Mobilna Android aplikacija za evidenciju nazočnosti studenata pomoću RFID/NFC čitača iksica

Benković, Mato

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

Sveučilišni prijediplomski studij Računarstvo

Mobilna Android aplikacija za evidenciju nazočnosti studenata pomoću RFID/NFC čitača iksica

Završni rad

Mato Benković

Osijek, 2024.

SADRŽAJ

1. UVOD 1

1.1. Zadatak završnog rada 2

2. ISTRAŽIVANJE POSTOJEĆIH RJEŠENJA 3

2.1. MyClass attendance 3

2.2. Attendance tracker by Jibble 4

3. KORIŠTENE TEHNOLOGIJE 5

3.1. Hardverske komponente 5

3.1.1. Dasduino ESP32 Connectplus 5

3.1.2. EasyC adapter 6

3.1.3. PN532 NFC RFID Module V4 6

3.2. Softver 7

3.2.1. Arduino IDE 7

3.2.2. Firebase 7

3.2.3. Android Studio i Kotlin 8

4. PROGRAMIRANJE HARDVERSKIH KOMPONENTI 9

4.1. Implementacija Arduino koda 9

4.2. Uključivanje biblioteka i definiranje pinova za komunikaciju 9

4.3. Inicijalizacija NDC i Bluetooth objekata 10

4.4. Setup funkcija 10

4.5. Loop funkcija 11

5. RAZVOJ APLIKACIJE I PRIKAZ FUNKCIONALNOSTI 13

5.1. Upravljanje podatcima 13

5.2. Upravljanje autentifikacjiom korisnika 14

5.3. Interakcija sa cloud firestore 15

5.4. Pokretanje aplikacije 17

6. KRETANJE KROZ ZASLONE 18

6.1. Zaslon za prijavu 18

6.2. Zaslon za registraciju 20

6.3. Nadzorna ploča **22**
6.3.1. Zasloni za dodavanje kolegija i rukovanje sa studentima 24
6.4. Zaslon detalja o kolegiju **27**
6.4.1. Zaslon za dodavanje predavanja 30
6.5. Zaslon detalja o predavanju **31**
7. ZAKLJUČAK **34**
LITERATURA **36**
SAŽETAK **37**
ABSTRACT **38**

1. UVOD

Praćenje prisutnosti studenata na predavanjima i vježbama od iznimne je važnosti u akademskom okruženju. Redovita nazočnost omogućuje studentima lakše praćenje nastavnog programa, dok profesorima pruža ključne informacije o sudjelovanju studenata u nastavnom procesu. Tradicionalni načini evidencije nazočnosti, poput ručnog bilježenja ili potpisa na listama, smatraju se nepraktičnima, vremenski zahtjevnima i podložnima pogreškama. S obzirom na ubrzani razvoj tehnologije, pojavila se potreba za automatiziranim sustavima koji unapređuju ovaj proces. Jedna od modernih tehnologija koja nudi rješenje za ovaj problem je RFID (*Radio Frequency Identification*) i NFC (*Near Field Communication*). RFID/NFC sustavi omogućuju brzu, sigurnu i učinkovitu evidenciju putem elektroničkih uređaja, pri čemu se eliminira potreba za ručnim unosom podataka i smanjuje mogućnost pogreške. Ova tehnologija, u kombinaciji s pametnim uređajima i mobilnim aplikacijama, predstavlja idealnu platformu za automatiziranu evidenciju nazočnosti studenata.

U ovom završnom radu istražena je mogućnost implementacije sustava za praćenje nazočnosti studenata pomoću RFID/NFC čitača, koji koristi studentske identifikacijske kartice (iksice). Fokus je stavljen na razvoj mobilne Android aplikacije koja služi kao sučelje za evidenciju, upravljanje podacima i analizu nazočnosti. Profesori pomoću aplikacije mogu pregledavati prisutnost u stvarnom vremenu, a podaci se automatski pohranjuju u bazu te se omogućuje njihova daljnja obrada i analiza. Uvođenjem ovakvog sustava, ne samo da se optimizira proces evidencije nazočnosti, već se otvara mogućnost šire primjene tehnologije u obrazovnom sustavu. Automatizacija i digitalizacija procesa poput ovog olakšavaju administrativne zadatke, štede vrijeme te omogućuju fokus na kvalitetu nastave i učenja. U radu se obrađuje tehnička implementacija, prednosti i potencijalni izazovi korištenja RFID/NFC tehnologije u obrazovnom okruženju te se nude prijedlozi za daljnja poboljšanja sustava.

Struktura ovog završnog rada sastoji se od nekoliko poglavlja.

U prvom poglavlju je ukratko predstavljena ideja završnog rada. U drugom poglavlju se istražuju slična rješenja koja imaju sličnu svrhu kao aplikacija ovog završnog rada. Treće poglavlje donosi pregled hardvera i softvera koji su korišteni u razvoju sustava, koji uključuje Dasduino

Connectplus ESP32 mikrokontroler, PN532 RFID/NFC modul, id kartica, Android studio, Kotlin te Firebase bazu podataka. Četvrto poglavlje detaljno objašnjava integraciju hardverskih komponenti. Peto poglavlje obrađuje arhitekturu aplikacije, upravljanje podacima, autentifikaciju korisnika, interakciju s Firestore bazom podataka te način pokretanja aplikacije.. Šesto poglavlje opisuje navigaciju kroz zaslone korisničkog sučelja aplikacije te funkcionalnosti i mogućnosti koje svaki zaslon pruža korisnicima. Zaključak sedmom poglavlju sažima ciljeve i postignuća rada na izradi mobilne aplikacije, ističe korištene moderne tehnologije i arhitekture te identificira izazove i potencijalne smjerove za daljnji razvoj aplikacije.

1.1. Zadatak završnog rada

Potrebno je napraviti mobilnu Android aplikaciju za evidenciju nazočnosti studenata u nastavi na temelju iksice. Aplikacija treba prikazivati ime, prezime, e-mail te ostale osnovne podatke o studentu. Potrebno je realizirati bazu podataka iz koje je moguće prikazati trenutni postotak nazočnosti. Potrebno je omogućiti izvoz baze u excel tablicu. Aplikaciju je potrebno testirati i dokumentirati rezultate testiranja.

