigaciju prema ekranu za prijavu (slika 3)

Ako neko od obaveznih polja (ime, email, lozinka) ostane prazno korisnik će primiti obavijest o potrebi da ih ispuni (slika 4). U slučaju neispravnog emaila korisniku će biti prikazana poruka o grešci.

Korisnički podaci uključujući email i lozinku šalju se Firebase Auth servisu koji upravlja registracijom i prijavom korisnika. Ostali korisnički podaci poput imena i ID-a pohranjuju se u Cloud Firestore unutar kolekcije "users" što omogućava jednostavno pohranjivanje i dohvaćanje informacija o korisnicima.

Slika 3 Registracija (Ime, Email, Password)

Ekran prikazuje kako aplikacija reagira kada korisnik ostavi prazna polja i pokuša se registrirati (slika 4).

Slika 4 Greška za prazna polja

3.2.2. Login (prijava)

Ekran za prijavu omogućava postojećim korisnicima da se prijave u aplikaciju koristeći svoje prethodno registrirane podatke. Ovaj ekran služi kao ulaz u aplikaciju i omogućava korisnicima pristup funkcionalnostima koje su im dostupne nakon prijave.

Korisnici trebaju unijeti email koji su koristili prilikom registracije. Ovaj email služi kao jedinstveni identifikator za njihov korisnički račun čime se osigurava da se prijavi ispravna osoba. Lozinka koju je korisnik odabrao tijekom registracije tijekom unosa skrivena je *obsecureText* metodom.

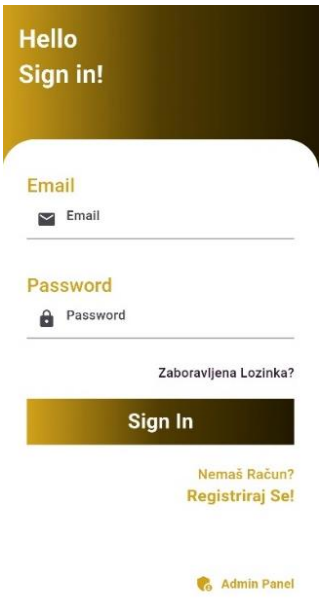
Aplikacija se oslanja na Firebase Auth za provjeru unesenih podataka u odnosu na pohranjene informacije. Ako su email i lozinka točni korisnik se uspješno prijavljuje i preusmjerava na glavni ekran aplikacije. U slučaju pogrešnih podataka kao što su netočan email ili lozinka korisnicima se prikazuje obavijest o grešci koja im omogućava da isprave unesene podatke.

Relevantne informacije o ključnim elementima i tehnologijama koje čine proces prijave jednostavnim i učinkovitim i poboljšavaju korisničko iskustvo unutar aplikacije prikazane su u tablici 2.

Element	Opis	Tehnologija
Ekran za login	Ekran za prijavu korisnika s poljima za unos emaila i lozinke.	Flutter UI
Validacija podataka	Provjera ispravnosti unesenih podataka i prikaz poruka o greškama.	Flutter Form Validation
Autentifikacija	Provjera korisničkih podataka u Firebase Authentication.	Firebase Authentication
Prikaz pogrešaka	Prikaz poruka o grešci ako unos nije ispravan.	Flutter SnackBar, AlertDialog

Tablica 2 Login funkcionalnost

Ekran za prijavu prikazuje polja za unos emaila i lozinke te gumb za prijavu. Također sadrži opciju za povratak na ekran registracije (slika 5).



Slika 5 Prijava (Email, Password)

Ekran prikazuje obavijesti o grešci ako korisnik unese pogrešne podatke (slika 6).

Slika 6 Neuspješna Prijava (Lozinka),

3.3 Glavna stranica

Glavna stranica aplikacije služi kao centralno mjesto s kojeg korisnici mogu jednostavno pristupiti različitim funkcionalnostima.

Temelji se na Flutterovom Scaffold widgetu koji pruža osnovnu strukturu. Korisnicima su dostupne različite usluge predstavljene kroz vizualne kartice koje sadrže slike i nazive usluga (slika 7) . Ovaj vizualni pristup olakšava korisnicima pronalaženje željene usluge. Usluge uključuju "Šišanje", "Uređivanje Brade", "Šišanje + Brada", "Dugo Šišanje", "Kompletni Tretman" i "Pranje Kose". Kada korisnik odabere neku od usluga automatski se preusmjerava na ekran za rezervaciju koristeći Navigator.push metodu što omogućava navigaciju između ekrana.

Slika 7 Glavna stranica i usluge

Na glavnoj stranici također se prikazuju postojeće rezervacije korisnika koristeći StreamBuilder za *real-time* ažuriranje informacija. Ova funkcionalnost omogućuje korisnicima pregled svih zakazanih termina uključujući detalje poput naziva usluge, datuma i vremena. Osim toga korisnici imaju mogućnost uklanjanja rezervacija jednim dodirom što pruža fleksibilnost i jednostavnost korištenja aplikacije (slika 8).

Slika 8 Postojeće rezervacije korisnika

Navigacija do rezervacije usluge

Kada korisnik odabere određenu uslugu automatski se preusmjerava na ekran za rezervaciju. Ovdje će korisnik imati mogućnost da unese sve potrebne podatke za zakazivanje kao što su datum i vrijeme (slika 9). Navigacija do ovog ekrana omogućena je korištenjem `Navigator.push` metode.

