Mobilna aplikacija za praćenje kartona pacijenata korištenjem NFC tehnologije

Forjan, Filip

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

Sveučilišni studij

Mobilna aplikacija za praćenje kartona pacijenata korištenjem NFC tehnologije

Diplomski rad

Filip Forjan

Osijek, 2023.

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

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 03.05.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Filip Forjan
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1118R, 13.10.2020.
OIB studenta:	50832823967
Mentor:	izv. prof. dr. sc. Ivica Lukić
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Mirko Köhler
Član Povjerenstva 1:	izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 2:	Miljenko Švarcmajer, mag. ing. comp.
Naslov diplomskog rada:	Mobilna aplikacija za praćenje kartona pacijenata korištenjem NFC tehnologije
Znanstvena grana diplomskog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	U okviru ovog dipkosmkog rada potrebno je izraditi mobilnu aplikaciju koristeći Flutter razvojni okvir i programski jeziki Dart koja će služiti za evidentiranje pacijenata pisanjem na NFC karticu i pohranjivati podatke u bazu podataka. Rezervirano za: Filip Forjan
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:	03.05.2023.

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, 15.05.2023.

Ime i prezime studenta:	Filip Forjan
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1118R, 13.10.2020.
Turnitin podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Mobilna aplikacija za praćenje kartona pacijenata korištenjem NFC tehnologije**

izrađen pod vodstvom mentora izv. prof. dr. sc. Ivica Lukić

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

U današnje vrijeme mobilne aplikacije postaju sve popularnije i sve više korištene u svakodnevnici. Porastom njihove potražnje na tržištu, ujedno raste i potražnja za programerima mobilnih aplikacija. Prije proizvodnje mobilne aplikacije potrebno je odlučiti kojom tehnologijom će se ona proizvesti, ovisno o njezinim zahtjevima. Pitanje koje se postavlja jest hoće li aplikacija biti razvijena u nativnim tehnologijama u ovisnosti o ciljanim platformama ili će biti proizvedena u sve češće korištenim višeplatformskim okvirima. Nativne tehnologije najčešće korištene za razvoj Android mobilnih aplikacija programski su jezici Java ili noviji Kotlin, dok se za iOS platformu koriste Objective C ili noviji Swift. Noviji pristup, odnosno korištenje višeplatformskih okvira, ubrzava proces proizvodnje, iz razloga što se jednim kodom aplikacije pokreću na više platformi, a samim time dolazi i do smanjenja troškova izrade aplikacije. Za izradu višeplatformskih aplikacija na tržištu postoji nekoliko popularnih izbora kao što su React Native, Ionic, Flutter i Xamarin. Budući da se teži digitalizaciji svakodnevnice, zadatak ovog diplomskog rada izrada je mobilne aplikacije, u nastavku Care Mate, za digitalno evidentiranje pacijenata. Za izradu aplikacije korišten je višeplatformski razvojni okvir Flutter i programski jezik Dart. Korištenjem ove aplikacije omogućeno je jednostavnije i brže spremanje i pristupanje pacijentovim podacima u stvarnom vremenu. Za pristup podacima, osim kroz sučelje mobilne aplikacije, koristi se i NFC tehnologija koja omogućava pristup pacijentovim podacima prislanjanjem mobilnog uređaja na NFC naljepnicu.

1.1. Zadatak završnog rada

U okviru ovog diplomskog rada bilo je potrebno izraditi mobilnu aplikaciju koristeći Flutter razvojni okvir i programski jezik Dart, koja će služiti za evidentiranje pacijenata pisanjem na NFC karticu te pohranjivati podatke u bazu podataka.

2. PREGLED PODRUČJA TEME

Mobilna aplikacija iz ovog diplomskog rada usporedit će se u nastavku s postojećim i sličnim aplikacijama na tržištu. Iako trenutno ne postoji mobilna aplikacija koja implementira NFC tehnologiju te evidenciju i upravljanje podacima pacijenata, ipak postoje aplikacije s razdvojenim funkcionalnostima.

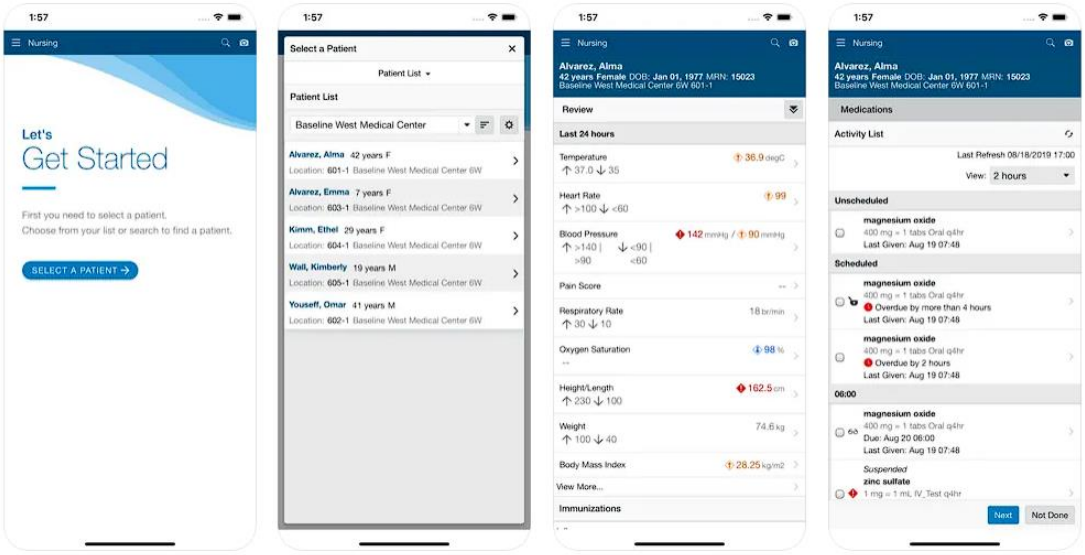
2.1. Cerner Corporation

Cerner Corporation američki je dobavljač usluga, uređaja i hardvera za zdravstvenu informacijsku tehnologiju (HIT) [1]. Posjeduju nekoliko objavljenih aplikacija vezanih za zdravstvo, a u nastavku su navedeni primjeri dviju aplikacija koje imaju sličnu primjenu kao aplikacija napravljena za ovaj diplomski rad.

Slika 2.1. Cerner corporation logo [2]

2.1.1. Connect Nursing

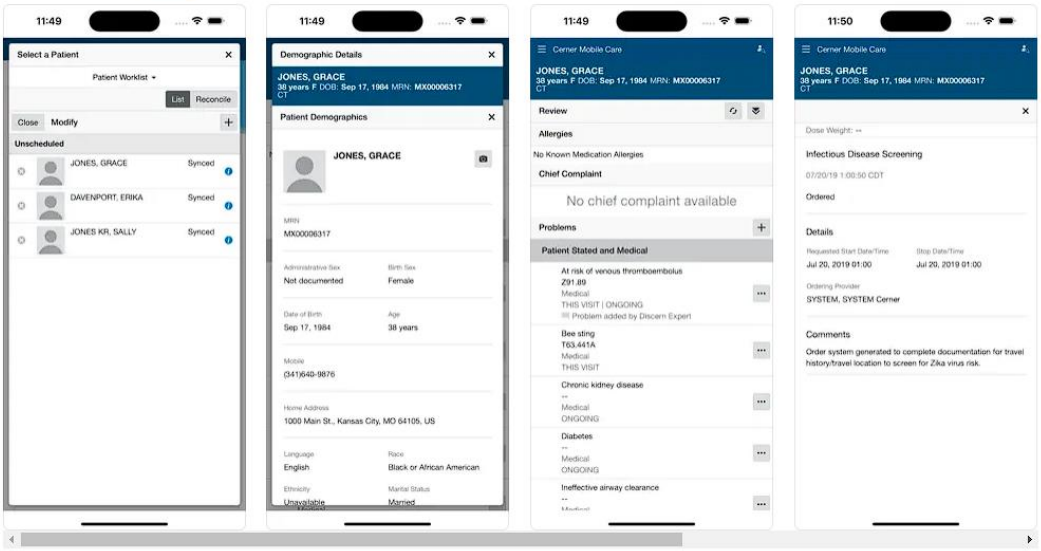
Connect Nursing je mobilna aplikacija pomoću koje zdravstveni radnici imaju brzi pristup informacijama koje su dodijeljene pacijentima. Aplikacija omogućava praćenje pacijentovih podataka, njihovih stanja i ažuriranje terapije. Moguć je unos pacijentove terapije, a terapiju je moguće dodati skeniranjem bar koda.



Slika 2.2. Connect Nursing aplikacija [3]

2.1.2. Cerner Mobile Care

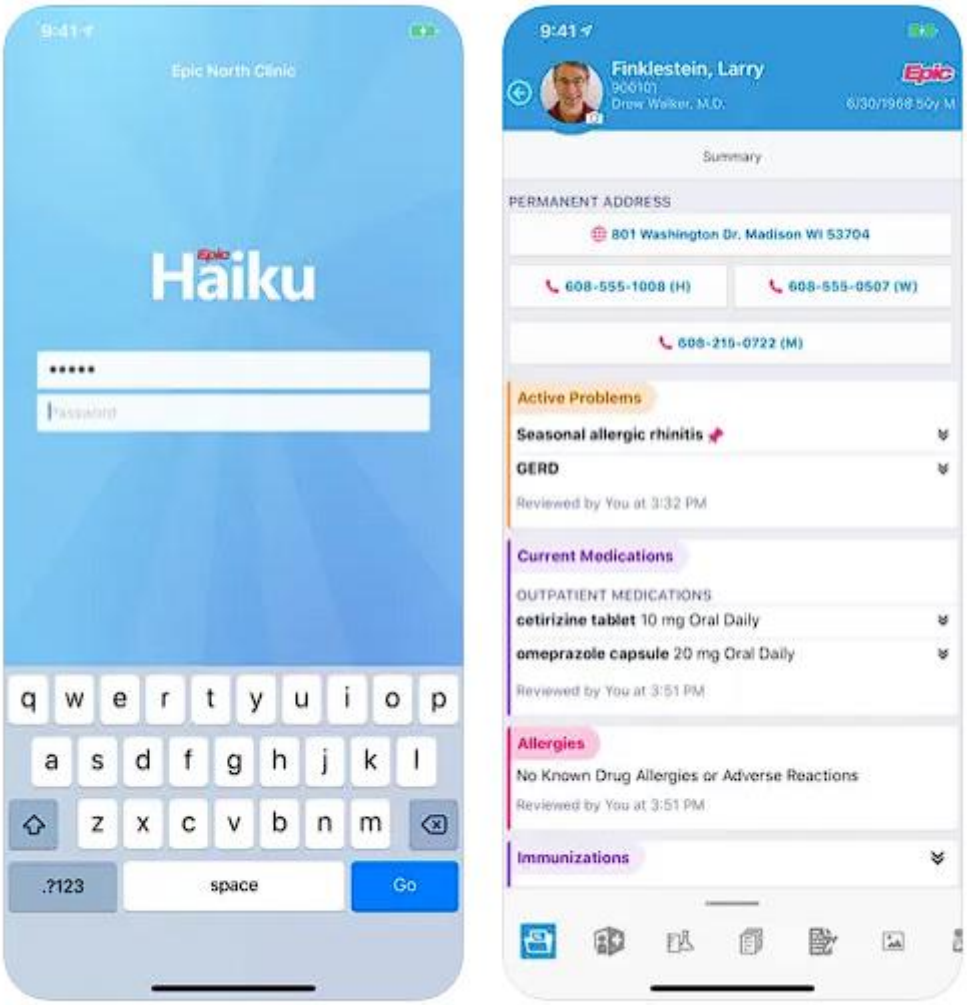
Cerner Mobile Care je mobilna aplikacija pomoću koje zdravstveni radnici bez pristupa internetu mogu pristupiti podacima pacijenata. Podatke je moguće preuzeti izvan prostorija zdravstvene ustanove što je ujedno i glavna primjena spomenute aplikacije.



Slika 2.3. Cerner Mobile Care aplikacija [4]

2.2. Epic Haiku & Limerick

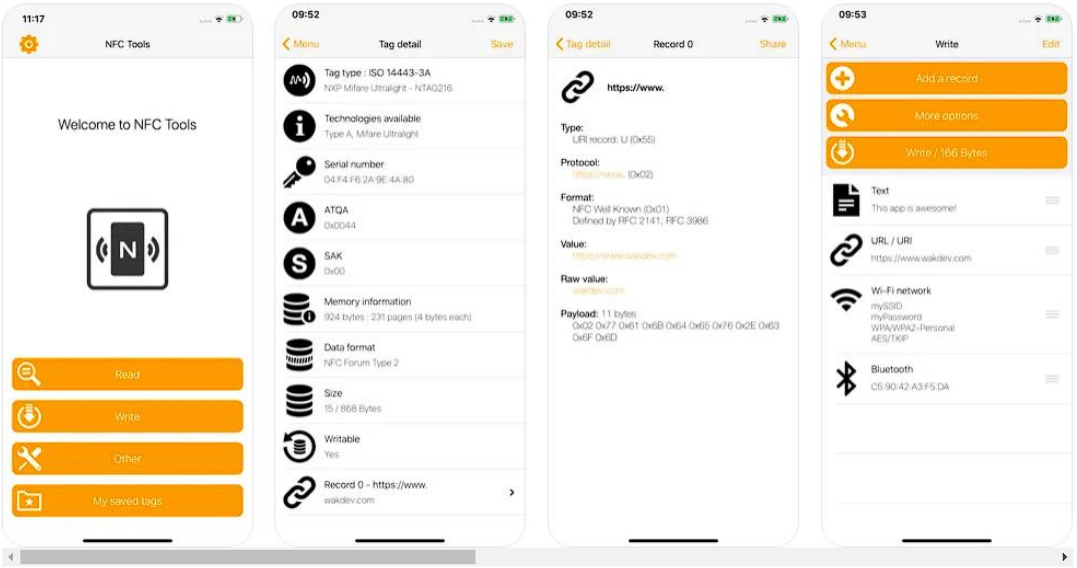
Epic Haiku je mobilna aplikacija koja omogućuje siguran pristup podacima kao što su kalendarski raspored klinike, podaci pacijenata bolnice, rezultati testiranih uzoraka i slično. Za korištenje aplikacije svaka organizacija mora imati licencu.



Slika 2.4. Epic Haiku & Limerick aplikacija [5]

2.3. NFC Tools

NFC Tools je mobilna aplikacija s jednostavnim sučeljem korištena za čitanje i pisanje na NFC naljepnicama. Komunikaciju između mobilnog uređaja i NFC naljepnice moguće je ostvariti ako mobilni uređaj ima podršku za NFC, a komunikacija se ostvaruje prislanjanjem mobilnog uređaja na naljepnicu [6].