2. ISTRAŽIVANJE POSTOJEĆIH RJEŠENJA

Prije implementacije vlastitog sustava, važno je analizirati postojeća rješenja koja imaju sličnu svrhu. Na tržištu postoje različite aplikacije i sustavi koji koriste RFID/NFC tehnologiju za praćenje prisutnosti ili identifikaciju, bilo u obrazovnim ustanovama ili u drugim sektorima. Ova analiza omogućava bolje razumijevanje postojećih pristupa, njihovih prednosti i nedostataka, što može pomoći u izgradnji učinkovitijeg i inovativnijeg rješenja za evidenciju nazočnosti studenata.

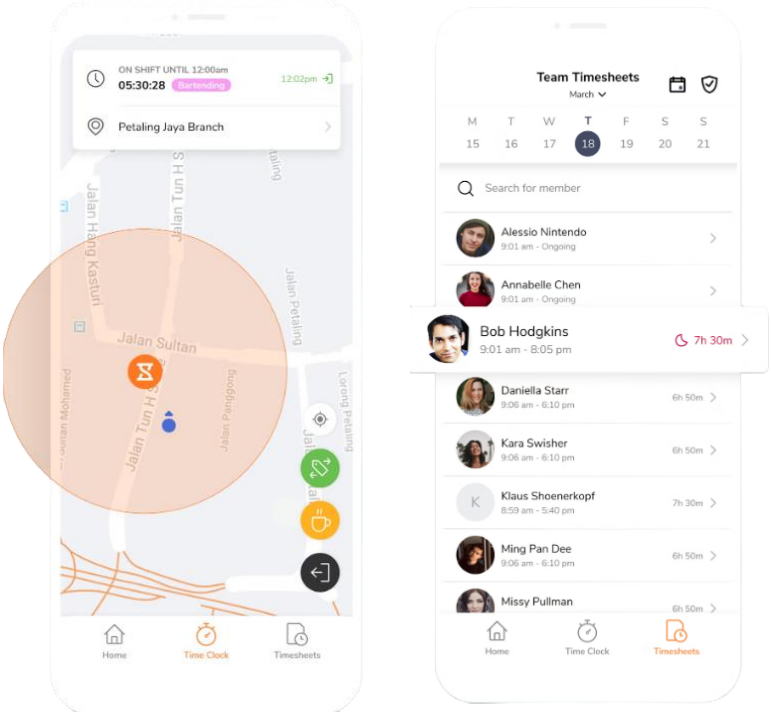
2.1. MyClass attendance

Attendy [1] je aplikacija za praćenje nazočnosti koja omogućava brzo i učinkovito upravljanje prisutnošću zaposlenika, studenata i osoblja putem interneta. S funkcijama za praćenje prisutnosti u stvarnom vremenu, vizualizaciju statistike po događajima ili članovima te izvoz podataka u PDF, CSV ili Excel format, Attendy olakšava vođenje evidencije bez potrebe za papirom. Podržano je online dijeljenje s višestrukim korisnicima i uređajima i omogućeno jednostavno dodavanje članova u grupe i nudi obavijesti kako bi se osiguralo da se evidencija ne zaboravi. Također pruža mogućnost praćenja prisutnosti putem kalendarskog načina rada i grafičkih prikaza za lakše praćenje i analizu podataka. Namijenjena je različitim korisnicima, uključujući učitelje, poslodavce i organizatore događaja te se može koristiti za praćenje prisutnosti u obrazovanju, poslovanju i drugim aktivnostima. Aplikacija je besplatna za preuzimanje na Android uređajima, a razvija je tvrtka Clever.Apps.

Slika 2.1. Korisničko sučelje aplikacije MyClass Attendance [1]

2.2. Attendance tracker by Jibble

Attendance tracker [2] predstavlja suvremeno rješenje za praćenje nazočnosti koje omogućava zaposlenicima i studentima prijavu i odjavu prisutnosti putem mobilnih uređaja ili računala. Aplikacija koristi GPS tehnologiju za provjeru lokacije prilikom prijave i odjave, što osigurava točnost podataka o nazočnosti, dok biometrijske metode poput prepoznavanja lica ili otiska prsta dodaju dodatni sloj sigurnosti. Uz to, Jibble nudi mogućnost generiranja detaljnih izvještaja i analize podataka o nazočnosti, omogućujući učinkovito praćenje i upravljanje prisutnošću u različitim okruženjima.



Slika 2.2. Korisničko sučelje aplikacije Attendance tracker [2]

3. KORIŠTENE TEHNOLOGIJE

U ovom poglavlju opisan je hardver i softver korišten za uspješnu izradu aplikacije.

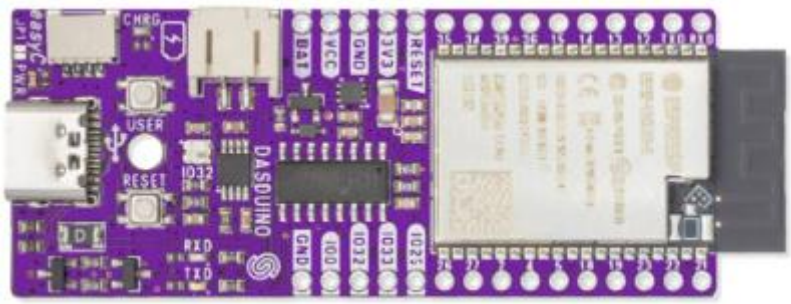
3.1. Hardverske komponente

Od hardverskih komponenti korišteni su:

1. Dasduino ESP32 Connectplus mikrokontroler,
2. EasyC adapter,
3. PN532 NFC RFID Module V4.

3.1.1. Dasduino ESP32 Connectplus

Dasduino CONNECTPLUS [3] je napredni mikrokontroler dizajniran za širok spektar aplikacija, od niskopotrošnih senzor mreža do zahtjevnih zadataka poput streaminga glazbe. Koristi moćni ESP32 kontroler s 4 MB flash memorije i 8 MB PSRAM-a. Mikrokontroler se pokazao izuzetno korisnim u kontekstu aplikacije nazočnosti s RFID/NFC čitačima zbog nekoliko ključnih karakteristika. Pouzdane i osigurane bežične veze omogućene su integriranom Wi-Fi i Bluetooth 4.2 tehnologijom, što je ključno za prijenos podataka s RFID/NFC čitača u stvarnom vremenu. Obrada i pohrana podataka s RFID/NFC čitača omogućene su ESP32 kontrolerom s velikim kapacitetom memorije uz minimalna kašnjenja. Razvoj i programiranje aplikacije pojednostavljeni su USB Type-C priključkom i Arduino IDE podrškom.