Na ekranu za rezervaciju korisnici mogu odabrati datum i vrijeme za svoju rezervaciju. Datum se bira pomoću `showDatePicker` dijaloga dok se vrijeme odabire iz unaprijed definiranih vremenskih slotova. Ovi alati pomažu korisnicima da precizno odrede željeni termin rezervacije.

Spremanje rezervacije

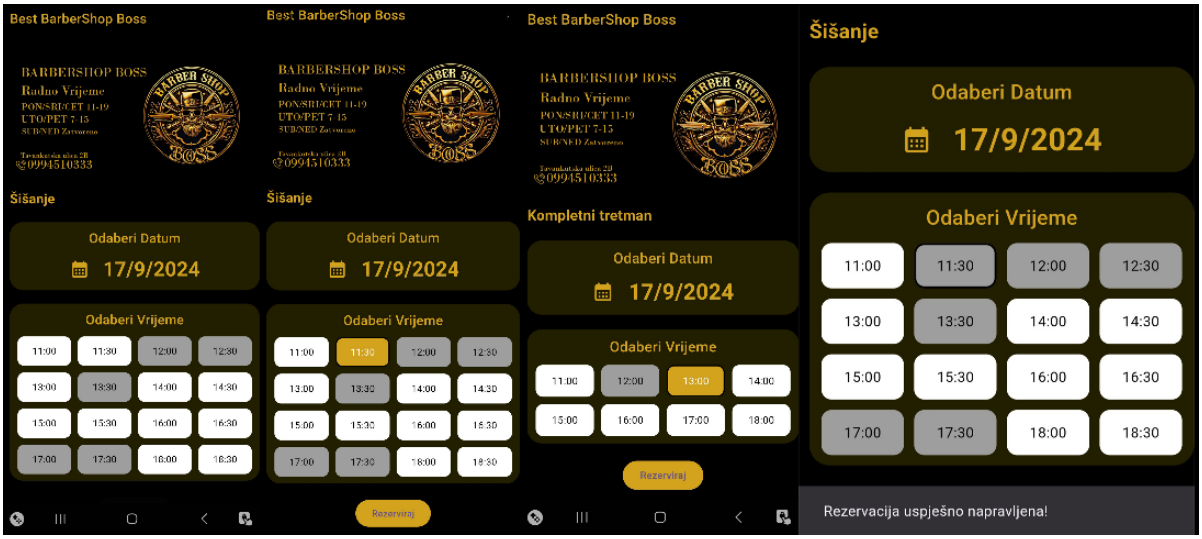
Nakon što korisnik završi s unosom svih potrebnih informacija na ekranu za rezervaciju podaci se pohranjuju u Cloud Firestore. Ovaj korak omogućava administratoru da vidi sve rezervacije.

Podaci o rezervaciji koji se pohranjuju:

- **Usluga**
- **Korisničko ime**
- **Datum**
- **Vrijeme**

Svaka rezervacija se sprema u kolekciju pod nazivom „Booking“.

Za implementaciju koristimo metodu `addUserBooking` unutar klase `DatabaseMethods` koja dodaje podatke o rezervaciji u Firestore bazu podataka. Time se osigurava da su svi podaci pravilno spremljeni i dostupni za daljnju obradu.



Slika 9 Proces rezervacije

3.4. Administratorski panel

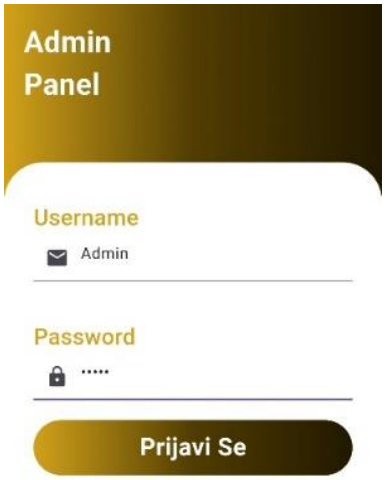
3.4.1. Admin login (administratorska prijava)

Ekran za prijavu administratora omogućava administratorima da se prijave u administratorski panel aplikacije. Ovaj panel omogućava upravljanje rezervacijama i pregled podataka o korisnicima. Ekran za prijavu zahtijeva od administratora da unese svoje korisničke podatke kako bi pristupio funkcionalnostima koje su specifične za administratore.

Administrator unosi jedinstveni korisnički ID koji je prethodno pohranjen u Firebase kolekciji "Admin". Ovaj ID služi kao prepoznatljiv identifikator za svakog administratora u sustavu. Lozinka koju administrator unosi kreirana je tijekom ručne registracije u istoj kolekciji.

Aplikacija koristi Firebase Firestore za verifikaciju unesenih podataka uspoređujući ih s postojećim zapisima u kolekciji "Admin". U slučaju da su korisnički ID i lozinka točni administrator je uspješno prijavljen te preusmjeren na administratorski panel. Ako podaci nisu ispravni aplikacija prikazuje poruku o grešci omogućujući administratoru da ispravi unesene informacije.

Ekran za prijavu adminstratora prikazuje polja za unos korisničkog ID-a i lozinke te gumb za prijavu (slika 10).