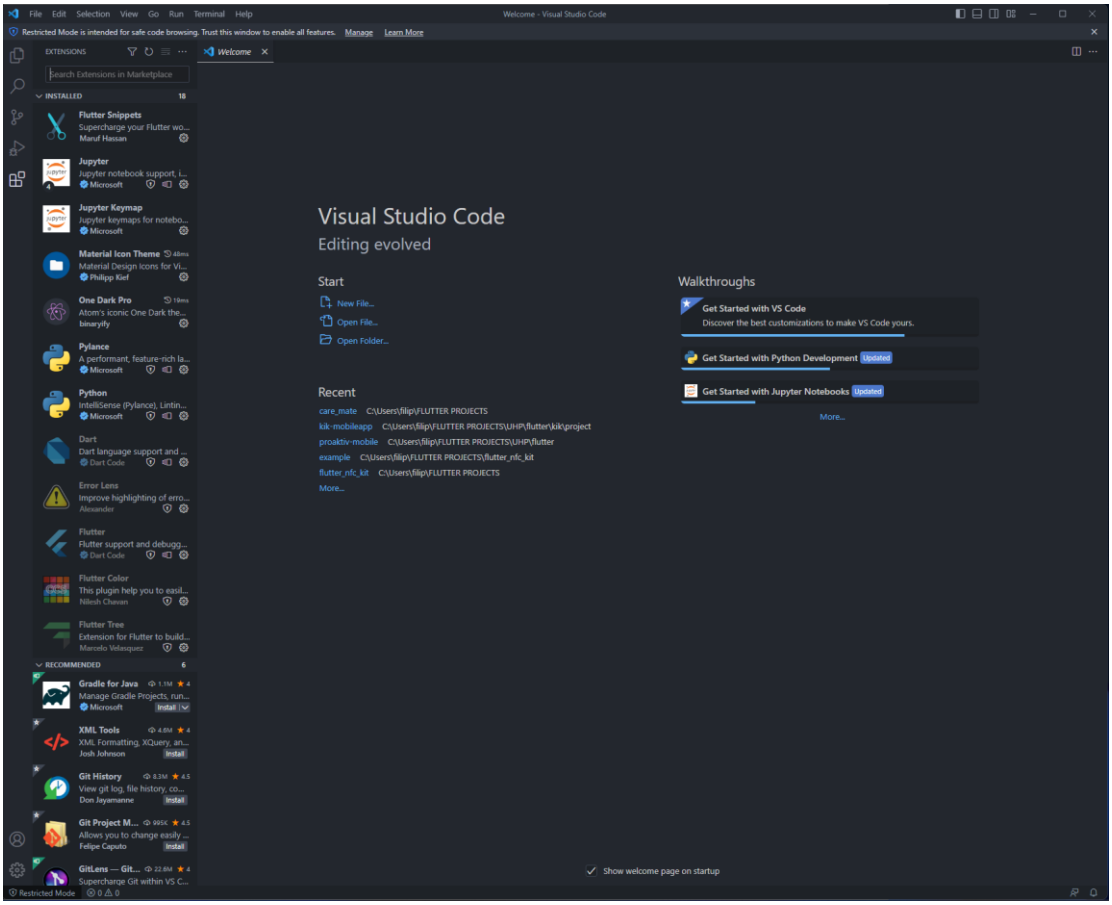
Slika 2.5. NFC Tools aplikacija

3. OPIS KORIŠTENIH TEHNOLOGIJA

Ovo poglavlje detaljnije opisuje tehnologije korištene za razvoj Care Mate mobilne aplikacije. Aplikacija je napisana u Flutter razvojnom okviru koristeći Dart programski jezik te unutar Visual Studio Code okruženja. Za pohranu podataka korišten je Google Firebase servis, dok se NFC naljepnice koriste za brz i jednostavan prikaz podataka pacijenata.

3.1. Visual Studio Code

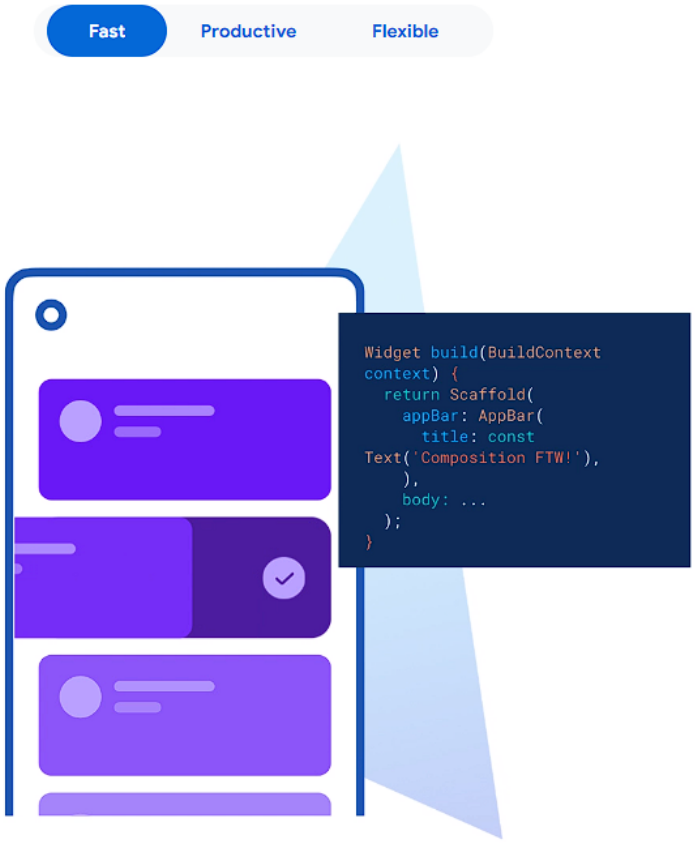
Visual Studio Code jedan je od najčešće korištenih programskih okruženja (engl. *Integrated development environment)* prilikom razvoja software-a. Za razliku od ostalih programskih okruženja poput Android Studija ili IntelliJ IDEA, Visual Studio Code je brz, lagan i kompaktan što znači da za razliku od ostalih okruženja koristi manje računalnih resursa. Dolazi s ugrađenom podrškom za Git sustav verzioniranja. Instaliranjem proširenja moguće je dodati podršku za nove jezike, teme, programe za ispravljanje grešaka i spajanje na druge servise [7]. Korisnici ga mogu uređivati i prilagođavati prema vlastitim željama.



Slika 3.1. Visual Studio Code sučelje

3.2. Flutter

Flutter je razvojni okvir otvorenog koda tvrtke Google za izradu lijepih, nativno kompajliranih, višeplatformskih aplikacija iz jednog koda [8]. Flutter je sve popularniji izbor prilikom odabira razvojnog okruženja za proizvodnju mobilnih aplikacija. Proizvela ga je tvrtka Google 2017. godine, a prva stabilna verzija objavljena je početkom 2018. godine. Flutter olakšava proizvodnju mobilnih aplikacija na jednostavan i poznat način. Ono što je posebno kod Fluttera jest da omogućuje proizvodnju aplikacija za više platformi uz "piši jednom, pokreni bilo gdje" pristup [9, str. 3]. Jednim kodom programer proizvodi internetsku stranicu i mobilnu aplikaciju koju je moguće pokrenuti na iOS i Android uređajima.



Slika 3.2. Flutter razvojno okruženje

3.2.1. Osnovni elementi

Korisnici Flutter-a vole reći kako je sve takozvani element (engl. *Widget*). Za izradu mobilnih aplikacija koriste se Google-ovi prethodno implementirani elementi, a njihovim kombinacijama i

spajanjem, stvaraju se novi prilagođeni elementi. Taj se proces nastavlja do samog kraja razvijanja mobilne aplikacije [10, str. 33]. U Flutter-u postoje elementi sa stanjem (engl. *Stateful*) i bez stanja (engl. *Stateless*). Glavna razlika je u tome što se elementi bez stanja ne mogu naknadno promijeniti tijekom izvođenja aplikacije. Takvi elementi korišteni su za prikazivanje statičnih sadržaja. Elementi sa stanjem mogu se mijenjati tijekom izvođenja aplikacije i korišteni su za pohranjivanje podataka te interakciju s korisnikom aplikacije.

```dart
class MyStatelessWidget extends StatelessWidget {
  const MyStatelessWidget({super.key});

  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

```dart
class MyStatefulWidget extends StatefulWidget {
  const MyStatefulWidget({super.key});

  @override
  State<MyStatefulWidget> createState() => _MyStatefulWidgetState();
}

class _MyStatefulWidgetState extends State<MyStatefulWidget> {
  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

Slika 3.1. Primjer klase bez stanja i klase sa stanjem

3.2.2. Upravljanje stanjem

Upravljanje stanjem aplikacije vrlo je važan korak pri izradi mobilne aplikacije, a načini odnosno odabir kako rukovati stanjem aplikacije svode se na osobne preferencije programera. Trenutno na tržištu ima nekoliko paketa za upravljanje stanjem aplikacije, a neki od poznatijih su Bloc, Provider i Riverpod.

Riverpod je biblioteka izrađena od strane programera koji je ujedno izradio i Provider biblioteku za upravljanje stanjem aplikacije. Za razliku od Provider-a, Riverpod ima dodatne mogućnosti poput automatskog čišćenja korištenih resursa ukoliko se više ne koristi. Vrlo je jednostavan za korištenje i olakšava testiranje izoliranjem dijelova koda. S Riverpod-om, stanju

se može pristupiti i ono se može promijeniti na bilo kojem mjestu u kodu, što olakšava dosljednost koda. [12]

Create a Provider

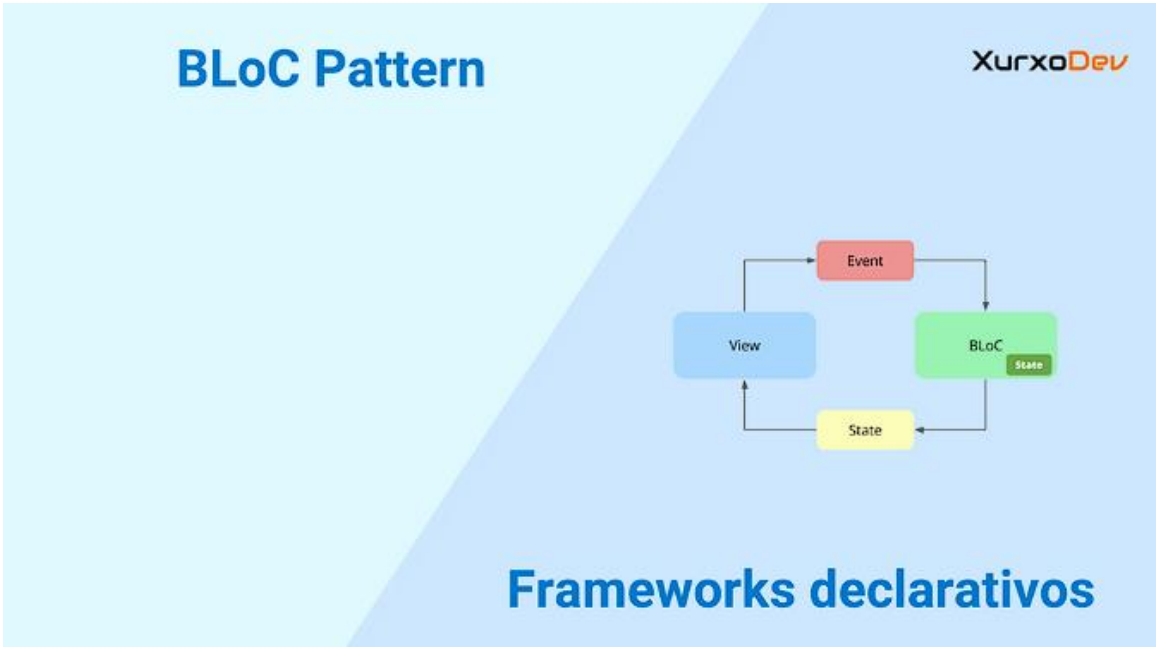
```dart
final counterProvider = StateNotifierProvider<Counter, int>((ref) {
  return Counter();
});

class Counter extends StateNotifier<int> {
  Counter() : super(0);
  void increment() => state++;
}
```

Consume the Provider

```dart
class Home extends ConsumerWidget {
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final count = ref.watch(counterProvider);
    return Text('$count');
  }
}
```

Slika 3.3. Primjer Riverpod-a [13]

Bloc je biblioteka koja u potpunosti razdvaja poslovnu logiku aplikacije od njezinog korisničkog sučelja. Sastoji se od tri elementa: stanja, događaja i upravljača stanjima. Stanje definira stanje aplikacije, događaji definiraju moguće događaje koji će se dogoditi interakcijom korisnika, a unutar upravljača se definira i upravlja logikom korištenjem prethodno definiranih stanja i događaja. Za jednostavne i male projekte Bloc biblioteka može biti nepotrebna zbog svoje veličine.



Slika 3.4. Bloc elementi [14]

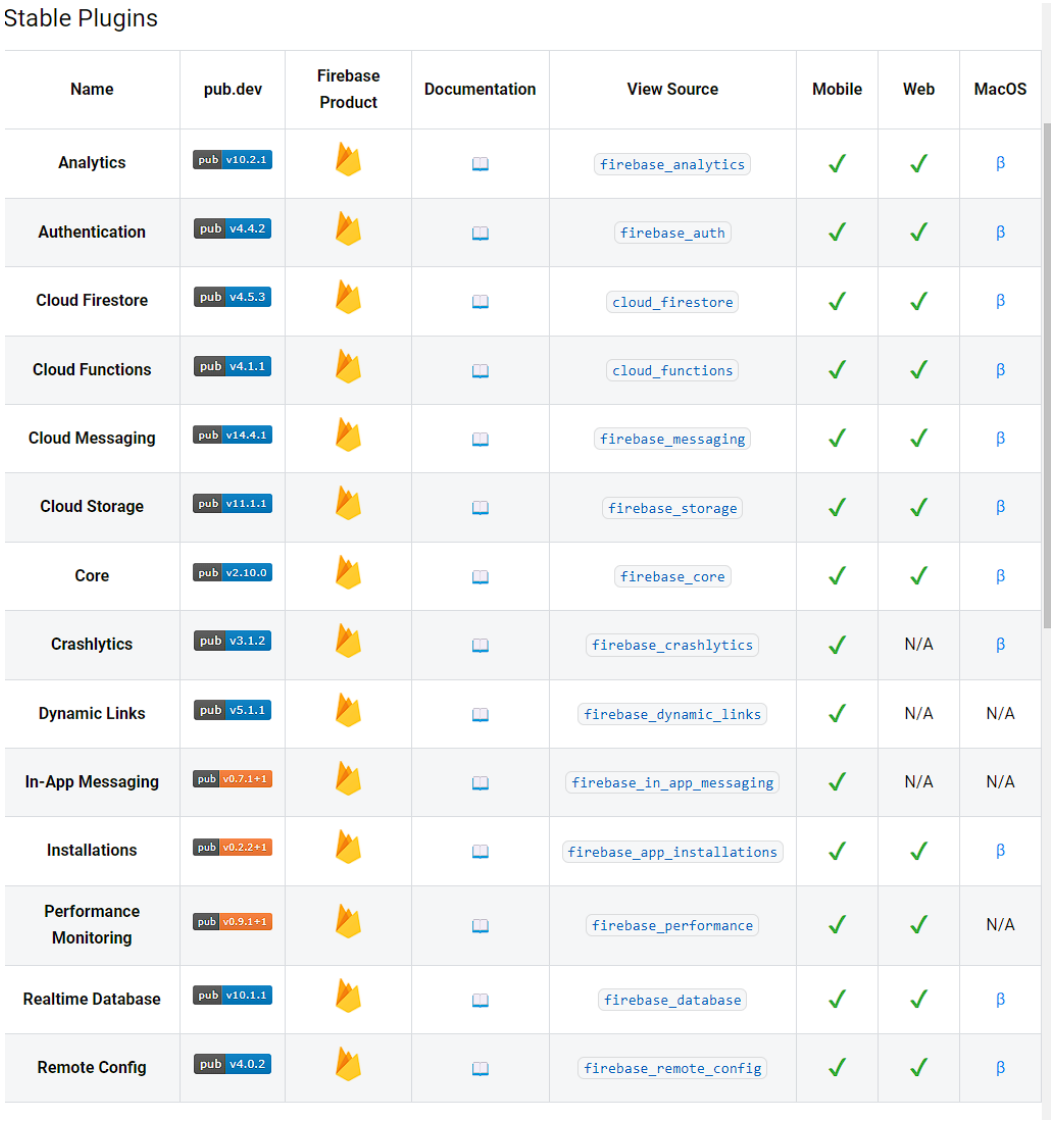
3.3. Dart

Dart je objektno orijentirani jezik razvijen od strane Google-a. Korištenjem Dart programskog jezika moguće je razvijati mobilne i web aplikacije, a također ima mogućnost izgradnje serverskih i zaslonskih (engl. *Desktop*) aplikacija [15]. Glavna obilježja Dart-a su JIT (engl. *Just in time*) kompilacija koja omogućuje brzo ponovno pokretanje (engl. *Hot reload*), AOT (engl. *Ahead of time*) kompilacija za kompajliranje koda na nativne platforme, asinkroni kod i null sigurnost [16]. Postoje još brojne značajke koje Dart programski jezik čine vrlo jednostavnim, brzim i pouzdanim programskim jezikom (Slika 3.5).