Slika 3.1. Izgled Dasduino CONNECTPLUS mikrokontrolera [3]

3.1.2. EasyC adapter

Povezivanje i komunikacija između različitih I2C uređaja, kao što je PN532 RFID/NFC čitač s mikrokontrolerom, omogućeni su EasyC [4] adapterom, čime je pojednostavljena integracija i smanjena potreba za kompleksnim povezivanjem žica.

Slika 3.2. Izgled EasyC adaptera [4]

3.1.3. PN532 NFC RFID Module V4

PN532 NFC RFID Module V4 [5] je popularan modul za rad s NFC (Near Field Communication) i RFID (Radio Frequency Identification) tehnologijama. Čitanje i pisanje RFID/NFC tagova te komunikacija s drugim NFC uređajima omogućeni su ovim modulom. Modul se smatra posebno korisnim u aplikacijama koje zahtijevaju interakciju s RFID/NFC tagovima, kao što su sustavi za evidenciju nazočnosti, te je korišten za skeniranje iksica studenata prilikom dolaska na predavanje.

Slika 3.3. Izgled PN532 NFC RFID Module V4 modula [5]

3.2. Softver

1. Od softvera korišteno je sljedeće:
2. Arduino IDE,
3. Google Firebase.

3.2.1. Arduino IDE

Arduino IDE [6] (Integrated Development Environment) je razvojno okruženje koje se koristi za programiranje mikrokontrolera poput Arduino ploča i drugih sličnih uređaja, kao što je Dasduino ESP32 Connectplus. Ovaj alat omogućuje pisanje, kompilaciju i prijenos programskog koda, koristeći jezik temeljen na C i C++, što olakšava korištenje dobro poznatih sintaktičkih pravila i struktura. Arduino IDE dolazi s bogatim skupom ugrađenih knjižnica koje proširuju funkcionalnosti mikrokontrolera, omogućujući jednostavnu integraciju s različitim senzorima i modulima, uključujući RFID/NFC čitače. Nadalje, njegova podrška za serijsku komunikaciju putem UART protokola olakšava prijenos podataka između mikrokontrolera i vanjskih uređaja, dok serijski monitor unutar IDE-a omogućuje praćenje komunikacije u stvarnom vremenu, što je ključno za dijagnostiku i testiranje. Zahvaljujući integriranom bootloaderu, prijenos koda je pojednostavljen jer nije potrebna vanjska hardverska podrška, što čini Arduino IDE idealnim izborom za brzo prototipiranje i razvoj aplikacija u obrazovnom, istraživačkom i industrijskom okruženju.

3.2.2. Firebase

Firebase [7] je razvojna platforma za mobilne i web aplikacije koju je razvio Google, pružajući niz usluga i alata koji olakšavaju brzo stvaranje i skaliranje aplikacija. U kontekstu aplikacije koja je izrađivana za potrebe završnog rada korišteni su firebase authentication i firestore database. Cloud Firestore je napredna NoSQL baza dizajnirana za rad u stvarnom vremenu, omogućuje pohranu podataka u dokumentima unutar kolekcija, što pruža fleksibilnu i skalabilnu strukturu podataka. Ova baza podržava automatsku sinkronizaciju u stvarnom vremenu, omogućujući trenutne promjene u aplikacijama bez potrebe za ručnim osvježavanjem. Uz mogućnost složenih upita, filtriranja i sortiranja podataka, Firestore također nudi podršku za offline rad, što omogućuje korisnicima da nastave koristiti aplikaciju i pristupaju podacima čak i

bez aktivne mrežne veze. Dodatno, integrira se s Firebase Authentication za upravljanje pristupom i sigurnošću podataka, čime pruža robusno rješenje za aplikacije koje zahtijevaju visoku dostupnost i brzu reakciju na promjene podataka. Firebase Authentication je usluga unutar Firebase platforme koja omogućuje jednostavno upravljanje autentifikacijomkorisnika. Podržava više metoda prijave, uključujući prijavu putem e-maila i lozinke, telefonskog broja, kao i popularne prijave putem društvenih mreža poput Googlea, Facebooka i Twittera. Ova usluga pojednostavljuje proces autentifikacije, jer nudi sigurnosno robusne metode za potvrdu identiteta korisnika, a sve se to odvija putem jednostavnog sučelja koje je lako integrirati u aplikaciju. Firebase Također omogućuje jednostavno upravljanje sesijama, sinkronizaciju korisničkih podataka te se može kombinirati s drugim Firebase alatima poput Firestore-a za dodatnu sigurnost i personalizaciju korisničkog iskustva.

3.2.3. Android Studio i Kotlin

Android Studio [8] je službeno razvojno okruženje (IDE) za izradu aplikacija za Android platformu, koje pruža sveobuhvatne alate za pisanje, testiranje, debugiranje i optimizaciju aplikacija. U kombinaciji s Kotlinom, modernim programskim jezikom koji je službeno podržan za razvoj Android aplikacija, Android Studio omogućuje bržu i učinkovitiju izradu mobilnih aplikacija.

Kotlin [9] nudi brojne prednosti u odnosu na tradicionalni Java jezik, uključujući jednostavniju sintaksu, sigurnost od null pointera, bolju podršku za funkcionalno programiranje te smanjenje broja redaka koda, čime se smanjuju i potencijalne greške. Posebno je važna integracija Jetpack Compose [10] tehnologije, koja koristi composable funkcije za izgradnju korisničkog sučelja na deklarativan način. Ove funkcije omogućuju stvaranje fleksibilnih, responzivnih i dinamičkih korisničkih sučelja s manje koda i kompleksnosti, što znatno ubrzava proces razvoja i prilagodbe aplikacije različitim uređajima. Kombinacija Android Studija i Kotlina s podrškom za composable funkcije predstavlja optimalno rješenje za razvoj naprednih, održivih i visokokvalitetnih Android aplikacija.