Slika 10 Unos Username i Password

3.4.2. Booking management (upravljanje rezervacijama)

Ekran za upravljanje rezervacijama omogućuje administratorima pregled i upravljanje svim rezervacijama koje su napravljene kroz aplikaciju. Ovaj ekran pruža pregled svih aktivnih rezervacija i omogućuje administratoru da obriše rezervacije koje su obrađene ili više nisu potrebne. Rezervacije se prikazuju u stvarnom vremenu što omogućava administratorima da uvijek imaju ažurne informacije.

Prikaz svih rezervacija u stvarnom vremenu

Ekran za upravljanje rezervacijama koristi `StreamBuilder` za prikaz svih rezervacija pohranjenih u kolekciji "Booking" u Cloud Firestore pružajući automatsko ažuriranje u stvarnom vremenu. Ova funkcionalnost osigurava da svi detalji uključujući uslugu, ime korisnika, datum i vrijeme rezervacije budu uvijek ažurni. Administrator može obrisati rezervacije koje su obrađene ili više nisu relevantne uz dodatnu potvrdu kako bi se izbjegle greške. Rezervacije su grupirane i sortirane po vremenu i datumu. Prikaz rezervacija iz perspektive korisnika i administratora pruža jasnu sliku o funkcionalnosti sustava za upravljanje rezervacijama (slika 11).

Slika 11 Streambuilder (korisnik i admin view)

4. Implementacija

Firebase Auth omogućuje jednostavno upravljanje korisničkim autentifikacijama kroz mobilne aplikacije. U ovoj aplikaciji koristi se za registraciju i prijavu korisnika te za prijavu administratora pružajući sigurno i učinkovito rješenje za upravljanje korisničkim računima i pristupom aplikaciji. Kôd za autentifikaciju pokazuje osnovne funkcionalnosti prijave i registracije dok slike iz Firebase konzole pomažu u vizualizaciji konfiguracije i upravljanja korisnicima.

4.1. Firebase autentifikacija

Firebase Auth pruža jednostavan i siguran način za autentifikaciju korisnika u mobilnim aplikacijama. Pruža različite metode autentifikacije uključujući prijavu putem emaila i lozinke što omogućava developerima da lako upravljaju korisničkim računima.

Registracija korisnika

Korisnici mogu kreirati nove račune pomoću svoje email adrese i lozinke. Firebase Auth automatski obrađuje proces registracije i pohranjuje korisničke podatke na siguran način (slika 16). Isječak koda prikazan je na slici 12.

```
Future<void> registration() async {
  if (_formkey.currentState?.validate() ?? false) {
    // Preuzmi unesene podatke
    String? name = namecontroller.text;
    String? email = emailcontroller.text;
    String? password = passwordcontroller.text;

    if (name == null || email == null || password == null) {
      ScaffoldMessenger.of(context).showSnackBar(SnackBar(
        content: Text(
          "Molimo unesite sve podatke!",
          style: TextStyle(fontSize: 20),
        ), // Text
      )); // SnackBar
      return;
    }

    try {
      // Registriraj korisnika
      UserCredential userCredential = await FirebaseAuth.instance
          .createUserWithEmailAndPassword(email: email, password: password);

      User? user = userCredential.user;
      String id = randomAlphaNumeric(10);

      // Ažuriraj Firebase Authentication profil s imenom korisnika
      if (user != null) {
        await user.updateProfile(displayName: name);
        await user.reload();
      }
```

Slika 12 Registracija korisnika

Prijava korisnika

Korisnici mogu pristupiti svom računu koristeći email adresu i lozinku. Firebase Auth provjerava podatke i omogućuje pristup samo ako su uneseni podaci ispravni. Isječak koda prikazan je na slici 13.

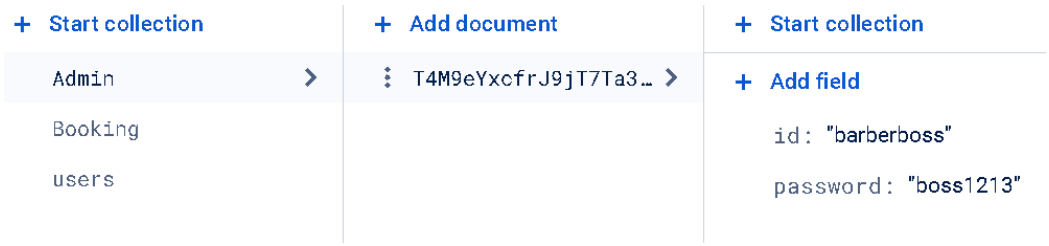
```
Future<void> userLogin() async {
  try {
    UserCredential userCredential =
        await FirebaseAuth.instance.signInWithEmailAndPassword(
      email: email!,
      password: password!,
    );

    await saveUserDetails(userCredential.user!);
```

Slika 13 Prijava korisnika

Prijava administratora

Administrator se prijavljuje koristeći jedinstveni ID i lozinku koja je pohranjena u Firebase kolekciji "Admin" (slika 14). Ova metoda omogućava administratorima da pristupe administratorskim funkcijama unutar aplikacije. Za dodavanje novih administratora potrebno je ručno unijeti njihove podatke u Firebase kolekciju "Admin".