Slika 3.5. Karakteristike Dart programskog jezika

3.4. Firebase

Firebase je platforma korištena za razvoj i skaliranje aplikacija i igara. Razvijena je od strane Google-a te ga veliki broj korisnika koristi kao pozadinski servis (engl. *Backend*) svojih aplikacija [17]. Omogućava pristup velikom broju priključaka (engl. *Plugin*) koje je moguće implementirati u aplikacije. Za Flutter razvojni okvir trenutno postoji desetak stabilnih priključaka kao što je prikazano na slici 3.7.



Stable Plugins

Name	pub.dev	Firebase Product	Documentation	View Source	Mobile	Web	MacOS
Analytics	pub v10.2.1			firebase_analytics	✓	✓	β
Authentication	pub v4.4.2			firebase_auth	✓	✓	β
Cloud Firestore	pub v4.5.3			cloud_firestore	✓	✓	β
Cloud Functions	pub v4.1.1			cloud_functions	✓	✓	β
Cloud Messaging	pub v14.4.1			firebase_messaging	✓	✓	β
Cloud Storage	pub v11.1.1			firebase_storage	✓	✓	β
Core	pub v2.10.0			firebase_core	✓	✓	β
Crashlytics	pub v3.1.2			firebase_crashlytics	✓	N/A	β
Dynamic Links	pub v5.1.1			firebase_dynamic_links	✓	N/A	N/A
In-App Messaging	pub v0.7.1+1			firebase_in_app_messaging	✓	N/A	N/A
Installations	pub v0.2.2+1			firebase_app_installations	✓	✓	β
Performance Monitoring	pub v0.9.1+1			firebase_performance	✓	✓	N/A
Realtime Database	pub v10.1.1			firebase_database	✓	✓	β
Remote Config	pub v4.0.2			firebase_remote_config	✓	✓	β

Slika 3.6. Firebase stabilni priključci za Flutter

3.4.1. Firebase provjera autentičnosti

Firebase provjera autentičnosti (engl. *Authentication*) korištena je kako bi bilo moguće identificirati korisnika aplikacije. Provjerom autentičnosti korisnika omogućava se sigurno spremanje korisnikovih podataka u bazu koja se nalazi u oblaku (engl. *Cloud*). Postoje brojne značajke koje se brinu za korisnikov identitet poput provjere više-faktorske autentičnosti (eng. Multi-factor authentication), blokiranje funkcija i bilježenja korisničkih aktivnosti [18].

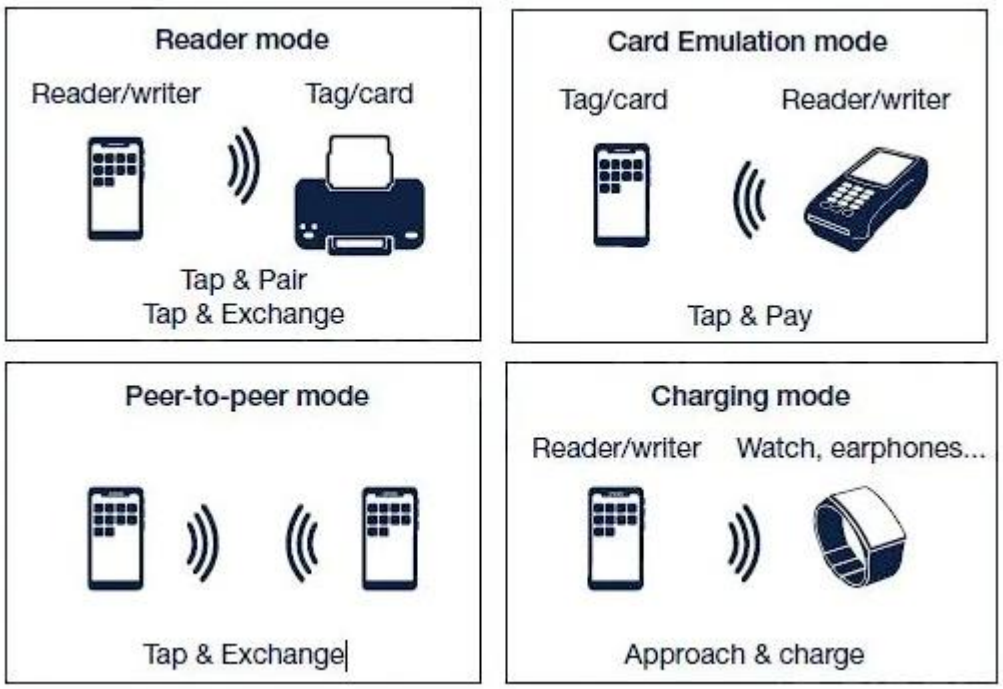
3.4.2. Cloud Firestore

Cloud Firestore je noviji oblik pohane podataka. Proizveden je od strane Firebase-a, a utemeljen je na NoSQL modelu. Koncept pohrane podataka svodi se na kolekcije (engl.

Collection) i dokumente (engl. *Document*). Kolekcija je skupina dokumenata, a dokument je osnovni model nalik objektu koji sadrži pohranjena polja (engl. *Field*) koja predstavljaju atribute tog objekta. Firestore pruža pristup sučelju na kojemu je moguće pristupiti podacima ili nad njima postavljati upite, a također omogućuje postavljanje upita iz vanjskih aplikacija. Održava podatke sinkroniziranima u svim povezanim aplikacijama u stvarnome vremenu, a isto tako podržava i spremanje podataka bez pristupa internetu [19].

3.5. NFC

NFC (engl. *Near-field communication*) funkcionira na principu bežične tehnologije povezivanja koja koristi magnetsko indukcijsko polje. Kratkog je dometa, a komunikacija se uspostavlja dodirivanjem dva uređaja ili približavanjem na nekoliko centimetara jedan od drugoga. [20]. Neke od najčešćih primjena NFC tehnologije su prijenos male količine podataka, sigurnosni pristup poput otključavanje vrata ili pristup povjerljivim podacima te mobilno plaćanje (Slika 3.7). NFC standardi pokrivaju komunikacijske protokole i formate razmjene podataka, a temelje se na postojećim RFID standardima [21]. Najčešće korišteni format za izmjenu podataka je NDEF (engl. *NFC Data Exchange Format*).



Slika 3.7. Primjeri komunikacije pomoću NFC tehnologije

4. RAZVOJ APLIKACIJE

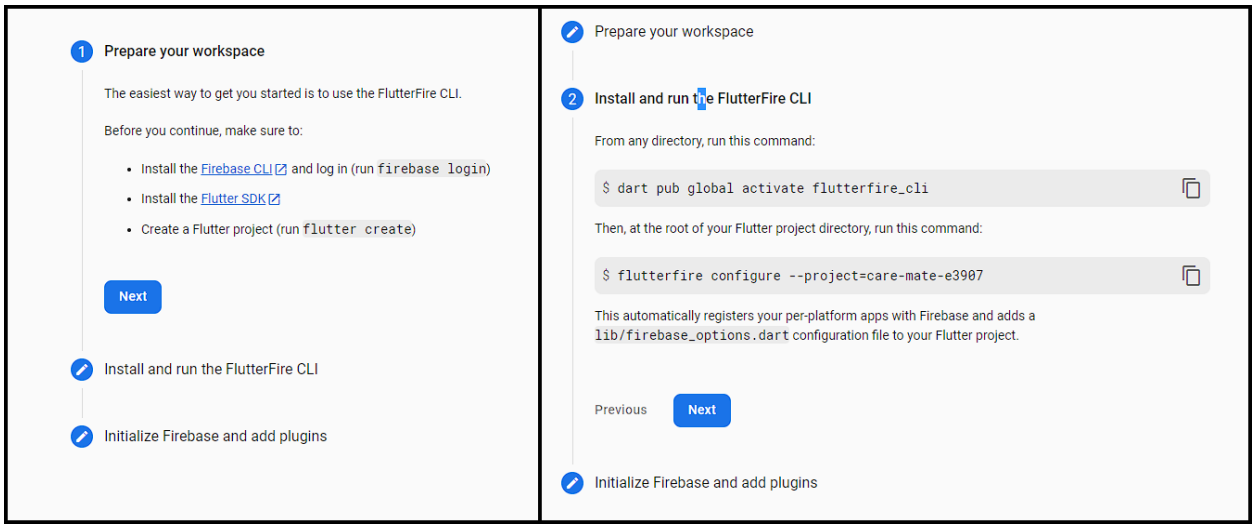
U ovome poglavlju ukratko je opisan tijek aplikacije te detaljno objašnjen proces izrade čitave aplikacije. Objašnjena je implementacija provjere autentičnosti i baze podataka te je objašnjena arhitektura aplikacije, upravljanje stanjem aplikacije i korisničko sučelje.

4.1. Koncept aplikacije

Aplikacija je osmišljena s namjerom olakšanja evidencije osnovnih podataka pacijenata medicinskom osoblju. Umjesto trenutnog i tradicionalnog načina upisivanja mjerenja temperature i krvnog tlaka na papir te prepisivanje u računalni sustav, korištenjem ove aplikacije medicinsko osoblje na jednostavan način prislanjanjem mobilnog uređaja na NFC naljepnicu, koja se nalazi na krevetu namijenjenom pacijentima, pristupaju zaslonu koji prikazuje pacijentove osobne podatke i povijest mjerenja temperature i krvnog tlaka.

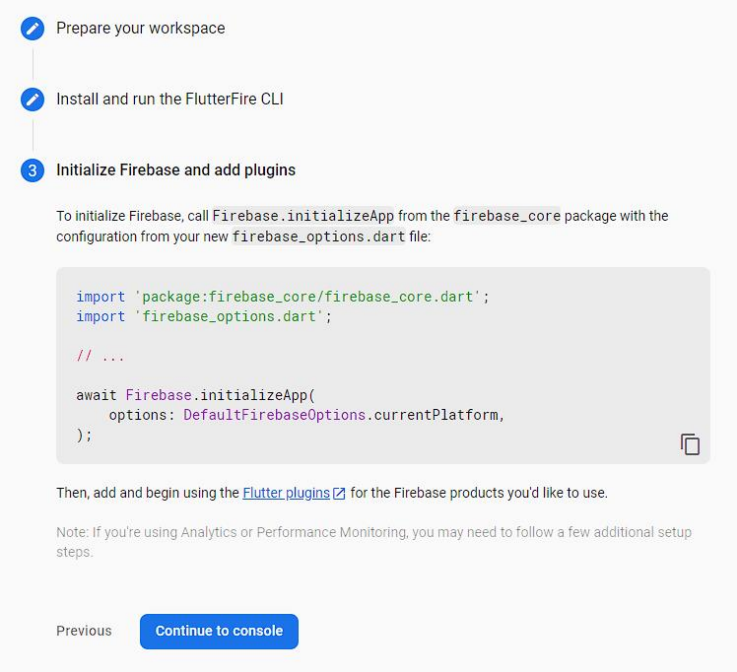
4.2. Implementacija Firebase servisa

Unutar ovog poglavlja objašnjena je implementacija Firebase alata za provjeru autentičnosti korisnika i implementacija Firestore baze podataka. Prije korištenja Firebase-ovih alata potrebno je dodati Firebase u svoj projekt. Prvi je korak instalacija i pokretanje FlutterFire CLI (engl. *Command-line interface*) [22] (Slika 4.1).



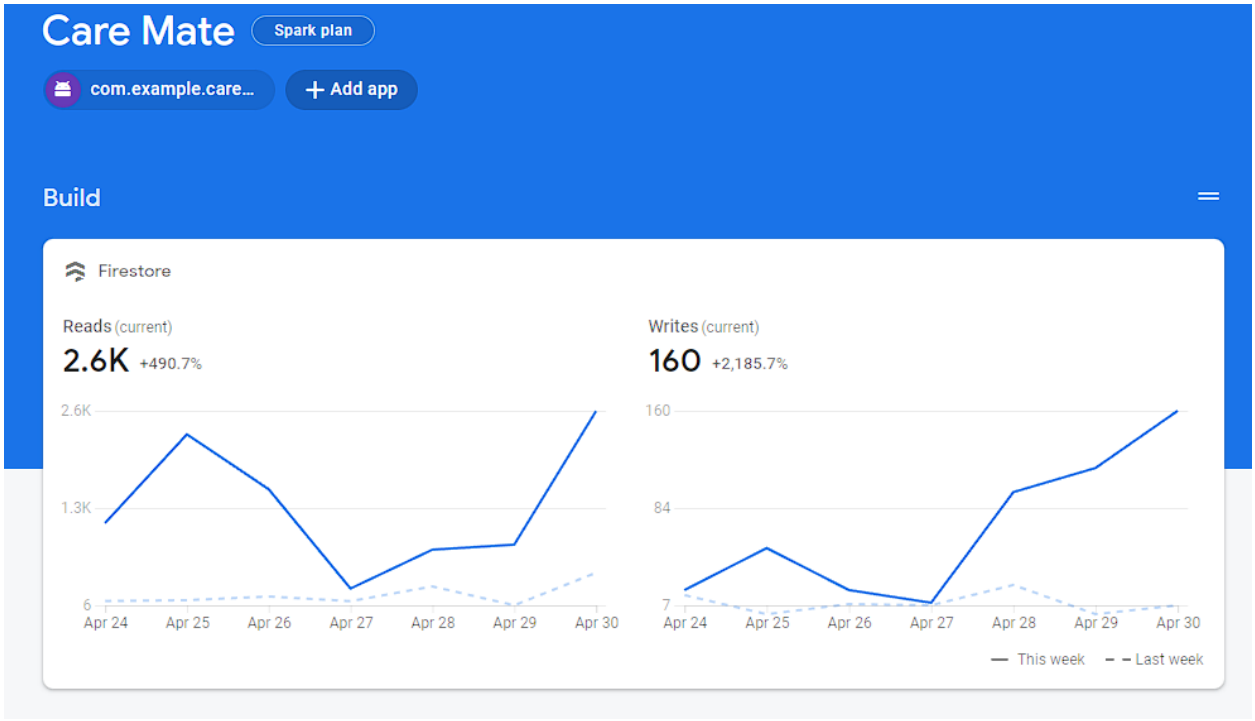
Slika 4.1. Instalacija i pokretanje FlutterFire CLI

Sljedeći je korak inicijalizacija Firebase paketa u aplikaciji. Za to je potrebno dodati željene pakete i pozvati instancu Firebase-a u main.dart datoteci. Proces je prikazan na slici 4.2.



Slika 4.2. Inicijalizacija Firebase paketa

Izvršavanjem prethodna tri koraka, Firebase paket inicijaliziran je u aplikaciji i spreman za korištenje. Na početnoj stranici Firebase-a prikazan je trenutni promet čitanja i pisanja u Firebase bazi podataka u posljednja 24 sata (Slika 4.3).