4. PROGRAMIRANJE HARDVERSKIH KOMPONENTI

Kompletan sklop koji se sastoji od Dasduino ESP32 Connectplus mikrokontrolera povezanog s PN532 NFC čitačem preko EasyC adaptera koji omogućuje I2C komunikaciju. Komunikacija između sklopa i računala je ostvarena uz pomoć USB kabela koji je potreban za napajanje samog sklopa kao i učitavanje koda.

Slika 4.1. Izgled povezanog uređaja

4.1. Implementacija Arduino koda

Ovaj kod omogućuje Dasduino ESP32 Connectplus mikrokontroleru da očitava podatke s NFC kartica koristeći PN532 modul i šalje te podatke preko Bluetooth veze na drugi uređaj, koristeći I2C protokol za komunikaciju između mikrokontrolera i NFC modula.

4.2. Uključivanje biblioteka i definiranje pinova za komunikaciju

U ovom dijelu koda (Slika 4.2.) se uključuju biblioteke i definiraju se pinovi potrebni za rad komponenti:

- *Wire.h* je biblioteka za I2C komunikaciju, koja omogućuje mikrokontroleru da komunicira s I2C uređajima poput PN532 NFC modula.

- *Adafruit_PN532.h* je biblioteka specifičnu za PN532 modul koju pruža Adafruit. Ova knjižnica olakšava rad s NFC/RFID funkcionalnostima modula.
- *BluetoothSerial.h* je biblioteka za Bluetooth komunikaciju, koja omogućuje mikrokontroleru Dasduino ESP32 komunikaciju s drugim uređajima putem Bluetootha.
- *SDA_PIN 21* i *SCL_PIN 22* su pinovi na mikrokontroleru koji se koriste za I2C komunikaciju s PN532 modulom. Pin 21 se koristi za SDA (Serial Data Line), a pin 22 za SCL (Serial Clock Line). Ovi pinovi omogućuju prijenos podataka između mikrokontrolera i NFC modula.

```
1   #include <Wire.h>
2   #include <Adafruit_PN532.h>
3   #include "BluetoothSerial.h"
4
5   // Pins for PN532
6   #define SDA_PIN 21
7   #define SCL_PIN 22
8
```

Slika 4.2. Uključivanje biblioteka i definiranje pinova

4.3. Inicijalizacija NDC i Bluetooth objekata

- *Adafruit_PN532 nfc(SDA_PIN, SCL_PIN)* kreira objekt nfc koristeći klasu Adafruit_PN532 i definira pinove SDA i SCL za I2C komunikaciju. Ovaj objekt se koristi za upravljanje komunikacijom s PN532 NFC modulom.
- *BluetoothSerial SerialBT* kreira objekt SerialBT koristeći klasu BluetoothSerial kako bi omogućio Bluetooth komunikaciju između mikrokontrolera i drugih Bluetooth uređaja.

```
8
9   Adafruit_PN532 nfc(SDA_PIN, SCL_PIN);
10  BluetoothSerial SerialBT;
11
```

Slika 4.3. Inicijalizacija NFC i Bluetooth objekata

4.4. Setup funkcija

Prilikom pokretanja uređaja izvodi se setup() funkcija(Slika 4.4.).

- Na početku, poziva se *Serial.begin(115200)* kako bi se serijska komunikacija uspostavila pri brzini od 115200 bita u sekundi, omogućujući ispisivanje podataka u serijski monitor, što pomaže u praćenju stanja sustava i dijagnostici potencijalnih problema.
- Nakon toga, započinje Bluetooth komunikacija pomoću *SerialBT.begin("ESP32_NFC")*, gdje se naziv "ESP32_NFC" koristi kao identifikacija uređaja na Bluetooth mreži.
- Inicijalizacija NFC čitača vrši se pomoću *nfc.begin()*, koja priprema PN532 modul za rad. Zatim se pomoću *nfc.getFirmwareVersion()* provjerava prisutnost NFC modula, a ako nije pronađen, ispisuje se poruka o grešci i program se zaustavlja.
- Ako je modul uspješno detektiran, konfigurira se pomoću *nfc.SAMConfig()*, što omogućuje postavljanje NFC modula u način rada za otkrivanje pasivnih NFC tagova, nakon čega uređaj prelazi u stanje čekanja na očitavanje NFC kartice.

```
12  void setup() {
13    Serial.begin(115200);
14    SerialBT.begin("ESP32_NFC"); // Name of your Bluetooth device
15    nfc.begin();
16
17    uint32_t versiondata = nfc.getFirmwareVersion();
18    if (!versiondata) {
19      Serial.println("Didn't find PN53x board");
20      while (1);
21    }
22
23    nfc.SAMConfig();
24    Serial.println("Waiting for an NFC card...");
25  }
26
```

Slika 4.4. Funkcija *setup()*

4.5. Loop funkcija

Funkcija *loop()* (Slika 4.5.) se kontinuirano izvršava nakon inicijalne postavke sustava i služi za neprestano očitavanje NFC kartica i slanje njihovih podataka putem Bluetootha.

- Na početku se definiraju varijable potrebne za očitavanje NFC kartice: *success* za praćenje statusa uspješnosti očitavanja, *uid[]* za pohranu jedinstvenog identifikatora (UID) očitane kartice i *uidLength* za pohranu duljine tog UID-a.
- Funkcija *nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid, &uidLength)* pokušava očitati NFC karticu koristeći MIFARE ISO14443A standard. Ako očitavanje

uspije, success varijabla dobiva vrijednost true (1), a polje *uid* se popunjava identifikatorom kartice, dok *uidLength* pohranjuje duljinu očitanog UID-a.