Slika 14 Kolekcija „Admin“

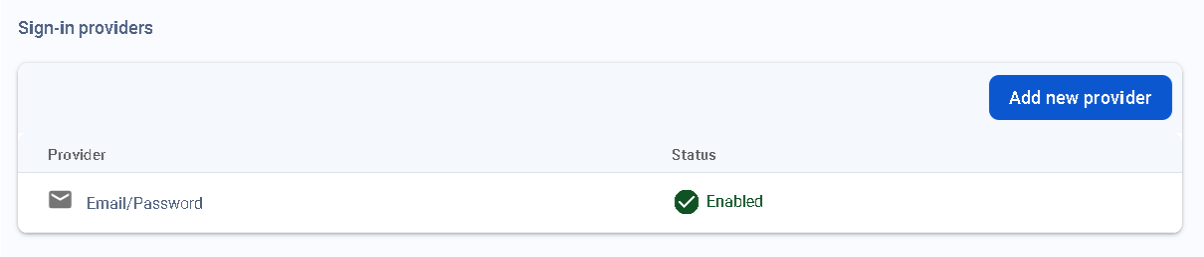
Postavljanje Firebase Autentifikacije

Kreiranje Firebase projekta: Kreiranje novog Firebase projekta započinje prijavom na Firebase Console. Nakon prijave odabire se opcija "Add Project" i slijede se upute za postavljanje projekta.

Konfiguracija autentifikacije: Nakon kreiranja projekta potrebno je pristupiti odjeljku "Authentication" u Firebase konzoli. Tamo se odabire kartica "Sign-in method" (slika 15) gdje

se omogućuju željene metode prijave kao što su Email/Password. Ako su potrebne dodatne metode prijave kao što su Google ili Facebook konfiguriraju se prema specifičnim zahtjevima projekta.

Integracija Firebase u projekt: Za integraciju Firebase u Flutter projekt slijede se upute iz Firebase konzole za dodavanje Firebase SDK-a. Ovo uključuje dodavanje datoteke google-services.json za Android u pripadajući direktorij unutar projekta.



Slika 15 Signed In Method

Slika 16 Users (signed In)

4.2. Cloud Firestore

Cloud Firestore je NoSQL baza podataka u oblaku koju pruža Google Firebase. Ona omogućuje pohranu, sinkronizaciju i dohvaćanje podataka u stvarnom vremenu za mobilne i web aplikacije [6]. Firestore koristi dokumente i kolekcije za organizaciju podataka što omogućava fleksibilno i skalabilno upravljanje podacima.

Ključne značajke Cloud Firestore-a:

Struktura podataka:

Dokumenti: Osnovna jedinica pohrane podataka u Firestore-u. Svaki dokument može sadržavati razne tipove podataka uključujući stringove, brojeve, nizove, objekte itd.

Kolekcije: Grupa dokumenata. Dokumenti unutar kolekcije mogu imati svoje podkolekcije što omogućava složenu hijerarhiju podataka.

Sinkronizacija u stvarnom vremenu:

Firestore omogućuje automatsko ažuriranje podataka u stvarnom vremenu. Svaka promjena u bazi podataka odmah se odražava na uređajima koji su povezani s bazom podataka.

Dodavanje podataka:

Dodavanje podataka u Firestore uključuje kreiranje novog dokumenta unutar određene kolekcije. Ako dokument s određenim ID-om već postoji podaci će biti ažurirani. Isječak koda za dodavanje podataka prikazan je na slici 17.

```
Future addUserDetails(Map<String, dynamic> userInfoMap, String id) async {
  return await FirebaseFirestore.instance
      .collection("users")
      .doc(id)
      .set(userInfoMap);
}
```

Slika 17 Dodavanje podataka

Dohvaćanje podataka:
Dohvaćanje podataka iz Firestore može biti jednokratno (query) ili putem streama za praćenje promjena u stvarnom vremenu. Isječak koda za dohvaćanje podataka prikazan je na slici 18.

```
Future<Stream<QuerySnapshot>> getUserBookings(String username) async {
  return FirebaseFirestore.instance
      .collection("Booking")
      .where("username", isEqualTo: username)
      .snapshots();
}
```

Slika 18 Dohvaćanje podataka

Brisanje podataka:

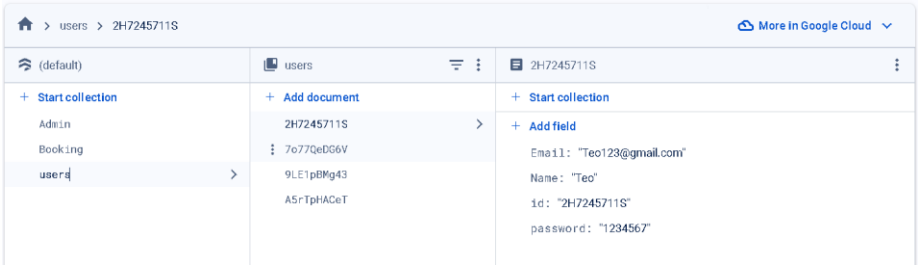
Brisanje podataka iz Firestore uključuje uklanjanje dokumenta iz kolekcije. Isječak koda za brisanje podataka prikazan je na slici 19.

```
Future DeleteBooking(String id) async {
  return await FirebaseFirestore.instance
      .collection("Booking")
      .doc(id)
      .delete();
}
```