Slika 4.3. Firebase promet aplikacije

4.2.1. Provjera autentičnosti

Provjera autentičnosti implementirana je na vrlo jednostavan način putem Firebase-ovog paketa Firebase Auth. Unutar funkcije za prijavu korisnika potrebno je pozvati instancu Firebase-ali i njegovu funkciju koja prima e-mail i zaporku. Ako su podaci ispravno uneseni, funkcija nam vraća korisnikove vjerodajnice (engl. *Credentials*). Odjava korisnika izvršava se pozivom Firebase-ove funkcije signOut(). Primjer koda prikazan je na slici 4.4.

```dart
import 'package:firebase_auth/firebase_auth.dart';

class AuthService {
  final FirebaseAuth _firebaseAuth;

  AuthService(this._firebaseAuth);

  Future<String?> signInWithEmailAndPassword({
    required String email,
    required String password,
  }) async {
    try {
      UserCredential userCredential =
          await _firebaseAuth.signInWithEmailAndPassword(
        email: email,
        password: password,
      );
      return userCredential.user!.uid;
    } on FirebaseAuthException catch (e) {
      throw e.message!;
    } catch (e) {
      rethrow;
    }
  }

  User? getCurrentUser() {
    return _firebaseAuth.currentUser;
  }

  Future<void> singOut() async {
    await _firebaseAuth.signOut();
  }
}
```

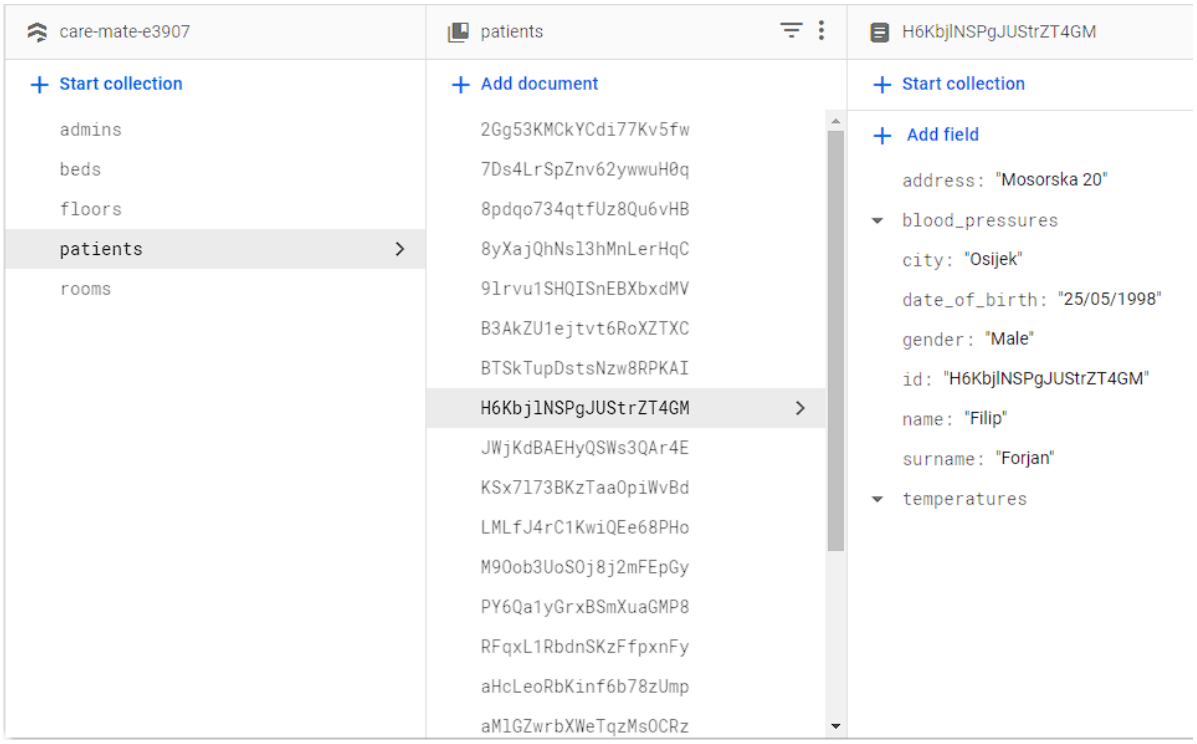
```dart
Future<String?> signInWithEmailAndPassword() async {
  state = state.copyWith(appState: AppState.loading);
  String? res;
  try {
    res = await ref.read(authRepositoryProvider).signInWithEmailAndPassword(
          email: state.email,
          password: state.password,
        );
  } catch (e) {
    state = state.copyWith(appState: AppState.error, error: e.toString());
  }
  state = state.copyWith(appState: AppState.success);
  return res;
}

User? getCurrentUser() {
  return ref.read(authRepositoryProvider).getCurrentUser();
}

Future<void> signOut() async {
  state = state.copyWith(appState: AppState.loading);
  try {
    await ref.read(authRepositoryProvider).signOut();
  } catch (e) {
    state = state.copyWith(appState: AppState.error, error: e.toString());
  }
  state = state.copyWith(appState: AppState.success);
}
}
```

Slika 4.4. Kod provjere autentičnosti

4.2.2. Firestore baza podatak

Baza podataka sastoji se od pet Firebase-ovih kolekcija. Za provjeru autentičnosti prijavljenog korisnika u aplikaciji putem Firebase-a, koristi se kolekcija *admins* kako bi se utvrdilo jesu li mu dodijeljena ovlaštena prava. Ako se korisnikov *id* nalazi u *admins* kolekciji, njemu će biti omogućene dodatne funkcionalnosti poput dodavanja novih katova, soba i kreveta ili spajanja prazne NFC kartice na krevet. Kolekcija *floors* sadržana je od dokumenata koji predstavljaju katove u bolnici, a kao polja odnosno atribute ima *name* i *id.* Kolekcija *rooms* istog je sadržaja kao kolekcija *floors*, ali ima dodatni parametar *floor_id*, odnosno identifikacijsku oznaku kata na koji je vezana. Kolekcije *patients* i *beds* sadrže dokumente s većim brojem parametara. Dokument *patient* sastoji se od polja *address*, *blood_pressures* što je lista mjerenja krvnih tlakova, *city*, *date_of_birth*, *gender*, id, *name*, *surname* i *temperatures* što je lista mjerenja Tjelesnih

temperatura. Dokument *bed* sadrži polja *id, name, nfc_id, patient_id, room_id.* Primjer strukturirane Firebase baze prikazan je na slici 4.5.



Slika 4.5. Struktura Firebase firestore baze podataka

Mobilna aplikacija Care Mate šalje dvije vrste upita na bazu podataka. Prva vrsta je uobičajeni upit za dohvaćanje podatka i primjeri koda takvog upita se mogu vidjeti na slici 4.6. Za dohvaćanje kreveta iz kolekcije *beds* potrebno je postaviti upit na željenu kolekciju (što je u ovom primjeru *beds*), postaviti uvjet kao u ovom primjeru *where* koji pretražuje sve dokumente u kolekciji po polju *nfc_id* i vraća samo jedan rezultat, gdje je traženo polje jednako funkciji predanom parametru *nfcID.* Dohvaćeni rezultat potrebno je pretvoriti iz JSON formata u objekt tipa *Bed.*

```
Future<Bed?> getBedByNfcId({required String nfcId}) async {
  try {
    final bedSnapshot = await FirebaseFirestore.instance
        .collection('beds')
        .where('nfc_id', isEqualTo: nfcId)
        .limit(1)
        .get();

    if (bedSnapshot.size == 0) {
      return null;
    }
    final bed = Bed.fromJson(bedSnapshot.docs.first.data());
    return bed;
  } catch (e) {
    rethrow;
  }
}

Future<Patient?> getPatientByBed({required String patientId}) async {
  try {
    final patientSnapshot = await FirebaseFirestore.instance
        .collection('patients')
        .where('id', isEqualTo: patientId)
        .limit(1)
        .get();

    if (patientSnapshot.size == 0) {
      return null;
    }
    final patient = Patient.fromJson(patientSnapshot.docs.first.data());
    return patient;
  } catch (e) {
    rethrow;
  }
}
}
```

Slika 4.6. Upit za dohvaćanje podataka iz Firestore baze

Drugi primjer upita na Firestore bazu podataka je *Stream*. *Stream* omogućuje neprestano slušanje na bazu podataka te za svaku promjenu na bazi obavještava pretplaćene korisnike. Informacije se na zaslonu prikazuju u stvarnom vremenu. Na slici 4.7. dani su primjeri koda koji vraćaju *stream*. Care Mate aplikacija osluškuje bazu pomoću *stream-a* kako bi u stvarnom vremenu prikazale promjene na korisnicima, katovima, sobama i krevetima. Dohvaća se referenca željene kolekcije te se na nju postavljaju upiti sa željenim uvjetima. Na danom primjeru na kolekciju *patients*, postavljen je upit s uvjetom da se korisnici sortiraju po prezimenu. Dohvaćene pacijente potrebno je pretvoriti iz Json oblika u objekt tipa *Patient* i na kraju vratiti listu dohvaćenih pacijenata.

```dart
Stream<List<Patient>> streamPatients({required String searchInput}) {
  try {
    final CollectionReference patientsCollectionRef =
        _firebaseFireStore.collection('patients');
    return patientsCollectionRef.orderBy("surname").snapshots().map((event) =>
        event.docs
            .map((e) => Patient.fromJson(e.data() as Map<String, dynamic>))
            .toList());
  } catch (e) {
    rethrow;
  }
}

Stream<List<Floor>> streamFloors() {
  try {
    final CollectionReference floorsCollectionRef =
        _firebaseFireStore.collection('floors');
    return floorsCollectionRef.orderBy('name').snapshots().map((event) =>
        event.docs
            .map((e) => Floor.fromJson(e.data() as Map<String, dynamic>))
            .toList());
  } catch (e) {
    rethrow;
  }
}

Stream<List<Room>> streamRooms({required Floor floor}) {
  try {
    final CollectionReference roomsCollectionRef =
        _firebaseFireStore.collection('rooms');
    return roomsCollectionRef
        .where('floor_id', isEqualTo: floor.id)
        .orderBy('name')
        .snapshots()
        .map((event) => event.docs
            .map((e) => Room.fromJson(e.data() as Map<String, dynamic>))
            .toList());
  } catch (e) {
    rethrow;
  }
}

Stream<List<Bed>> streamBeds({required Room room}) {
  try {
    final CollectionReference roomsCollectionRef =
        _firebaseFireStore.collection('beds');
    return roomsCollectionRef
        .where('room_id', isEqualTo: room.id)
        .orderBy('name')
        .snapshots()
        .map((event) => event.docs
            .map((e) => Bed.fromJson(e.data() as Map<String, dynamic>))
            .toList());
  } catch (e) {
    rethrow;
  }
}

```

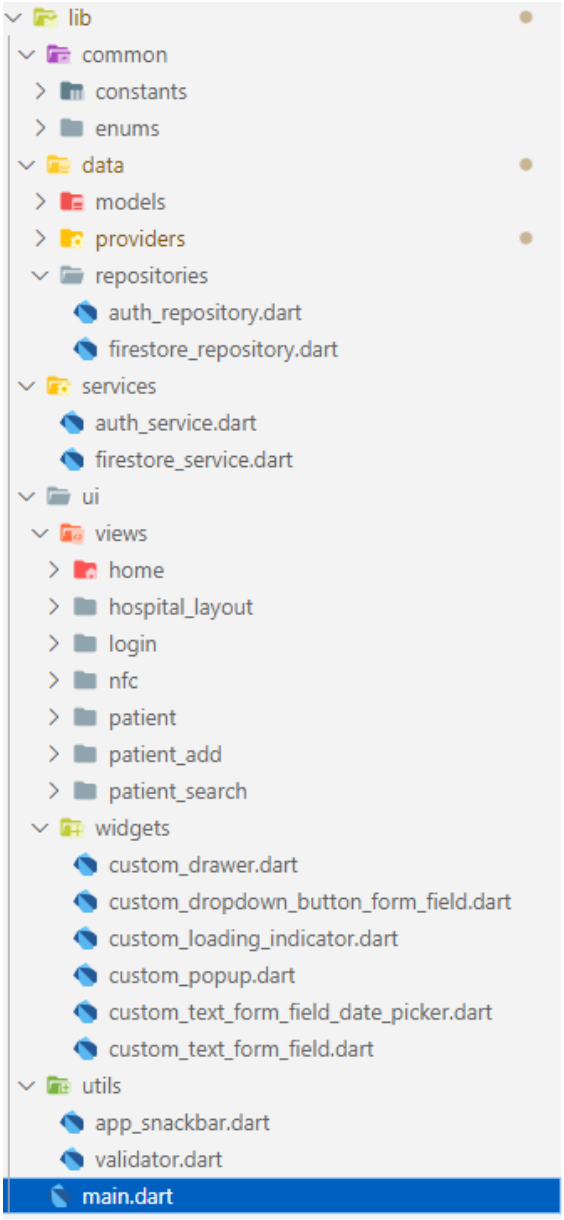
Slika 4.7. Upit za stream na Firestore bazu podataka

4.3. Implementacija aplikacije

Unutar ovog poglavlja detaljnije je objašnjena arhitektura aplikacije, upravljanje stanjem aplikacije i korisničko sučelje.

4.3.1. Arhitektura

Arhitektura je vrlo bitna prilikom izrade mobilnih aplikacija. Kako bi aplikacije bile što čitljivije i kako bi ih bilo jednostavnije proširivati i mijenjati, bitno je dobro rasporediti strukturu. Razdvajanje poslovne logike, stanja i korisničkog sučelja gotovo je nužno kako bi snalaženje po kodu aplikacije bilo moguće. Struktura Care Mate aplikacije prikazana je na slici 4.8.



Slika 4.8. Arhitektura Care Mate aplikacije

Mapa *common* sadrži konstante i podatke tipa *Enum* koji se koriste za enumeraciju stanja aplikacije. Svi modeli stanja i objekata aplikacije, *provideri* korišteni za upravljanje stanjima aplikacije i repozitoriji nalaze se u mapi *data*. *UI* mapa razdvojena je na mapu *widgets* koja sadrži

sve elemente koji se ponavljaju kroz aplikaciju i koje je potrebno koristiti više puta te na mapu *views* koja sadrži sve zaslone i elemente koji se koriste na samo jednom zaslonu. Posljednja mapa *utils* koristi se za pohranjivanje privremenih tipova podataka, što je u ovom primjeru datoteka za prikaz poruka na dnu zaslona, ili *validator* koji služi za provjeru ispravnosti napisanog e-mail.