- Ako je očitavanje bilo uspješno (success je true), funkcija ispisuje poruku "RFID card detected!" na serijski monitor, a zatim prolazi kroz svaki bajt UID-a, konvertira ga u heksadecimalni oblik i dodaje u uidStr string.
- Zatim se skenirani UID kartice ispisuje na serijski monitor uz poruku "Scanned RFID UID: ", omogućujući korisniku da vidi jedinstveni identifikator kartice.
- UID kartice se također šalje na drugi uređaj putem Bluetootha koristeći *SerialBT.println(uidStr)*, čime se omogućuje daljinska obrada ili evidencija prisutnosti.
- Na kraju, funkcija uvodi pauzu od 1 sekunde (*delay(1000)*) kako bi se spriječilo prebrzo ponovno očitavanje iste kartice, osiguravajući stabilan rad sustava i izbjegavajući višestruka skeniranja u kratkom vremenskom razdoblju.

```
27  void loop() {
28    uint8_t success;
29    uint8_t uid[] = {0, 0, 0, 0, 0, 0, 0};
30    uint8_t uidLength;
31
32    success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid, &uidLength);
33
34    if (success) {
35      Serial.println("RFID card detected!");
36      String uidStr = "";
37      for (uint8_t i = 0; i < uidLength; i++) {
38        uidStr += String(uid[i], HEX);
39      }
40
41      // Print UID to the console
42      Serial.print("Scanned RFID UID: ");
43      Serial.println(uidStr);
44
45      SerialBT.println(uidStr);  // Send UID over Bluetooth
46      delay(1000);  // Delay to prevent rapid rescan
47    }
48  }
```

Slika 4.5. Upravljanje NFC komunikacijom

5. RAZVOJ APLIKACIJE I PRIKAZ FUNKCIONALNOSTI

Aplikacija je razvijena korištenjem MVVM (*Model-View-ViewModel*) arhitekturnog uzorka, čime je omogućeno jasno razdvajanje poslovne logike od korisničkog sučelja te olakšano održavanje koda, poboljšana skalabilnost i ubrzana prilagodba aplikacije. U ovom projektu, MVVM arhitektura očituje se kroz tri glavne komponente:

1. Model sloj predstavlja strukturu i upravljanje podacima unutar aplikacije. U ovom projektu, ovaj sloj je definiran klasama poput Student.kt, Subject.kt, Lecture.kt, Attendance.kt i User.kt, koje se nalaze u mapi data/model. Ove klase definiraju strukturu podataka koja se koristi za upravljanje informacijama o studentima, predmetima, predavanjima i evidencijama prisutnosti. Model sloj je odgovoran za pristup podacima iz različitih izvora, kao što su lokalne baze podataka ili udaljeni serveri i za njihovu obradu.

2. View sloj obuhvaća korisničko sučelje aplikacije, izrađeno pomoću XML datoteka i Jetpack Compose funkcija, gdje se definiraju izgled aplikacije i način interakcije s korisnicima. U projektu se ovaj sloj ostvaruje pomoću datoteka kao što su activity_main.xml i drugih layout datoteka koje definiraju izgled specifičnih zaslona aplikacije. Korištenje Jetpack Compose omogućuje deklarativan pristup dizajniranju sučelja, čime se pojednostavljuje razvoj i održavanje korisničkog iskustva.

3. ViewModel sloj služi kao posrednik između Modela i Viewa, omogućujući prijenos podataka i upravljanje korisničkim događajima te interakcijama. Iako nije izričito nazvana "ViewModel", logika ovog sloja može se pronaći u datotekama poput AuthenticationProvider.kt, koja upravlja procesom autentifikacije korisnika putem Firebase-a. Ova klasa preuzima ulogu posrednika između korisničkog sučelja i podataka, osiguravajući da podaci ostanu sinkronizirani i ažurirani, bez potrebe da View izravno pristupa Modelu.

5.1. Upravljanje podatcima

Upravljanje podacima realizirano je korištenjem posebnih klasa koje su definirane unutar model sloja. Ove klase koriste se za strukturiranje i pohranu podataka te za olakšavanje njihove obrade, manipulacije i prikaza kroz različite dijelove aplikacije. Svaka klasa modela predstavlja određeni entitet u aplikaciji, kao što su studenti, predmeti, predavanja, evidencija prisutnosti i korisnici. Klasa *Student.kt* koristi se za predstavljanje studenta s atributima kao što su ime,

prezime, e-mail adresa i jedinstveni ID. Omogućuje pohranu i pristup informacijama o studentima te njihovu povezanost s evidencijom prisutnosti i predavanjima. *Klasa Subject.kt* koristi se za definiranje predmeta kroz atribute poput naziva, šifre predmeta i imena predavača, čime se omogućuje upravljanje informacijama o predmetima i njihovo povezivanje s predavanjima. U *Lecture.kt* sadržani su podaci o predavanjima, uključujući datum, vrijeme, predmet i popis sudionika, čime je omogućeno praćenje događaja i točnu evidenciju prisutnosti. Klasa *Attendance.kt* koristi se za evidentiranje prisutnosti studenata na predavanjima, bilježeći identifikatore studenata i predavanja te status prisutnosti (prisutan, odsutan, opravdan izostanak), čime su pružene ključne informacije za analizu. Klasa *User.kt* koristi se za definiranje korisnika aplikacije s atributima kao što su uloga (student, predavač, administrator), e-mail, lozinka i druge informacije potrebne za autentifikaciju i autorizaciju putem Firebase Authentication. *StudentSubject.kt* predstavlja relaciju između studenta i predmeta unutar aplikacije. Ovom klasom povezuje se specifičan student s određenim predmetom, uključujući atribute kao što su jedinstveni identifikatori studenta i predmeta (*studentId* i *subjectId*), ime i prezime studenta (*studentName* i *studentSurname*), broj kartice studenta (*cardUid*) te naziv predmeta (*subjectName*). Korištenjem ove klase, aplikacija može efikasno upravljati i pratiti koji su studenti upisani na koje predmete, olakšavajući organizaciju akademskih podataka i povezivanje s drugim entitetima, poput evidencije prisutnosti i predavanja.

```
1  package com.example.evidencijastudenata.data.model
2  data class User(
3      val id: String = "",
4      val displayName: String = "",
5      val name: String = "",
6      val surname: String = "",
7      val email: String = "",
8      val password: String = ""
9  )
```

Slika 5.1. Izgled podatkovne klase *User*

5.2. Upravljanje autentifikacjiom korisnika

Klasa *AuthenticationProvider* (Slika 5.2.) implementirana je kao Composable funkcija koja upravlja autentifikacijskim stanjem korisnika unutar aplikacije. Funkcija koristi Firebase Authentication servis kako bi se provjerilo je li korisnik prijavljen te kako bi se na temelju toga kontrolirala navigacija unutar aplikacije i omogućio pristup samo autoriziranim korisnicima.