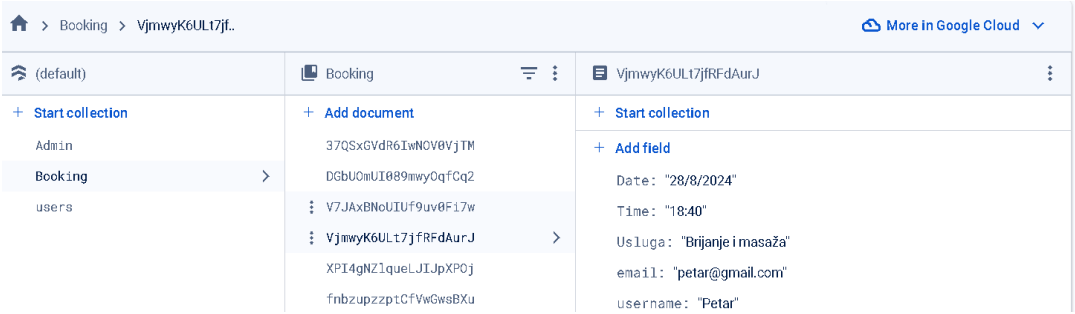
Slika 19 Brisanje podataka

Strukturirani prikaz Firestore baze podataka:
Primjeri struktura:

- **Kolekcija "users"** s dokumentima koji sadrže korisničke podatke (slika 20).
- **Kolekcija "Booking"** s dokumentima koji sadrže podatke o rezervacijama uključujući korisničko ime, uslugu, datum i vrijeme (slika 21).



Slika 20 Users



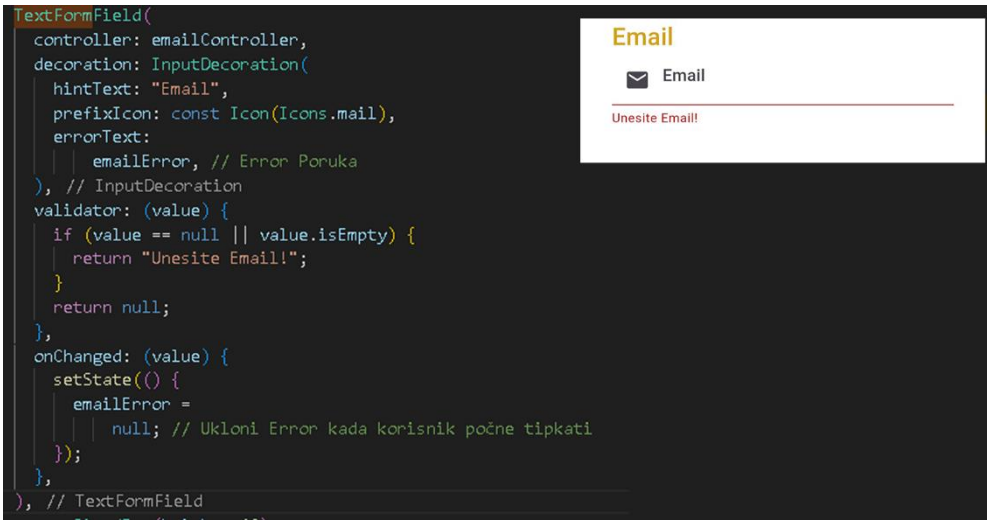
Slika 21 Booking

4.3. Flutter Widgeti i UI

Flutter pruža bogat skup widgeta koji omogućuju razvoj interaktivnog i vizualno privlačnog korisničkog sučelja. Widgeti predstavljaju osnovne građevne blokove u Flutteru koji omogućuju izradu vizualnog i funkcionalnog dijela aplikacije. Svaki widget je element korisničkog sučelja i može se kombinirati s drugim widgetima kako bi se kreirale složenije UI strukture [7].

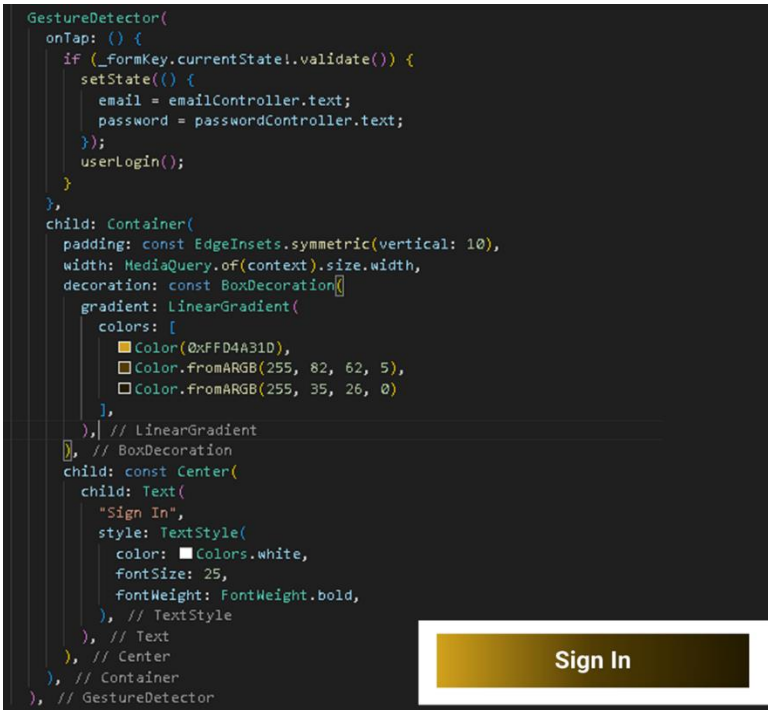
Ključni Flutter widgeti u ovoj aplikaciji

TextFormField widget koristi se za unos tekstualnih podataka kao što su ime, email ili lozinka. Ovaj widget omogućava korisnicima unos informacija i pruža podršku za validaciju unosa osiguravajući da su podaci ispravno uneseni prije obrade. Kod za unos emaila prikazan je na slici 22.