4.3.2. Upravljanje stanjem aplikacije

Upravljanje stanjem aplikacije Care Mate implementirano je koristeći Riverpod. Primjer koda prikazan je na slici 4.9. Klasa *NfcNotifer* posjeduje svoje stanje tipa *NfcState*. Unutar stanja pohranjene su informacije koje je potrebno pratiti i prikazivati na zaslonu ovisno o njihovim promjenama. Zaslon za rukovanje NFC naljepnicama sadrži stanje o identifikacijskoj oznaci NFC naljepnice, krevet na koji će se NFC naljepnica povezivati, pacijent povezan s krevetom i stanje potrebno za upravljanje zaslonom što su tekst greške (engl. *Error*) i *enum* tipa AppState koji govori u kojem stanju se aplikacija trenutno nalazi. Moguća stanja aplikacije su učitavanje (engl. *Loading*), početno stanje (engl. *Initial*), greška i uspjeh (engl. *Success*). Na primjeru dohvaćanja pacijenta pomoću kreveta, može se objasniti upravljanje stanjem aplikacije. Na početku izvođenja funkcije, stanje aplikacije stavlja se u učitavanje. Zatim se pokušava dohvatiti pacijent pozivanjem funkcije deklarirane u repozitoriju koji zove instancu prethodno objašnjenog servisa za upravljanje Firestore bazom podataka. Funkcija prima krevet kao parametar, a pomoću kreveta u bazi podataka pretražuje se postoji li pacijent čija identifikacijska oznaka odgovara identifikacijskog oznaci pacijenta pridruženog tom krevetu. Nakon završetka pretrage, stanje se prebacuje u uspjeh, a ako je pronašla odgovarajućeg pacijenta, funkcija vraća njegovu instancu ili u suprotnom vraća praznu vrijednost. Ako je došlo do pogreške za vrijeme pretrage pacijenata, stanje se prebacuje u stanje greške, a povratna vrijednost se sprema u tekstualnu varijablu *error*. Na temelju prethodno objašnjenog principa upravlja se sa stanjima svih zaslona koji imaju stanje.

```
@freezed
class NfcState with _$NfcState {
  const factory NfcState({
    @Default("") String nfcId,
    @Default(Bed()) Bed bed,
    @Default(Patient()) Patient patient,
    @Default(AppState.initial) AppState appState,
    @Default("") String error,
  }) = _NfcState;
}
```

```
class NfcNotifier extends StateNotifier<NfcState> {
  NfcNotifier(this.ref) : super(const NfcState());

  final Ref ref;

  void setId(String id) {
    state = state.copyWith(nfcId: id);
  }

  void setInitialState() {
    state = state.copyWith(error: '', appState: AppState.initial);
  }

  Future<Bed?> getBedByNfcId() async {
    Bed? bed;
    state = state.copyWith(appState: AppState.loading);
    try {
      bed = await ref
          .read(firestoreRepositoryProvider)
          .getBedByNfcId(nfcId: state.nfcId);
      if (bed != null) {
        state = state.copyWith(bed: bed);
      }
    } catch (e) {
      state = state.copyWith(appState: AppState.error, error: e.toString());
    }
    state = state.copyWith(appState: AppState.initial);
    return bed;
  }

  Future<Patient?> getPatientByBed({required Bed bed}) async {
    Patient? patient;
    state = state.copyWith(appState: AppState.loading);
    try {
      patient = await ref
          .read(firestoreRepositoryProvider)
          .getPatientByBed(patientId: bed.patientId);
      if (patient != null) {
        state = state.copyWith(patient: patient);
      }
    } catch (e) {
      state = state.copyWith(appState: AppState.error, error: e.toString());
    }
    state = state.copyWith(appState: AppState.success);
    return patient;
  }
}

final nfcProvider =
    StateNotifierProvider.autoDispose<NfcNotifier, NfcState>((ref) {
  return NfcNotifier(ref);
});
```

Slika 4.9. Prikaz rukovanja s jednim od stanja Care Mate aplikacije

4.3.3. Korisničko sučelje

Održavanje korisničkog sučelja čitkim vrlo je bitna stavka kod izrade mobilnih aplikacija. Elemente koji se ponavljaju potrebno je pohraniti kao zasebne klase te ih tako implementirati u aplikaciju. Ovisno o stanju zaslona potrebno je prikazivati određene elemente kao što je prikazivanje obavijesti na dnu ekrana ukoliko je došlo do greške ili promjene zaslona, ukoliko je to potrebno. Na primjeru koda vidljivog na slici 4.10. prikazana je implementacija primjene stanja aplikacije na zaslone. Prilikom stvaranja zaslona *NfcView* u njegovom početnom stanju (engl. *Init state*) pokreće se funkcija zaslužena za čitanje NFC naljepnica i manipuliranje pročitanim podacima. Čitanje s NFC naljepnica omogućeno je pomoću paketa *FlutterNfcKit*. Funkcija započinje traženje i čitanje s NFC naljepnica. Kada pročita NFC naljepnicu, poziva funkciju koja pretražuje Firestore bazu podataka postoji li krevet koji je povezan s pronađenom NFC naljepnicom. Ako postoji povezan krevet, odnosno ako vrijednost nije jednaka *null*, započinje pretragu pacijenta pomoću pronađenog kreveta. Pretraga pacijenta pomoću kreveta detaljno je objašnjena u prethodnom poglavlju. Završetkom pretrage pacijenata poziva se funkcija koja gasi proces traženja signala NFC naljepnica. Za vrijeme izvršavanja ovih funkcija i pretraga po Firestore bazi podataka, slušatelj (engl. *Listener*) je osluškivao kako se mijenja stanje zaslona. Ako je stanje zaslona iz učitavanja prešlo u uspjeh i vrijednost pacijenta nije jednaka početnoj vrijednosti, pozivaju se funkcije za popunjavanje pacijentovih podataka, stanje NFC zaslona se postavlja na početno i vrši se navigacija na idući zaslon koji je u ovom slučaju prikaz pacijentovih podatka. Ako je došlo do pogreške unutar funkcija za pretragu te se stanje aplikacije promijenilo iz stanja *učitavanje* u stanje *greška*, na dnu zaslona prikazuje se obavijest s tekstom greške.

Tijelo (engl. *Body*) zaslona prikazuje definirane elemente. Kada je aplikacija u stanju učitavanja prikazuje se indikator učitavanja, a u suprotnom se prikazuju elementi ovisno o stanju aplikacije. Prilikom prikazivanja elemenata na zaslonu pravi se dodatna provjera je li prijavljeni korisnik ovlaštena osoba, a ako jest, njemu se prikazuju dodatni elementi ili u ovom slučaju drugačiji elementi od običnog korisnika. Prilikom očitavanja NFC naljepnice koja nije povezana s krevetom, ovlaštenom se korisniku prikazuje gumb za povezivanje NFC naljepnice i kreveta, za razliku do uobičajenog korisnika kojemu se prikazuje tekstualna poruka da se obrati ovlaštenom korisniku.

```dart
class NfcView extends ConsumerStatefulWidget {
  const NfcView({super.key});

  @override
  ConsumerState<NfcView> createState() => _DiscoveryViewState();
}

class _DiscoveryViewState extends ConsumerState<NfcView> {
  Bed? bed;
  Patient? patient;

  @override
  void initState() {
    // TODO: implement initState    TODO: implement initState
    super.initState();
    _scanNfc();
  }

  Future<void> _scanNfc() async {
    try {
      //get NFC data and set id to state
      NFCTag tag = await FlutterNfcKit.poll(androidPlatformSound: true);
      ref.read(nfcProvider.notifier).setId(tag.id);
      //get bed if nfc is connected to one
      bed = await ref.read(nfcProvider.notifier).getBedByNfcId();
      //get the patient if he is lying in bed
      if (bed != null) {
        patient =
            await ref.read(nfcProvider.notifier).getPatientByBed(bed: bed!);
      }
      await FlutterNfcKit.finish();
    } catch (e) {
      print(    Don't invoke 'print' in production code.Try using a logging framework.
        e.toString(),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    var provider = ref.watch(nfcProvider);
    var adminProvider = ref.watch(userProvider);

    ref.listen(nfcProvider, (previous, next) {
      if (next.appState == AppState.success &&
          previous?.appState == AppState.loading) {
        if (next.patient != const Patient()) {
          ref.read(tabsProvider.notifier).setInitialPatientData(next.patient);
          ref.read(initialPatientProvider.notifier).state = next.patient;

          ref.read(nfcProvider.notifier).setInitialState();
          GoRouter.of(context).pushReplacement(AppRoutes.patientTabs);
        }
      } else if (next.appState == AppState.error &&
          previous?.appState == AppState.loading) {
        WidgetsBinding.instance.addPostFrameCallback(
          (timeStamp) {
            showAppSnackBar(
                context: context,
                text: next.error,
                closedCallback: (value) {
                  if (mounted) {
                    ref.read(nfcProvider.notifier).setInitialState();
                  }
                });
          },
        );
      }
    });
    return Scaffold(
      appBar: AppBar(
        title: const Text('Scan NFC'),
      ), // AppBar
      body: provider.appState == AppState.loading
          ? const CustomLoadingIndicator()
          : Center(
              child: provider.nfcId == ''
                  ? const Text("Searching for NFC...")
                  : bed == null
                      ? adminProvider.isAdmin
                          ? const AdminConnectNfcToBedView()
                          : const Text(
                              "Contat admin to connect this NFC to a bed") // Text
                      : Column(
                          mainAxisAlignment: MainAxisAlignment.center,
                          children: [
                            const Text("This bed is empty"),
                            const SizedBox(height: 20),
                            ElevatedButton(
                              onPressed: () {
                                GoRouter.of(context)
                                    .push(AppRoutes.patientSearch);
                              },
                              child: const Text("Add patient to this bed"),
                            ), // ElevatedButton
                          ],
                        ), // Column
            ), // Center
    ); // Scaffold
  }
}

class AdminConnectNfcToBedView extends StatelessWidget {
  const AdminConnectNfcToBedView({super.key});

  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        const Text("This NFC is empty"),
        ElevatedButton(
          onPressed: () {
            GoRouter.of(context).push(AppRoutes.hospitalFLoors);
          },
          child: const Text("Connect this NFC to a bed"),
        ), // ElevatedButton
      ],
    ); // Column
  }
}
```

Slika 4.10. Kod zaslona za upravljanje NFC naljepnicama

Primjer elementa koji je namijenjen za višekratnu upotrebu prikazan je na slici 4.11. U danom primjeru ovisno o predanom parametru *isFullDateTimeFormat*, element prikazuje tekstualno polje čijim se odabirom prikazuje birač datuma. Ako je vrijednost predanog parametra negativna (engl. *False*), prikazuje se birač s mogućnostima odabira godine, mjeseca i dana prilagođenog za odabir datuma rođena. Ako je predani parametar točan (engl. *True*), osim birača za odabir godine, mjeseca i dana dodatno se prikazuje birač sata i minuta namijenjen za točnu odredbu vremena koja se koristi za definiranje vremena kada su napravljena mjerenja temperature i krvnog tlaka.

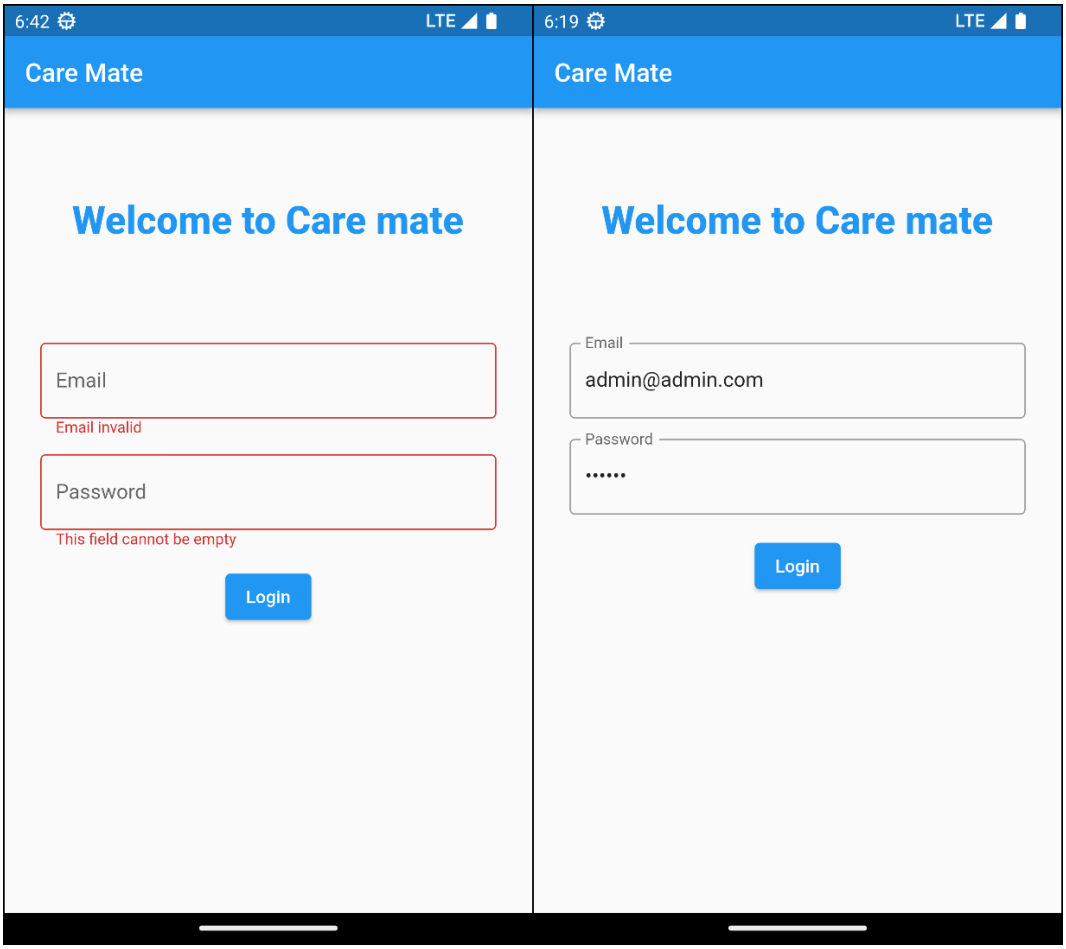
```dart
class _CustomTextFormFieldDatePickerState
    extends State<CustomTextFormFieldDatePicker> {
  final TextEditingController _dateController = TextEditingController();

  @override
  void initState() {
    super.initState();
    _dateController.text = widget.initialValueString;
  }

  Future<void> _selectDate(BuildContext context) async {
    DateTime? pickedDate = await showDatePicker(
      context: context,
      initialDate: widget.initialValueString.isNotEmpty
          ? DateFormat('dd/MM/yyyy').parse(widget.initialValueString)
          : DateTime.now(),
      firstDate: DateTime(1900),
      lastDate: DateTime.now(),
    );

    if (pickedDate != null) {
      if (widget.isFullDateTimeFormat) {
        TimeOfDay? pickedTime = await showTimePicker(    Don't use 'BuildContext's across asyn
          context: context,
          initialTime:
              TimeOfDay(hour: pickedDate.hour, minute: pickedDate.minute),
        );

        if (pickedTime != null) {
          pickedDate = DateTime(pickedDate.year, pickedDate.month,
              pickedDate.day, pickedTime.hour, pickedTime.minute);
        }
      }

      setState(() {
        _dateController.text = widget.isFullDateTimeFormat
            ? DateFormat('dd/MM/yyyy HH:mm').format(pickedDate!)
            : DateFormat('dd/MM/yyyy').format(pickedDate!);
        widget.setDate!(_dateController.text);
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return TextFormField(
      controller: _dateController,
      readOnly: true,
      enabled: widget.setDate != null ? true : false,
      onTap: () {
        _selectDate(context);
      },
      style: TextStyle(
          color: widget.setDate == null ? Colors.grey[400] : Colors.black), // TextStyle
      decoration: InputDecoration(
        labelText: widget.lableText,
        border: const OutlineInputBorder(),
        suffixIcon: const Icon(Icons.calendar_today),
        errorStyle: const TextStyle(height: 0.3),
      ), // InputDecoration
      validator: (value) {
        return value!.isEmpty ? "This field cannot be empty" : null;
      },
    ); // TextFormField
  }
}

```

Slika 4.11. Kod elementa za odabir datuma namijenjen za višekratnu upotrebu

5. KORIŠTENJE APLIKACIJOM

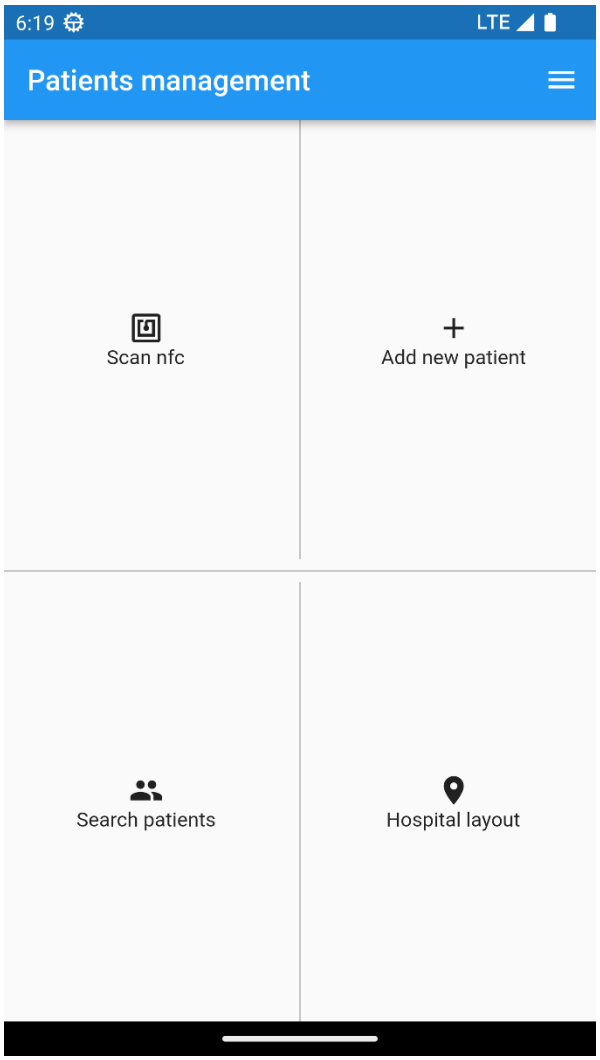
Unutar ovog poglavlja detaljno je opisan tijek aplikacije potkrijepljen slikama zaslona mobilne aplikacije Care Mate.