Inicijalizacija Firebase Authentication klijenta započinje pomoću metode *FirebaseAuth.getInstance()*, čime je omogućen dohvat trenutnog stanja autentifikacije korisnika. Korištenjem Jetpack Compose funkcija *remember* i *derivedStateOf*, prati se stanje autentifikacije kroz varijablu *isAuthenticated*. Ako je *auth.currentUser* različit od null, smatra se da je korisnik prijavljen. Za dinamičko upravljanje stanjem autentifikacije koristi se Compose efekt *LaunchedEffect*. Ovaj efekt se aktivira svaki put kada se promijeni stanje autentifikacije te provjerava je li korisnik prijavljen. Ako korisnik nije prijavljen (*isAuthenticated* je false), korisnik se preusmjerava na zaslon za prijavu korištenjem *navController.navigate(route = 'login')*. Kako bi se spriječilo vraćanje na prethodne zaslone prije prijave, koristi se metoda *popUpTo*, koja uklanja sve prethodne destinacije iz navigacijskog grafa.

```
@Composable
fun AuthenticationProvider(navController: NavController, content: @Composable () -> Unit) {
    val auth = FirebaseAuth.getInstance()
    val isAuthenticated by remember { derivedStateOf { auth.currentUser != null } }

    LaunchedEffect(isAuthenticated) {
        if (!isAuthenticated) {
            navController.navigate( route: "login") {
                popUpTo(navController.graph.findStartDestination().id) { inclusive = true }
            }
        }
    }

    if (isAuthenticated) {
        content()
    }
}
```

Slika 5.2. *AuthenticationProvider* funkcija

5.3. Interakcija sa cloud firestore

U ovoj aplikaciji, repository sloj služi kao posrednik između modela podataka i vanjskih izvora podataka, poput Cloud Firestore baze podataka. Svaka od pet repository klasa specijalizirana je za upravljanje određenim skupom podataka, omogućujući centralizirano upravljanje pristupom podacima te olakšavajući održavanje i prilagodbu koda. U ovom poglavlju prikazan je pregled svih repository klasa korištenih u aplikaciji te njihovih funkcija i odgovornosti u kontekstu interakcije s Firestore bazom podataka. Sve repository klase sadrže standardne funkcije koje omogućuju osnovne CRUD operacije (*Create, Read, Update, Delete*) nad podacima, kao i specifične funkcije prilagođene pojedinim potrebama. *UserRepository* klasom upravlja se

podacima o korisnicima unutar aplikacije. Ključne funkcije uključuju *getUserById,* kojom se dohvaća korisnik iz Firestore baze podataka prema jedinstvenom ID-u te *addUser*, kojom se dodaje novi korisnik u bazu. Obje funkcije koriste suspendirane korutine za asinkrono izvršavanje mrežnih operacija. Podacima o predmetima u aplikaciji upravlja se kroz *SubjectRepository*. Ključne funkcije uključuju *addSubject,* kojom se dodaje novi predmet u Firestore bazu podataka i *getSubjectById,* kojom se dohvaćaju detalji predmeta prema jedinstvenom ID-u. Funkcijom *getSubjects* dohvaćaju se svi predmeti povezani s određenim korisnikom na temelju njegovog ID-a, dok se funkcijom *deleteSubject* briše predmet i sve povezane zapise predavanja i prisutnosti. Sve funkcije osiguravaju konzistentnost podataka u bazi i rukovanje greškama. Podacima o studentima u aplikaciji, uključujući dodavanje, dohvaćanje i brisanje informacija, upravlja se kroz *StudentRepository*. Ključne funkcije uključuju *addStudent,* kojom se dodaje novi student i *getStudentById,* kojom se dohvaća studenta prema ID-u. Dodatne funkcije su *getStudentByCardUid,* kojom se dohvaća student na temelju UID-a kartice te getStudents, kojom se dohvaćaju svi studenti povezani s određenim profesorom. Funkcijom *addStudents* omogućeno je dodavanje više studenata odjednom, dok se *deleteStudent* funkcijom briše studenta i sve povezane evidencije prisutnosti. *LectureRepository* korišten je za upravljanje podacima o predavanjima, uključujući dodavanje novih i brisanje postojećih predavanja. Ključne funkcije uključuju *addLecture,* kojom se dodaje novo predavanje i *getLectureDetails,* kojom se dohvaćaju detalji predavanja prema ID-u. Funkcijom *getLecturesBySubject* dohvaćaju se sva predavanja povezana s određenim predmetom, dok se *deleteLecture* funkcijom briše predavanje i sve povezane evidencije prisutnosti. Podacima o evidenciji prisutnosti studenata na predavanjima upravljano je kroz *AttendanceRepository*. Ključne funkcije uključuju *logAttendance,* kojom se dodaje novi zapis prisutnosti te *getAttendancesByLecture,* kojom se dohvaćaju svi zapisi prisutnosti za određeno predavanje. Funkcijom *getAttendancesByStudent* dohvaćaju se svi zapisi prisutnosti za određenog studenta. Sve funkcije osiguravaju stabilnost i dosljednost aplikacije kroz učinkovito rukovanje greškama. Podacima koji su veza studenata s predmetima unutar aplikacije upravljano je kroz *StudentSubjectRepository*. Ključne funkcije uključuju *addStudentSubject,* kojom se dodaje novi zapis o povezanosti studenta s predmetom te *removeStudentSubject,* kojom se uklanja zapis o povezanosti određenog studenta s predmetom, istovremeno ažurirajući broj studenata na predmetu u Firestore bazi podataka. Funkcijom *getStudentSubjectsBySubjectId* dohvaćaju se svi studenti povezani s određenim predmetom, dok se uz pomoć *syncStudentSubjects* osigurava sinkronizacija postojećih zapisa s najnovijim podacima iz baze, dodajući ili uklanjajući studente ovisno o trenutnoj povezanosti s predmetom.

5.4. Pokretanje aplikacije

Prilikom pokretanja aplikacije, prva komponenta koja se pokreće je *MainActivity*. Ova aktivnost djeluje kao početna točka aplikacije i definira se u datoteci *AndroidManifest.xml*. Unutar *MainActivity*, metoda onCreate se poziva pri inicijalizaciji aktivnosti. U ovom slučaju, metoda *onCreate* korištena je za postavljanje sadržaja aplikacije pomoću Jetpack Compose i setContent bloka, unutar kojeg je pozvana funkcija *CheckPointApp*.