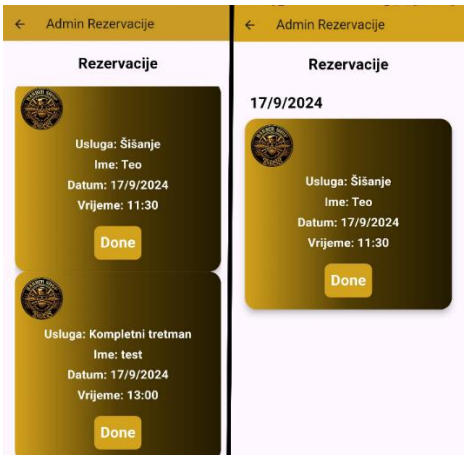
Slika 22 TextFormField Widget za unos korisničkih podataka

GestureDetector widget prepoznaje dodirne geste i omogućuje interakciju s korisničkim sučeljem. Koristi se za detekciju klikova na elementima kao što su gumbi za prijavu ili registraciju te pokreće odgovarajuće funkcije unutar aplikacije. Kod koji ilustrira ovu funkcionalnost zajedno s izgledom gumba prikazan je na slici 23."



Slika 23 GestureDetector gumb za prijavu

StreamBuilder widget omogućava prikaz podataka koji se automatski ažuriraju u stvarnom vremenu (slika 24). Povezuje se s "streamovima" podataka i ažurira korisničko sučelje kada se podaci mijenjaju. Koristi se za prikaz rezervacija koje se učitavaju iz baze podataka.



Slika 24 StreamBuilder Widget prikazuje listu rezervacija (Admin)

5. Logotip

Izrada logotipa započela je detaljnom analizom i usporedbom svih logotipova barbershopova u Hrvatskoj. Ova analiza omogućila je prikupljanje inspiracija i prepoznavanje trendova u dizajnu što je bilo ključno za razvoj jedinstvenog logotipa. Nakon stečene ideje i dizajna proces je nastavljen izradom skice u alatu **Adobe Firefly**. Ovaj alat koristi umjetnu inteligenciju za generiranje slika na temelju tekstualnih upita i omogućava brzo eksperimentiranje s različitim vizualnim elementima.

Adobe Firefly nudi razne mogućnosti te kroz nekoliko upita koji su se fokusirali na brijačke simbole i moderne elemente, generirane su različite skice (slika 25). Nakon pregleda generiranih slika jedna skica je odabrana kao najbolja za daljnji rad.

Slika 25 Skice generiranih slika

Odabrana skica prenesena je u **Adobe Illustrator** gdje je postavljena kao template. Korišteni su različiti alati za precizno oblikovanje uključujući alate za crtanje, bojanje i rad s tekstom čime je logotip dobio profesionalan i privlačan izgled (slika 26).

Konačno uz prilagodbu boja i balansiranje svih elemenata dobijen je logotip koji osigurava da dizajn bude prepoznatljiv i u skladu s brendom „BarberShop Boss“.

Slika 26 Logotip "BarberShop Boss"

6. Asinkrono programiranje u Dartu

Asinkrono programiranje u Dartu omogućava izvršavanje zadataka koji traju dulje vremena bez blokiranja glavnog toka aplikacije čime se poboljšava korisničko iskustvo. Flutter aplikacije često koriste asinkrono programiranje za operacije poput dohvaćanja podataka iz baze interakcije s bazom podataka ili izvođenje dugotrajnih zadataka [2].

Prednosti asinkronog programiranja

Prednosti asinkronog programiranja u Flutteru donose brojne koristi posebno u kontekstu responzivnosti aplikacije. Glavni tok aplikacije ostaje neblokiran što omogućava korisnicima neprekidnu interakciju čak i dok se zadaci izvršavaju u pozadini. To značajno poboljšava korisničko iskustvo.

Pored toga asinkroni zadaci povećavaju učinkovitost korištenjem sistemskih resursa na optimiziran način. Operacije poput dohvaćanja podataka ili interakcije s bazama podataka mogu se obavljati neovisno o glavnom tijeku što doprinosi bržem i stabilnijem radu aplikacije.

Korištenje ključnih riječi async i await dodatno pojednostavljuje logiku aplikacije. Ovaj način pisanja koda čini ga čitljivijim i lakšim za razumijevanje, a upravljanje s više Future objekata postaje jednostavnije što olakšava razvoj i održavanje složenih procesa.

Asinkrono programiranje omogućava paralelizam u aplikacijama što je ključno u modernim aplikacijama koje zahtijevaju mrežne operacije ili rukovanje velikim količinama podataka [2].

Primjena Future i async/await

Korištenje Future omogućava definiranje operacija koje se izvršavaju u budućnosti. Kada operacija završi Future vraća rezultat. Kombinacija async i await pojednostavljuje rad s Future objektima te čini kod čitljivijim.