Na prvom zaslonu aplikacije medicinski radnik/radnica upisuje njemu/njoj prethodno dodijeljene podatke koje prolaze provjeru autentičnosti (Slika 5.1).



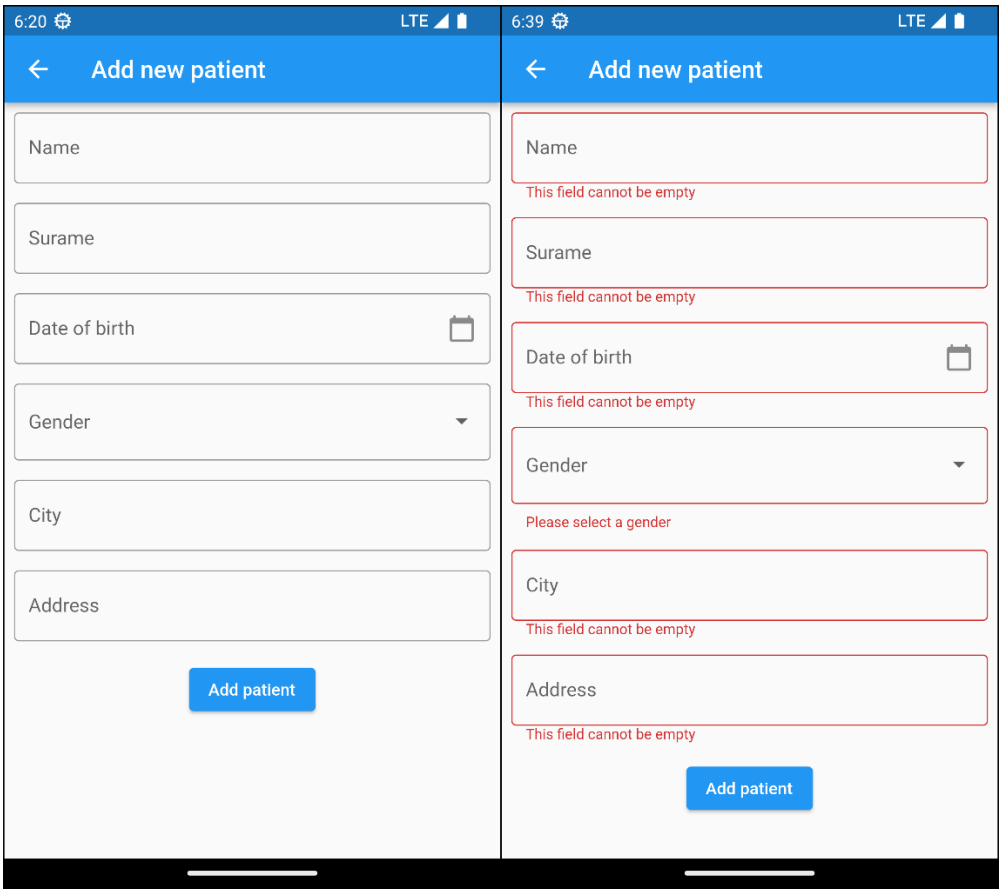
Slika 5.1. Zaslon prijave u aplikaciju

Ukoliko su podaci ispravni, otvara se početni ekran s četiri velika gumba. Otvoreni je zaslon početni zaslon iz kojeg je moguće pristupiti svim funkcionalnostima aplikacije (Slika 5.2).



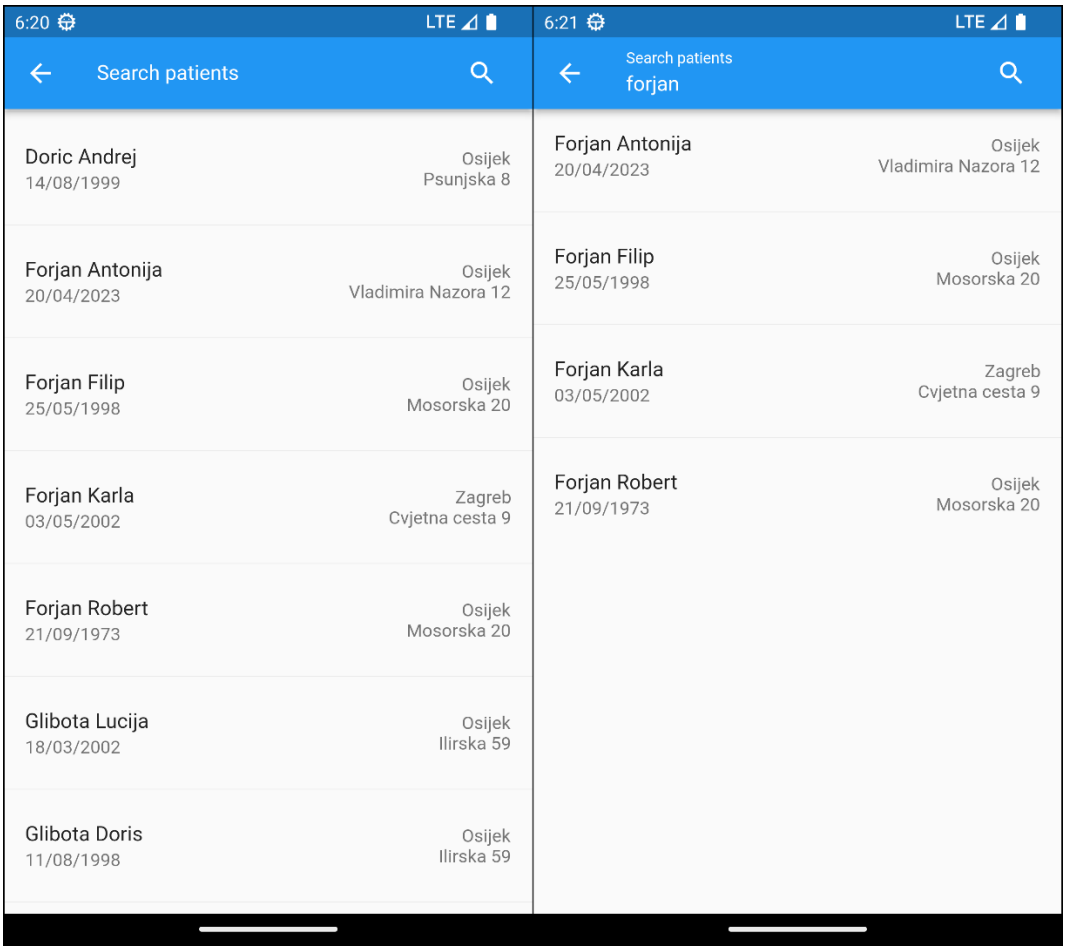
Slika 5.2. Početni zaslon

U gornjem desnom uglu nalazi se gumb čijim se pritiskom otvara zaslon za dodavanje novog pacijenta. Zaslon sadrži osnovne pacijentove podatke. Potrebno je unijeti sve podatke jer u suprotnom kreiranje novog pacijenta nije moguće. Zaslon se sastoji od tekstualnih polja kao što su ime, prezime, grad i adresa, kalendarski odabir datuma rođenja i padajući izbornik pomoću kojeg se odabire spol pacijenta. Nakon upisivanja svih podataka pritiskom na gumb, pacijent će biti pohranjen (Slika 5.3).



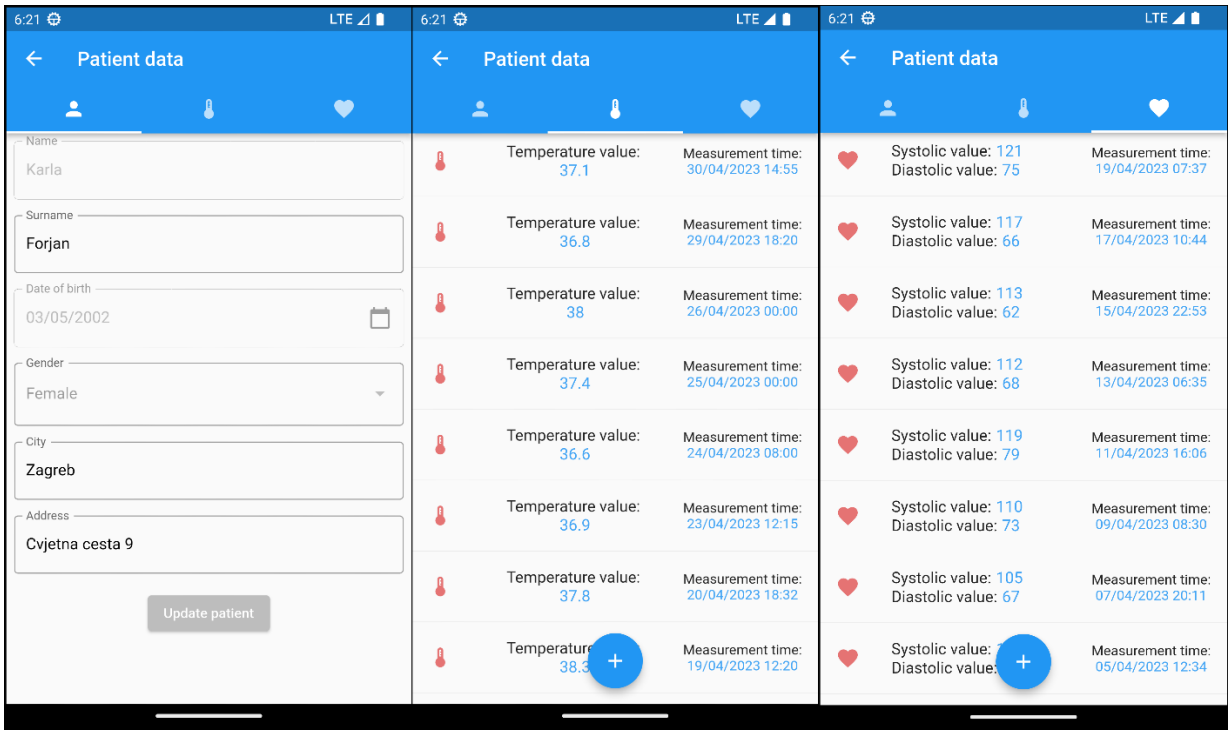
Slika 5.3. Zaslon za dodavanje novog pacijenta

Donji lijevi gumb otvara zaslon s popisom svih pacijenata, a na vrhu zaslona nalazi se tražilica pomoću koje je olakšan pronalazak željenog pacijenta. Tražilica vraća rezultat preklapanja upisanog teksta s imenom i prezimenom pacijenta, odnosno moguće je pretraživati po imenu i prezimenu pacijenta (Slika 5.4).



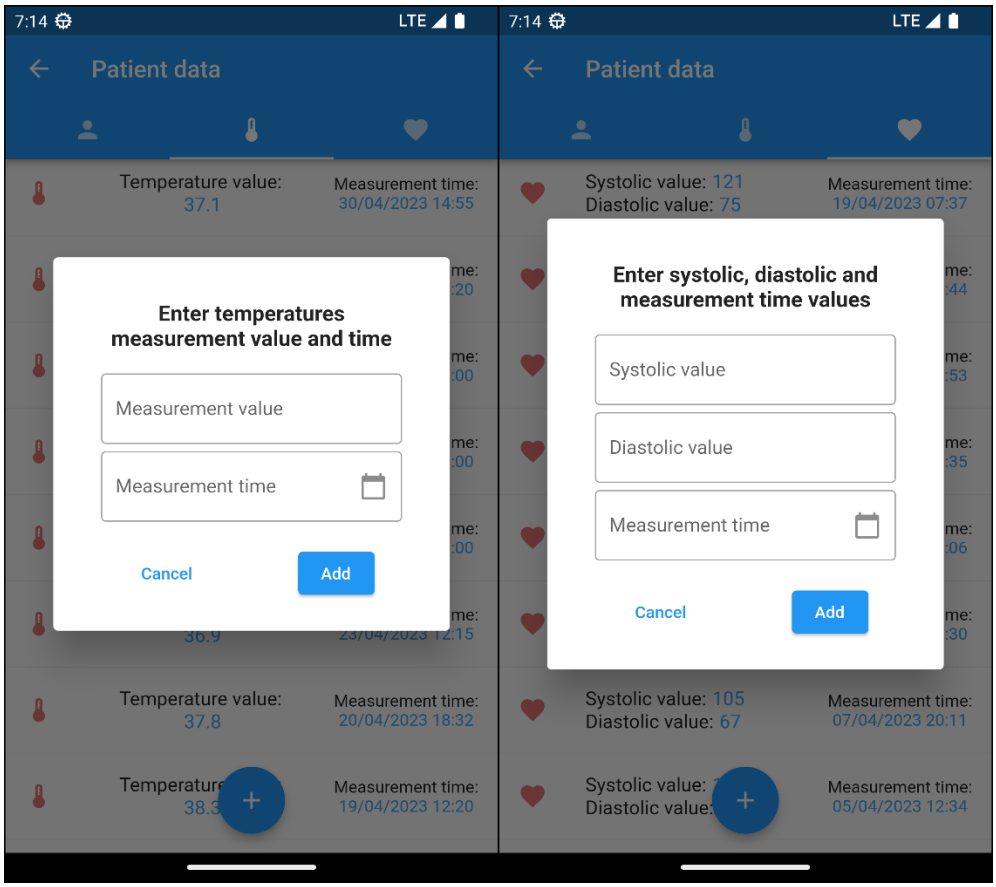
Slika 5.4. Zaslon liste pacijenata s tražilicom

Odabirom jednog od ponuđenih pacijenata otvara se zaslon s tri kartice (engl. *Tab*). Prva kartica istog je izgleda kao i zaslon za dodavanje novog pacijenta, ali s već popunjenim pacijentovim podacima. Umjesto gumba za dodavanje pacijenta nalazi se gumb za ažuriranje pacijentovih podataka. Neke podatke nije moguće izmijeniti unutar tog zaslona. Druga kartica prikazuje povijest mjerenja pacijentove temperature poslagane od najnovijeg mjerenja prema najstarijem. Posljednja kartica prikazuje povijest mjerenja pacijentovog krvnog tlaka (Slika 5.5).