Funkcija *CheckPointApp* (Slika 5.3.) uspostavlja navigacijski sustav aplikacije koristeći *NavHost* i *NavController* za upravljanje prijelazima između različitih zaslona. Početni zaslon je "login", a definirane su rute za prijavu, registraciju, dashboard, detalje predmeta i predavanja, dodavanje predmeta i predavanja te lista i dodavanje studenata.

```
@Composable
fun CheckPointApp() {
    val navController = rememberNavController()
    CheckPointNavHost(navController = navController)
}

@Composable
fun CheckPointNavHost(navController: NavHostController) {

    NavHost(navController = navController, startDestination = "login") {
        composable( route: "login") { LoginScreen(navController) }
        composable( route: "register") { RegisterScreen(navController) }
        composable( route: "dashboard") {
            AuthenticationProvider(navController) {
                DashboardScreen(navController)
            }
        }
        composable( route: "subject_detail/{subjectId}") { backStackEntry ->
            val subjectId = backStackEntry.arguments?.getString( key: "subjectId")
            subjectId?.let { SubjectDetailScreen(navController, it) }
        }
        composable( route: "lecture_detail/{lectureId}") { backStackEntry ->
            val lectureId = backStackEntry.arguments?.getString( key: "lectureId")
            lectureId?.let { LectureDetailScreen(navController, it) }
        }
        composable( route: "add_subject") { AddSubjectScreen(navController) }
        composable( route: "add_lecture/{subjectId}") { backStackEntry ->
            val subjectId = backStackEntry.arguments?.getString( key: "subjectId")
            subjectId?.let { AddLectureScreen(navController, it) }
        }


        composable( route: "student_list") {
            StudentListScreen(navController, profesorId = FirebaseAuth.getInstance().currentUser?.uid ?: "")
        }
        composable( route: "add_student") {
            AddStudentScreen(navController = navController, profesorId = FirebaseAuth.getInstance().currentUser?.uid ?: "")
        }
    }
}
```

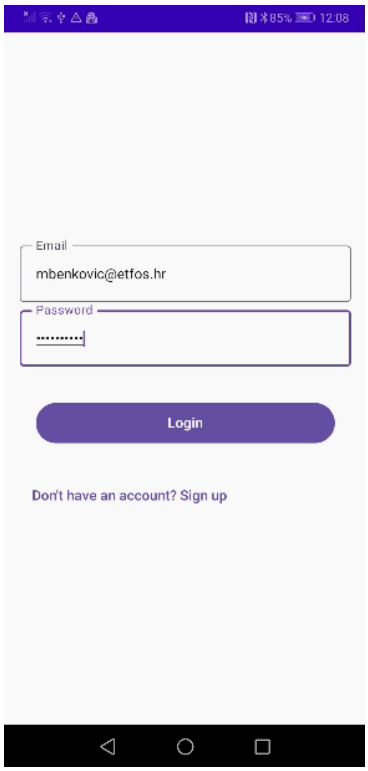
Slika 5.3. *CheckPointApp* funkcija

6. KRETANJE KROZ ZASLONE

U ovom dijelu završnog rada navode se dostupne opcije za kretanje zaslonima korisničkog sučelja te se opisuje način odabira i izmjene stavki prikazanih na zaslonu.

6.1. Zaslon za prijavu

Login ekran (Slika 6.1.) služi kao ulazna točka u aplikaciju, omogućavajući korisnicima da se prijave korištenjem e-mail adrese i lozinke. Implementiran je korištenjem Jetpack Compose komponenti i integriran s Firebase Authentication servisom kako bi se osigurala sigurna autentifikacija korisnika. Na vrhu ekrana nalaze se dvije prilagođene komponente za unos podataka (Slika 6.2.), *AuthTextField*, koje omogućuju unos e-mail adrese i lozinke. Polje za lozinku ima postavljenu opciju skrivanja unosa (*isPassword = true*) radi sigurnosti. Nakon unosa podataka, pritiskom na gumb *AuthButton* pokreće se funkcija *loginWithFirebase,* koja se povezuje s Firebase Authentication servisom i provjerava ispravnost unesenih podataka. U slučaju uspješne prijave, korisnik se preusmjerava na "dashboard" ekran (DashboardScreen), dok u slučaju neuspjeha funkcija *onFailure* prikazuje poruku o pogrešci koristeći *Toast* komponentu i vraća stanje aplikacije na početno (Idle). Funkcija *loginWithFirebase* koristi asinkroni poziv prema Firebase-u za provjeru vjerodajnica i sigurno upravljanje prijavom. Na dnu ekrana nalazi se i *TextButton* koji omogućuje navigaciju na ekran za registraciju (*RegisterScreen*), čime se korisnicima koji još nemaju račun omogućava brz i jednostavan način za stvaranje novog korisničkog računa.



Slika 6.1. Izgled zaslona za prijavu

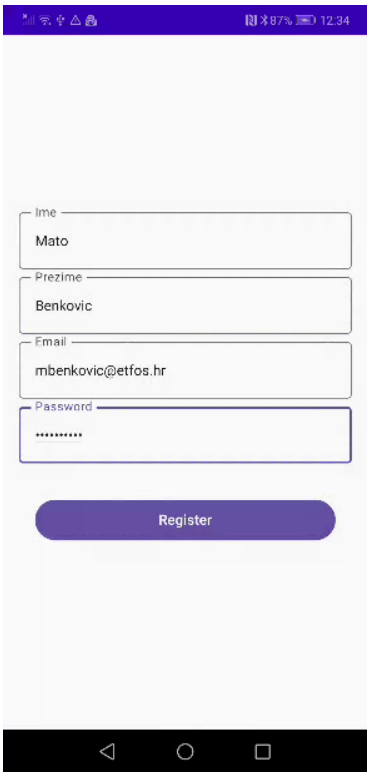
Cijeli proces prijave prati se kroz definirana stanja (*LoginState*), poput "učitavanja" (*Loading*) i "pogreške" (*Failure*), osiguravajući da korisnik ima jasno razumijevanje o tijeku prijave u stvarnom vremenu. Time se osigurava korisničko iskustvo koje je intuitivno, responzivno i sigurno.

```
@Composable
fun AuthTextField(
    valueState: MutableState<String>,
    label: String,
    isPassword: Boolean = false,
    modifier: Modifier = Modifier
) {
    OutlinedTextField(
        value = valueState.value,
        onValueChange = { valueState.value = it },
        label = { Text(label) },
        modifier = modifier.fillMaxWidth(),
        singleLine = true,
        visualTransformation = if (isPassword) PasswordVisualTransformation() else VisualTransformation.None,
        keyboardOptions = KeyboardOptions.Default.copy(
            imeAction = ImeAction.Done
        )
    )
}

@Composable
fun AuthButton(onClick: () -> Unit, text: String, modifier: Modifier = Modifier) {
    Button(
        onClick = onClick,
        modifier = modifier.fillMaxWidth().padding(16.dp)
    ) {
        Text(text)
    }
}
```