Funkcija `getthedatafromsharedpref()` poziva metode za dohvaćanje korisničkog imena i e-maila iz lokalne pohrane. Kada se pozovu te metode program čeka da se podaci vrate pomoću await. Nakon što se podaci uspješno dobiju oni se pohranjuju u varijable name i email. Pozivom `setState()` osigurava se da se korisničko sučelje ponovno učita s novim informacijama čime se održava responzivnost aplikacije. Ovaj pristup omogućava korisnicima da nastave interakciju s aplikacijom dok se podaci učitavaju umjesto da čekaju da se svi podaci preuzmu prije nego što nastave. Implementirani kod koji prikazuje ovu funkcionalnost može se vidjeti na slici 27.

```
Future<void> getthedatafromsharedpref() async {
  name = await SharedpreferenceHelper().getUserName();
  email = await SharedpreferenceHelper().getUserEmail();
  setState(() {});
}
```

Slika 27 Primjena Future i async/await

Dohvaćanje podataka iz Firestore

Kada se podaci dohvaćaju iz Firestore baze podataka koristi se asinkrona funkcija kako bi se dobila lista rezervacija. Ovaj pristup također pomaže u smanjenju opterećenja na mrežu jer se podaci preuzimaju samo kada su potrebni, a ne unaprijed. Isječak koda prikazan je na slici 28.

```
Future<void> _initBookings() async {
  futureBookingStream = Future.value(DatabaseMethods().getBookings());
}
```

Slika 28 Dohvaćanje podataka

Metoda `getBookings()` koristi se za dohvaćanje rezervacija, a rezultati se vraćaju kao Future. Korištenjem `Future.value` omogućava se rad s tim podacima bez blokiranja glavne niti. Stream koji se dobije od ove metode koristi se za automatsko ažuriranje sučelja kada se podaci promijene. Ovaj pristup osigurava da korisnici uvijek vide ažurne informacije o rezervacijama što je od važnosti u situacijama gdje se korisnički unos može brzo mijenjati (slika 28). Asinkroni pristup omogućava da se aplikacija fokusira na korisničko iskustvo dok se u pozadini upravlja složenim operacijama dohvaćanja podataka.

7. Testiranje flutter aplikacije

Testiranje je ključno za osiguranje kvalitete i stabilnosti aplikacije. Flutter pruža razne alate i okvire za testiranje uključujući unit testove, widget testove i integracijske testove. Ovi testovi omogućuju provjeru ispravnosti funkcionalnosti i performansi aplikacije te pomažu u pronalaženju i rješavanju bugova. Testovi se pokreću pomoću naredbe `flutter test`.

U procesu razvoja ove aplikacije većina testova je provedena ručno na stvarnom mobilnim uređaju. Ručno testiranje omogućilo je detaljnu provjeru funkcionalnosti u stvarnim uvjetima uključujući interakcije korisnika i performanse aplikacije na različitim modelima uređaja. Ovaj pristup osigurao je da su sve funkcionalnosti UI elementi i korisničke interakcije optimizirane za mobilne platforme, a eventualne greške brzo su otkrivene i ispravljene.

Unit testovi

Unit testovi testiraju pojedinačne funkcije ili klase neovisno o ostatku aplikacije. Oni su brzi i pomažu u provjeri da li osnovne funkcionalnosti rade ispravno. Unit test za registraciju korisnika prikazan je na slici 29 gdje se provjerava ispravnost registracije.

```
Run | Debug
testWidgets('Signup registration test', (WidgetTester tester) async {
  await tester.pumpWidget(MaterialApp(home: Signup()));

  // Popunjavanje polja
  await tester.enterText(find.byType(TextFormField).at(0), 'Test');
  await tester.enterText(find.byType(TextFormField).at(1), 'test@gmail.com');
  await tester.enterText(find.byType(TextFormField).at(2), '12345678');


  // Simulacija pritiska na dugme "Registriraj se"
  await tester.tap(find.text('Registriraj se'));
  await tester.pump();


  expect(find.text('Uspješno Registriran!'), findsOneWidget);
});
```

Slika 29 Unit test

Ovaj test simulira registraciju korisnika tako što unosi ime, email i lozinku u odgovarajuća polja. Nakon što korisnik pritisne dugme za registraciju test provjerava da li se ispravna poruka o uspješnoj registraciji prikazuje. Ovaj test osigurava da logika registracije Funkcionira kako treba.

Widget testovi

Widget testovi testiraju pojedinačne UI elemente kako bi osigurali da se pravilno prikazuju i ponašaju. Ovi testovi simuliraju interakcije s korisničkim sučeljem i provjeravaju da li je ponašanje aplikacije ispravno.

Efikasnost widget testova ključna je u Flutter aplikacijama jer omogućavaju brzo testiranje vizualnih komponenti bez potrebe za simulacijom cijele aplikacije što štedi vrijeme i resurse prilikom razvoja [8]. Primjer widget testa za onboarding komponentu prikazan je na slici 30, a uključuje provjeru ispravnosti prikaza i interakcije s dugmetom.

```
void main() {
  Run | Debug
  testWidgets('Onboarding widget test', (WidgetTester tester) async {
    // Učitavanje Onboarding widget-a
    await tester.pumpWidget(MaterialApp(
      home: Onboarding(),
    )); // MaterialApp

    // Provjera da li se slika prikazuje
    expect(find.byType(Image), findsOneWidget);

    // Provjera da li se dugme "Rezerviraj Frizuru!" prikazuje
    expect(find.text('Rezerviraj Frizuru!'), findsOneWidget);

    // Simulacija pritiska na dugme
    await tester.tap(find.text('Rezerviraj Frizuru!'));
    await tester.pumpAndSettle();

    // Provjeri da li je navigacija na Login stranicu
    expect(find.byType(Login), findsOneWidget);
  });
}
```

Slika 30 Widget test

Ovaj test provjerava UI elemente na ekranu. Osigurava da se slika i dugme ispravno prikazuju kao i da pravilno reagira na korisničku interakciju (pritisak na dugme). Test također provjerava da li se korisnik uspješno preusmjerava na stranicu za prijavu nakon klika.