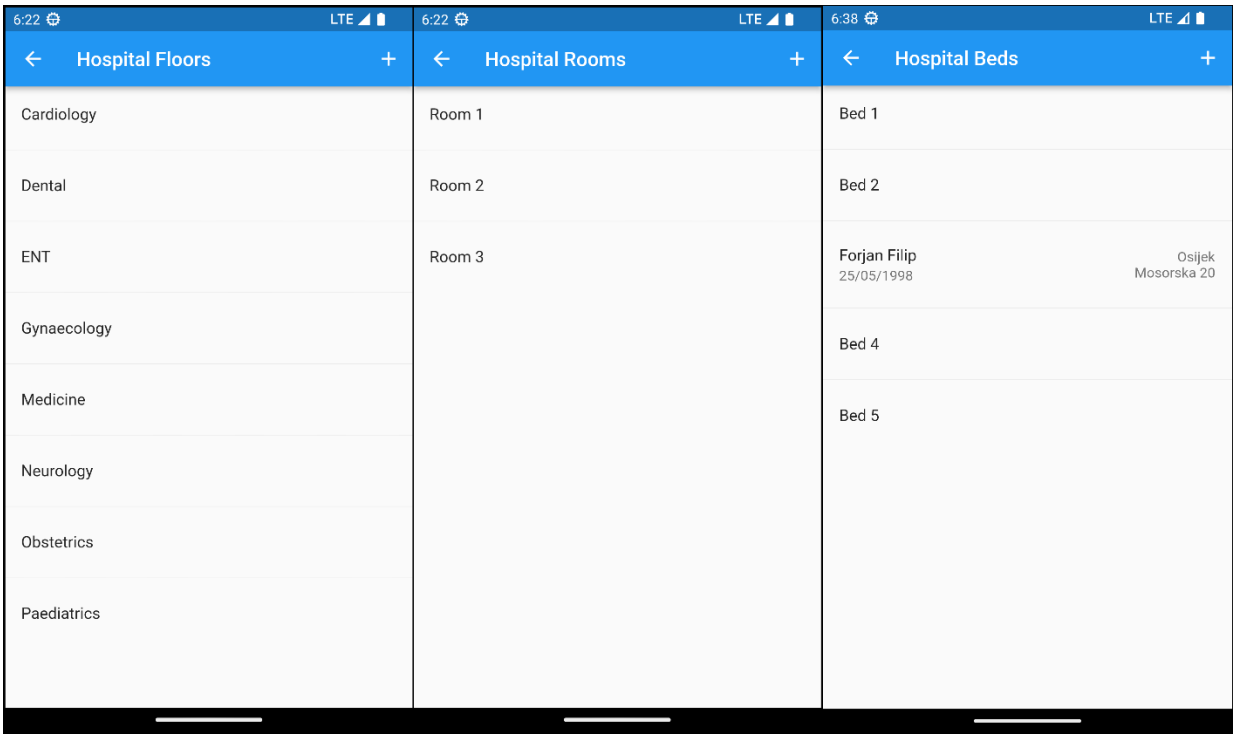
Slika 5.5. Zasloni pacijentovih podataka i povijesti mjerenja temperature i krvnog tlaka

Kartice koje bilježe povijest mjerenja temperature i krvnog tlaka na dnu sadrže gumb za dodavanje novog mjerenja u ovisnosti na kojoj kartici se korisnik nalazi. Pritiskom na gumb otvara se prozor u koji je potrebno unijeti podatke ovisno unosi li se mjerenje temperature ili krvnog tlaka. Unos novih mjerenja temperature i krvnog tlaka prikazan je na slici 5.6.



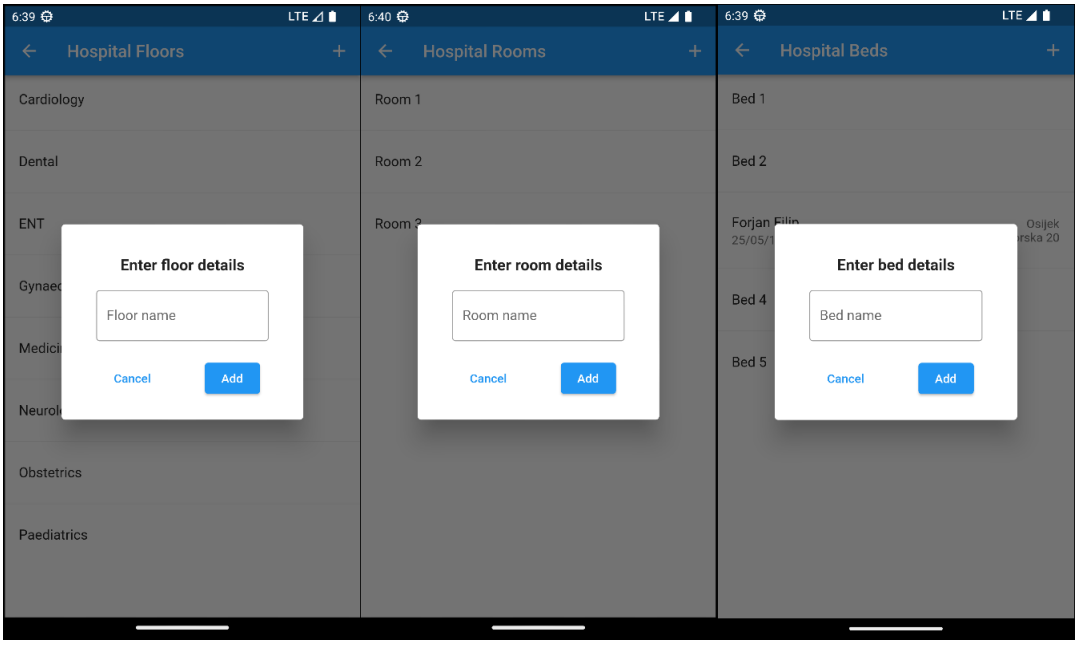
Slika 5.6. Zasloni za dodavanje novih mjerenja temperature i krvnog tlaka

Na početnom zaslonu aplikacije donji desni gumb otvara zaslon koji prikazuje popis katova u bolnici. Odabirom jednog od katova prikazuje se popis soba na odabranom katu, a odabirom jedne od soba prikazuje se popis postojećih kreveta unutar odabrane sobe. Ako se u jednom od kreveta nalazi pacijent, umjesto imena kreveta pokazat će se podaci pacijenta, a odabirom tog pacijenta otvara se prethodno objašnjen zaslon s pacijentovim podacima i povijestima mjerenja temperature i krvnog tlaka (Slika 5.7).



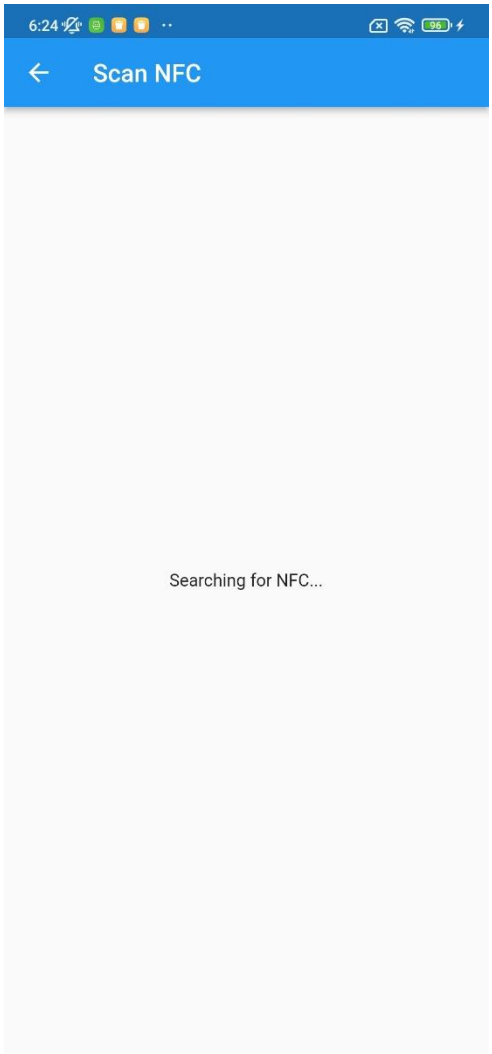
Slika 5.7. Zasloni katova, soba i kreveta bolnice

Ovlašteni korisnik ima mogućnost dodavanja novog kata, sobe ili kreveta odabirom ikone u gornjem desnom uglu zaslona ovisno na kojem zaslonu se nalazi. Pritiskom na gumb otvara se prozor u koji je potrebno upisati ime kata, sobe ili kreveta (ovisno koji se podatak želi dodati) (Slika 5.8).



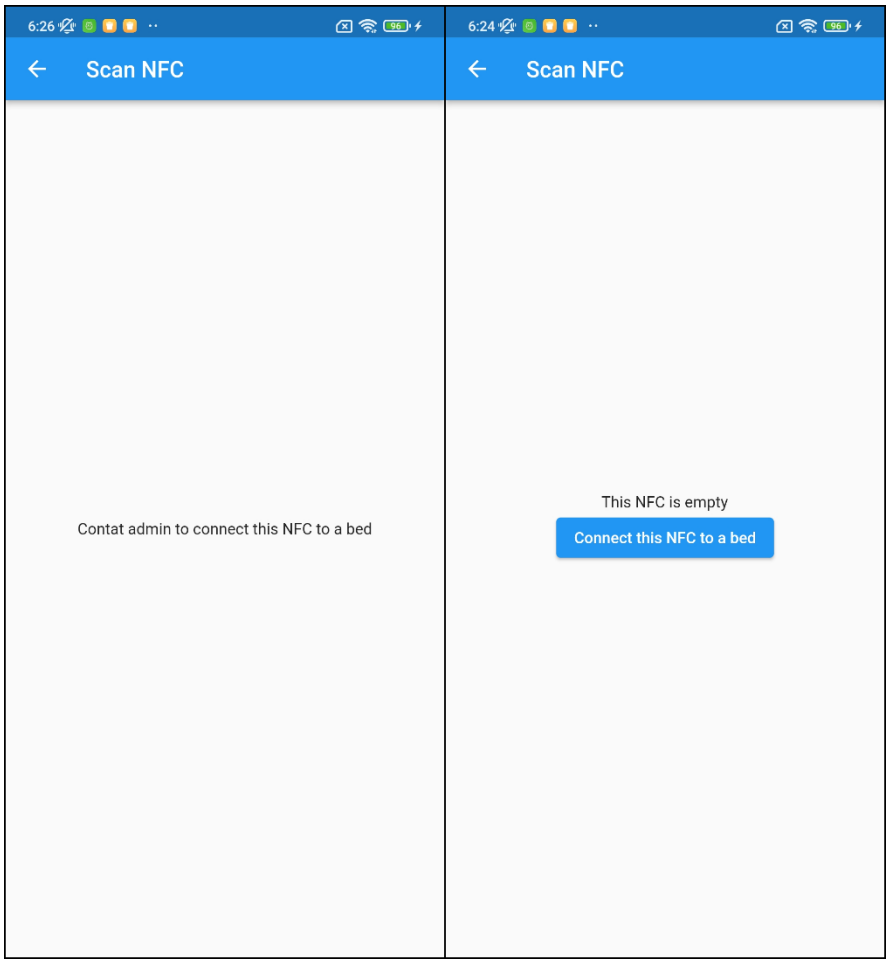
Slika 5.8. Zasloni dodavanja novog kata, sobe i kreveta

Posljednji se gumb nalazi u gornjem lijevom uglu početnog zaslona, a njegovim otvaranjem prikazuje se zaslon koji očekuje skeniranje NFC naljepnice. Zaslon na kojemu se skeniraju NFC naljepnice prikazan je na slici 5.9.



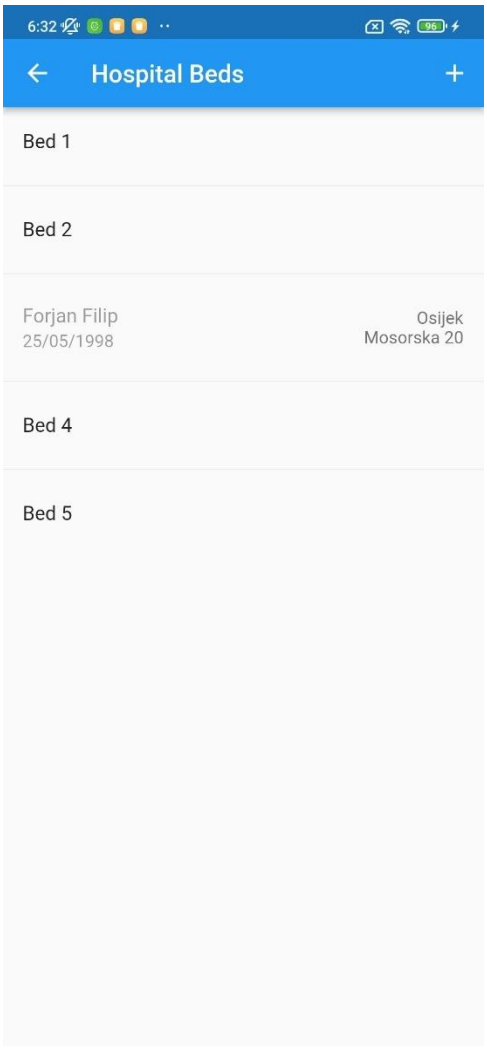
Slika 5.9. Zaslon skeniranja NFC naljepnice

Skeniranjem NFC naljepnice, ukoliko ona nije povezana s krevetom, medicinskom osoblju prikazuje se poruka kako je nužno zvati ovlaštenog korisnika kako bi se povezala NFC naljepnica s krevetom. To je moguće napraviti tako što se osobi koja ima ovlašteni pristup, skeniranjem prazne naljepnice prikazuje gumb koji otvara popis katova (Slika 5.10).



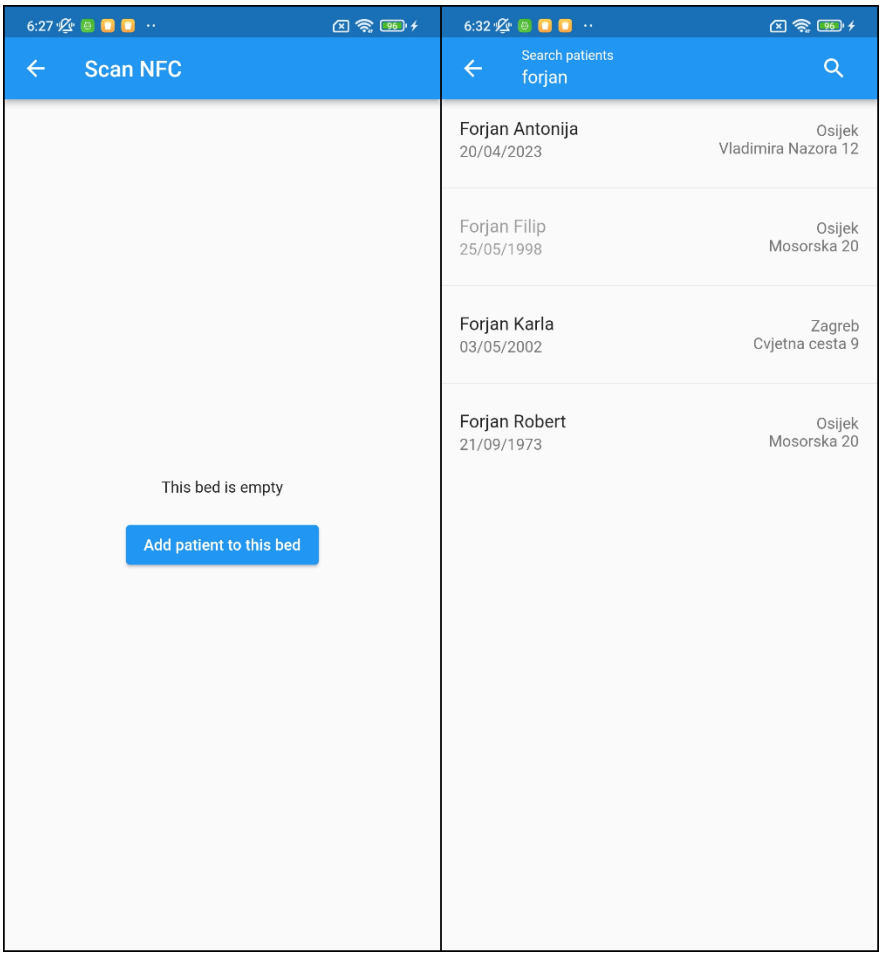
Slika 5.10. Zaslon skenirane NFC naljepnice koja nije povezana s krevetom

Odabirom željenog kata, sobe i na posljetku kreveta, ovlaštena osoba završava povezivanje NFC naljepnice i kreveta. Kreveti koji su povezani s pacijentom, prikazani su sivom bojom i sadrže pacijentove podatke. Te krevete nije moguće odabrati za ponovno povezivanje s NFC naljepnicom (Slika 5.11).