Integracijski testovi

Integracijski testovi ispituju međusobnu interakciju različitih dijelova aplikacije uključujući komunikaciju između modula i vanjskih sustava. Oni omogućavaju provjeru cjelokupnog toka aplikacije uključujući mrežne zahtjeve i integracije s bazama podataka. Na slici 31 prikazan je kod koji testira registraciju korisnika uz validaciju unosa.

```dart
testWidgets('Signup registration with validation test', (WidgetTester tester) async {
await tester.pumpWidget(MaterialApp(home: Signup()));

// Pokušaj pritisnuti dugme bez unosa podataka
await tester.tap(find.text('Registriraj se'));
await tester.pump();

// Proveri da li se prikazuje poruka o grešci za ime
expect(find.text('Unesite ime!'), findsOneWidget);

// Popuni polja
await tester.enterText(find.byType(TextFormField).at(0), 'Test');
await tester.enterText(find.byType(TextFormField).at(1), 'invalid-email'); // neispravan email

// Pritisni dugme
await tester.tap(find.text('Registriraj se'));
await tester.pump();

// Proveri grešku za email
expect(find.text('Unesite Email!'), findsOneWidget);
});
```

Slika 31 Integracijski test

Ovaj test simulira proces registracije gdje korisnik pokušava da se registrira bez unosa podataka. Prvo se provjerava da li se prikazuje poruka o grešci za ime, a zatim se unosi pogrešan email. Test provjerava da aplikacija pravilno reagira na pogrešne unose osiguravajući da korisnik dobije odgovarajuće upozorenje.

Testiranje u Flutteru donosi brojne prednosti koje značajno unapređuju proces razvoja aplikacija. Jedna od ključnih prednosti je brzo otkrivanje grešaka jer testovi omogućuju rano prepoznavanje problema i bugova daleko prije nego što aplikacija stigne do krajnjih korisnika. Ovo ne samo da štedi vrijeme već i sprječava poteškoće u produkciji.

Pored toga integracijski testovi igraju važnu ulogu u osiguravanju stabilnosti aplikacije. Oni provjeravaju kako različiti dijelovi aplikacije međusobno funkcioniraju te pružaju developerima da se uvjere da svi moduli ispravno surađuju i da nema problema u komunikaciji između komponenti.

Zaključak

Zadatak ovog završnog rada bio je razvoj mobilne aplikacije "BarberShop Boss" koja je namijenjena upravljanju rezervacijama u salonu. Korištenjem modernih tehnologija kao što su Flutter i Dart aplikacija je razvijena s ciljem poboljšanja korisničkog iskustva, optimizacije poslovanja te smanjenja administrativnih troškova.

Tijekom razvoja Flutter se pokazao kao izuzetno učinkovit framework za izradu mobilnih aplikacija zahvaljujući svojoj mogućnosti kreiranja aplikacija za više platformi iz jednog koda. Dart kao programski jezik koji je integriran s Flutterom omogućio je brzu implementaciju korisničkog sučelja dok je Firebase platforma pružila siguran i prilagodljiv backend za upravljanje korisničkim podacima i rezervacijama.

Kroz analizu i implementaciju jasno je uočeno da Flutter zajedno s Dartom ne samo da omogućava visokokvalitetno korisničko sučelje već i smanjuje složenost razvoja aplikacija. Ove tehnologije osigurale su brzo testiranje, responzivnost aplikacije te sigurnu pohranu podataka što su faktori za uspješnu digitalizaciju poslovnih procesa u barber salonima.

Na temelju rezultata ovog rada može se zaključiti da je "BarberShop Boss" inovativno rješenje koje koristi napredne tehnologije za unaprjeđenje poslovanja te postavlja solidne temelje za budući razvoj i proširenje funkcionalnosti aplikacije.

Literatura

[1] DeveloperAndroid, »DeveloperAndroid,« Google, 2023. Available: https://developer.android.com/guide/platform.

[2] D. S. &. A. Jonathan Sande, »Data Strcutres & algortihms in Dart,« u *Algorithmic Thinking Made Easy*, Jonathan Sande, 2023..

[3] G. D. (2023)., »Flutter.dev,« Google, 2023. [Mrežno]. Available: https://flutter.dev/docs.

[4] G. F. (FireBase, »Google Firebase.,« Google, 2023. Available: https://firebase.google.com/docs.

[5] G. C. (F. FireBase Security, »FireBase Security,« Google, 2023. Available: https://firebase.google.com/docs/rules.

[6] M. -. F. (2023)., »FIDIT-Merlin,« Srce. Available: https://merlin.srce.hr.

[7] Coursera, »Flutter,« Coursera, Inc., Available: https://www.coursera.org/learn/flutter-developing-cross-platform-mobile-apps.

[8] F. i. A. Eric Windmill, »Flutter in Action,« u *Flutter in Action*, Manning; First Edition, 2019.

Popis Slika

Popis Tablica