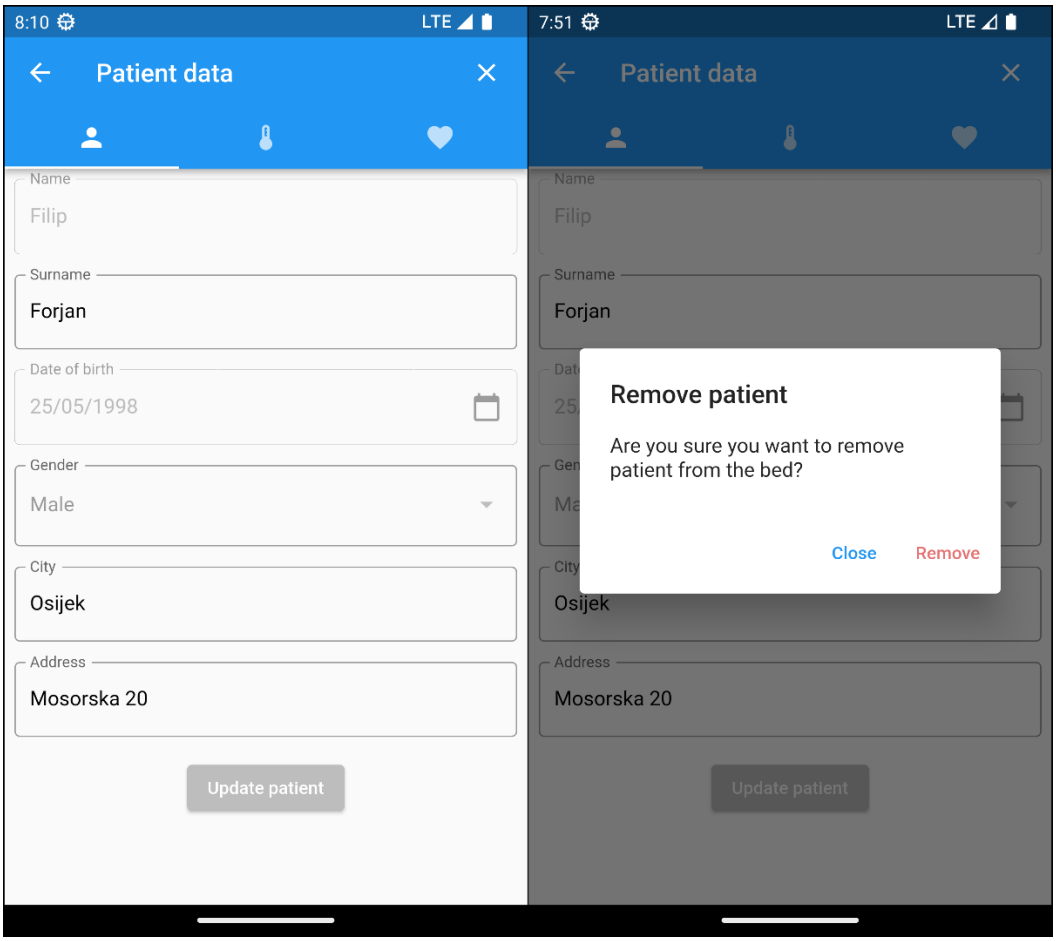
Slika 5.11. Povezivanje prazne NFC naljepnice i kreveta

Ako nakon skeniranja NFC naljepnice na prethodno povezan krevet nije spojen ni jedan pacijent, ovlaštenoj osobi ili medicinskom osoblju otvara se gumb koji otvara zaslon s popisom svih pacijenata. Nije moguće odabrati pacijenta ako je već povezan s jednim krevetom. Odabirom željenog pacijenta, ostvaruje se poveznica pacijenta s krevetom i otvara se zaslon s pacijentovim podacima i povijestima mjerenja (Slika 5.12).



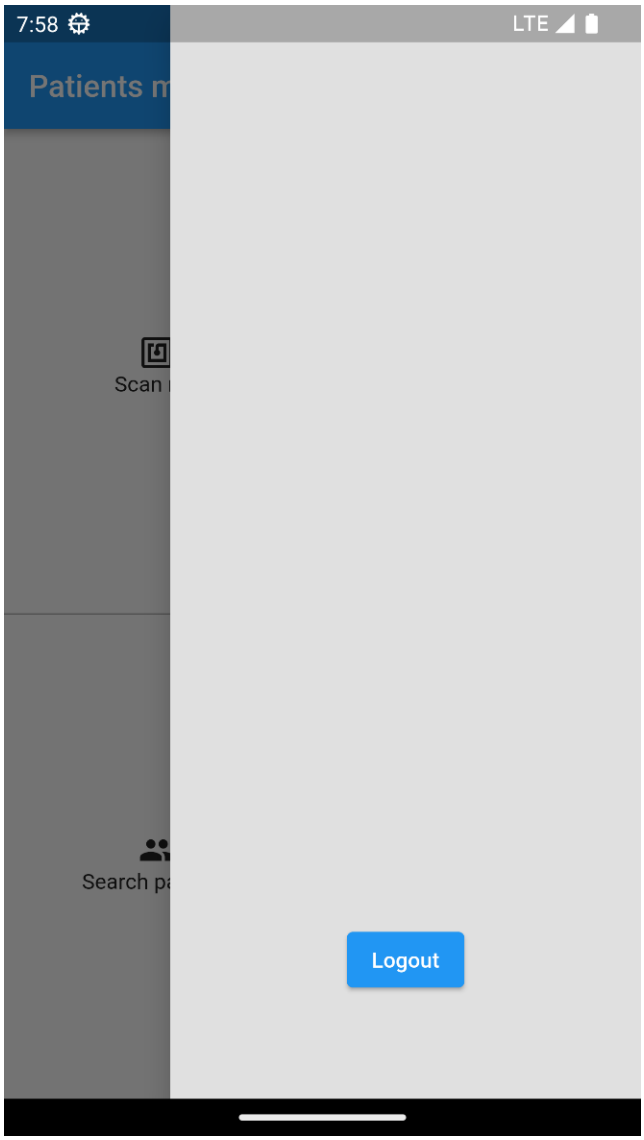
Slika 5.12. Zasloni za povezivanje pacijenta i kreveta

Ako je pacijent spojen s jednim od kreveta, u gornjem desnom uglu kartica nalazi se ikona za uklanjanje pacijenta. Pritiskom na nju otvara se skočni zaslon za dodatnu potvrdu ako želimo ukloniti poveznicu pacijenta i kreveta (Slika 5.13).



Slika 5.13. Zaslon prozora za uklanjanje veze pacijenta i kreveta

Posljednja funkcionalnost aplikacije je odjavljivanje trenutno prijavljenog korisnika, a tome se pristupa s početnog zaslona odabirom ikone na alatnoj traci u gornjem desnom uglu. Otvara se bočni izbornik s jednim gumbom čijim se pritiskom korisnik odjavljuje iz aplikacije. Nakon odjavljivanja prikazuje se zaslon za prijavu korisnika (Slika 5.14).



Slika 5.14. Zaslon za odjavu iz aplikacije

6. ZAKLJUČAK

Zadatak ovog diplomskog rada bio je izraditi mobilnu aplikaciju za evidentiranje podataka pacijenata i pristupati tim podacima putem NFC tehnologije. Aplikacija je izrađena koristeći Flutter razvojni okvir dok je za pisanje korišten programski jezik Dart. Prvi dio ovog diplomskog rada je uvod u temu, gdje su navedeni razlozi odabira izrade Care Mate aplikacije i navedene su korištene tehnologije za njenu izradu. Drugo poglavlje daje uvid na trenutno tržište sličnih mobilnih aplikacija na temu ovog diplomskog rada. Navedene su i ukratko objašnjene aplikacije čija primjena ima slične funkcionalnosti Care Mate-u. Treće poglavlje detaljnije opisuje korištene tehnologije za izradu mobilne aplikacije i korištene baze podataka za pohranu pacijenata. Četvrto poglavlje najprije daje uvid u funkcionalnosti izrađene aplikacije i njezinu moguću primjenu u svakodnevnom životu te daje osvrt na korisničko sučelje aplikacije. Zatim detaljno opisuje razvoj provjere autentičnosti i baze podataka koristeći Firebase alate, dok je na kraju opisan razvoj mobilne aplikacije podijeljen na izradu arhitekture, upravljanje stanjem aplikacije i korisničkog sučelja. Peto poglavlje daje uvid u tijek rada aplikacije iz perspektive korisnika.

Care Mate aplikacija ima velike mogućnosti nadogradnje. Osim bilježenja osnovnih podataka pacijenata moguće je proširiti spektar na specifične detalje poput alergija pacijenata, propisane terapije i rezultati raznih testiranja. Moguće je dodati raspored uzimanja propisane terapije pacijenata i na osnovu toga implementirati notifikacije koje će obavijestiti medicinsko osoblje o potrebnom davanju terapije pacijentima. Aplikacija se također može proširiti dodavanjem lokalizacije, kako bi ju bilo moguće koristiti na više jezika.

LITERATURA

[1] „Cerner“, *Wikipedia*. 07. travanj 2023. Pristupljeno: 01. svibanj 2023. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Cerner&oldid=1148663091

[2] „cerner corporation - Google Search“. https://www.google.com/search?q=cerner+corporation&tbm=isch&source=lnms&sa=X&ved=2ahUKEwislf-S79L-AhUE_SoKHS2UANYQ_AUoAnoECAEQBA&biw=1920&bih=929&dpr=1#imgrc=PR3fzfIr-2jqkM (pristupljeno 01. svibanj 2023.).

[3] „Connect Nursing“, *App Store*, 21. siječanj 2022. https://apps.apple.com/us/app/connect-nursing/id1524456577 (pristupljeno 01. svibanj 2023.).

[4] „Cerner Mobile Care“, *App Store*, 05. travanj 2023. https://apps.apple.com/us/app/cerner-mobile-care/id1548508618 (pristupljeno 01. svibanj 2023.).

[5] „Epic Haiku & Limerick“, *App Store*, 04. travanj 2023. https://apps.apple.com/us/app/epic-haiku-limerick/id348308661 (pristupljeno 01. svibanj 2023.).

[6] „NFC Tools“, *App Store*, 08. veljača 2023. https://apps.apple.com/us/app/nfc-tools/id1252962749 (pristupljeno 01. svibanj 2023.).

[7] „Visual Studio Code - Code Editing. Redefined“. https://code.visualstudio.com/ (pristupljeno 01. svibanj 2023.).

[8] „Flutter - Build apps for any screen“. //flutter.dev/ (pristupljeno 01. svibanj 2023.).

[9] ERIC WINDMILL i Ray Rischpater, *Flutter IN ACTION*. Manning.

[10] Rap Payne, *Beginning App Development with Flutter,Create Cross-Platform Mobile Apps*. apress, 2019.

[11] „Stateful vs. Stateless • Cloudification - Get more from Cloud and Open Source 🚀☁️“, 09. veljača 2022. https://cloudification.io/2022/02/09/stateful-vs-stateless/ (pristupljeno 01. svibanj 2023.).

[12] „Top Flutter State Management Libraries in 2023 - Foresight Mobile“. https://foresightmobile.com/blog/top-flutter-state-management-libraries-in-2023 (pristupljeno 01. svibanj 2023.).

[13] „Riverpod“. https://riverpod.dev/ (pristupljeno 01. svibanj 2023.).

[14] „Introducción al patrón BLoC“, *XurxoDev*, 26. ožujak 2020. https://xurxodev.com/introduccion-al-patron-bloc/ (pristupljeno 01. svibanj 2023.).

[15] „Dart (programming language)“, *Wikipedia*. 18. ožujak 2023. Pristupljeno: 01. svibanj 2023. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Dart_(programming_language)&oldid=1145322174

[16] „Dart programming language“. https://dart.dev/ (pristupljeno 01. svibanj 2023.).

[17] „Firebase“, *Firebase*. https://firebase.google.com/ (pristupljeno 02. svibanj 2023.).

[18] „Firebase Authentication“, *Firebase*. https://firebase.google.com/docs/auth (pristupljeno 02. svibanj 2023.).

[19] „Firestore“, *Firebase*. https://firebase.google.com/docs/firestore (pristupljeno 02. svibanj 2023.).

[20] „What is near-field communication (NFC)?“, *Mobile Computing*. https://www.techtarget.com/searchmobilecomputing/definition/Near-Field-Communication (pristupljeno 02. svibanj 2023.).

[21] „Near-field communication“, *Wikipedia*. 26. travanj 2023. Pristupljeno: 02. svibanj 2023. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Near-field_communication&oldid=1151800377

[22] „Add Firebase to your Flutter app“, *Firebase*. https://firebase.google.com/docs/flutter/setup (pristupljeno 02. svibanj 2023.).

SAŽETAK

Zadatak ovog diplomskog rada bio je izraditi mobilnu aplikaciju za pojednostavljeno praćenje pacijentovih podataka. Aplikacija je namijenjena medicinskom osoblju. Za izradu aplikacije korišten je programski okvir Flutter, a pisana je pomoću programskog jezika Dart. Baza podataka pacijenta i provjera autentičnosti korisnika izvršena je pomoću Firebase alata, a za kompletnu izradu aplikacije korišteno je programsko okruženje Visual Studio Code. Medicinskom osoblju brzi pristup pacijentovim podacima omogućen je korištenjem NFC tehnologije. Korisnici aplikacije odnosno medicinski radnici imaju mogućnost dodavanja novih pacijenata, uvid i pretragu pacijenata pomoću tražilice, uvid u cjelokupni raspored bolnice, njezinih katova, raspored soba i kreveta. Ako se u jednom od prikazanih kreveta nalazi pacijent, korisnik ima mogućnost pristupa njegovim podacima odabirom tog pacijenta. Glavna funkcionalnost aplikacije koja olakšava medicinskom osoblju pristup pacijentima koji se trenutno nalaze u bolnici omogućen je skeniranjem NFC naljepnice. Ovlašteni korisnik prvobitno spaja NFC naljepnicu s postojećim krevetom, a zatim medicinsko osoblje može dodavati i uklanjati pacijente s tog kreveta. Skeniranjem NFC naljepnice prikazuje se pacijentov karton.

Ključne riječi: Firebase, Flutter, Medicinsko osoblje, Mobilna aplikacija, NFC

ABSTRACT

Mobile application for tracking patient records using NFC technology

The task of this master's thesis was to develop a mobile application for simplified tracking of patient data, intended for medical staff. The Flutter framework was used to create the application, written in the Dart programming language. Patient database and user authentication were implemented using Firebase tools, while Visual Studio Code was used as the programming environment for the entire application development. Quick access to patient data for medical personnel is enabled through the use of NFC technology. Application users or medical workers can add new patients, view and search patients using the search bar, access the entire hospital schedule, its floors, room schedules, and beds. If a patient is located in one of the displayed beds, the user can access the patient's data by selecting that patient. The main functionality of the application, which facilitates medical staff's access to patients currently in the hospital, is enabled by scanning an NFC tag. Authorized users initially connect the NFC tag to an existing bed, and then medical personnel can add and remove patients from that bed. Scanning the NFC tag displays the patient's record.

Keywords: Flutter, Firebase, Medical staff, Mobile application, NFC

ŽIVOTOPIS

Filip Forjan rođen je u Osijeku 25.5.1998. godine. Pohađa Programsko inženjerstvo na diplomskom studiju Fakulteta elektrotehnike, računarstva i informacijskih tehnologija u nastavku FERIT. Tijekom diplomskog studija jedan semestar završava u Švedskoj na fakultetu Mälardalens universitet. Akademski naziv sveučilišni prvostupnik (baccalaureus) inženjer Računarstva stekao je 2020. godine na FERIT-u. Srednjoškolsko školovanje završio je u osječkoj III. Gimnaziji, Prirodoslovno-matematičkoj. Tijekom fakultetskog obrazovanja radno iskustvo stječe razvijajući mobilne aplikacije u jednoj od osječkih firmi.

_____

Potpis autora