Slika 6.2. *AuthTextField* i *AuthButton* funkcije

6.2. Zaslon za registraciju

Register zaslon (Slika 6.3.) omogućuje korisnicima kreiranje novog korisničkog računa u aplikaciji putem e-mail adrese i lozinke. Korištenjem Jetpack Compose komponenti omogućena je izrada korisničkog sučelja sa četiri prilagođena tekstualna polja (*AuthTextField*) za unos imena, prezimena, e-maila i lozinke. Nakon unosa potrebnih podataka, korisnik može pritisnuti gumb za registraciju (*AuthButton*), koji pokreće proces registracije putem Firebase Authentication servisa.



Slika 6.3. Izgled zaslona za registraciju

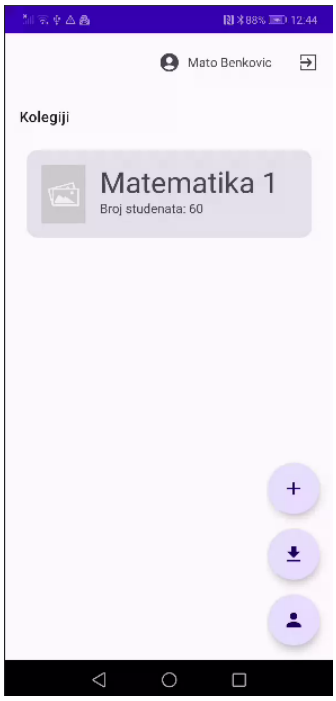
Funkcionalnost ekrana je osigurana funkcijom *registerWithFirebase* (Slika 6.4.), koja koristi Firebase Authentication servis za kreiranje novog korisničkog računa koristeći e-mail i lozinku. Nakon uspješne registracije, podaci o korisniku se dodatno pohranjuju u Firestore bazu podataka putem *UserRepository* klase, koristeći suspendiranu korutinu (*coroutineScope.launch*). Ako je registracija uspješna, korisnik se preusmjerava na početni ekran ("dashboard"), dok se u slučaju pogreške prikazuje odgovarajuća poruka o grešci koristeći stanje *RegisterState*. Zaslon također koristi stanja (*RegisterState*) kako bi prikazao različite faze procesa registracije - "učitavanje" (*Loading*) kada je registracija u tijeku te "pogrešku" (*Failure*) ako dođe do problema, osiguravajući korisnicima jasan pregled svakog koraka u procesu.

```
private fun registerWithFirebase(
    auth: FirebaseAuth,
    email: String,
    password: String,
    onSuccess: () -> Unit,
    onFailure: (String) -> Unit
) {
    auth.createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener { task ->
            if (task.isSuccessful) {
                onSuccess()
            } else {
                onFailure(task.exception?.localizedMessage ?: "Unknown error occurred")
            }
        }
}

sealed class RegisterState {
    object Idle : RegisterState()
    object Loading : RegisterState()
    data class Failure(val errorMessage: String) : RegisterState()
}
```

Slika 6.4. Funkcija *registerWithFirebase*

6.3. Nadzorna ploča

Dashboard ekran (Slika 6.5.) je centralno mjesto u aplikaciji koje omogućuje korisnicima pregled i upravljanje predmetima, studentima te izvođenje dodatnih funkcija poput dodavanja novih predmeta i izvoza podataka. Ovaj ekran prikazuje sve kolegije korisnika, omogućuje brisanje postojećih kolegija, dodavanje novih te izvozi popis predmeta u Excel datoteku. Na vrhu ekrana nalazi se *TopAppBar* s informacijama o trenutnom korisniku i gumbom za odjavu. Odabirom ikone za odjavu, korisnik se odjavljuje iz aplikacije i vraća se na zaslon za prijavu. *LazyColumn* komponenta koristi se za prikazivanje liste kolegija koje korisnik ima. Svaki kolegij predstavljen je pomoću *SubjectCard* komponente koja prikazuje naziv kolegija i broj studenata. Klikom na pojedini kolegij, korisnik može pristupiti detaljima kolegija, dok dugi pritisak na karticu omogućuje brisanje kolegija uz prethodnu potvrdu putem dijaloškog okvira (*AlertDialog*). Na dnu ekrana nalazi se skup plutajućih akcijskih gumba (*FloatingActionButton*) koji korisnicima omogućuju dodavanje novog kolegija, pregled liste studenata ili izvoz podataka u Excel format.



Slika 6.5. Izgled nadzorne ploče

Funkcionalnost izvoza u Excel koristi *exportSubjectsToExcel* (Slika 6.6.) funkciju, koja traži dozvolu za pisanje u vanjsku pohranu pomoću *permissionLauncher*, a zatim generira i pohranjuje Excel datoteku s podacima o kolegijima koristeći *Apache POI* biblioteku. Ova funkcija osigurava da se operacije izvoza izvode asinkrono, bez blokiranja glavnog korisničkog sučelja. Ekran također koristi *LaunchedEffect* za učitavanje podataka o kolegijima i korisniku prilikom inicijalizacije, čime se osigurava da su informacije uvijek aktualne. Tijekom bilo kakve interakcije koja zahtijeva dulje trajanje, poput izvoza ili brisanja kolegija, prikazuje se *CircularProgressIndicator* kako bi korisnici bili svjesni da je operacija u tijeku.

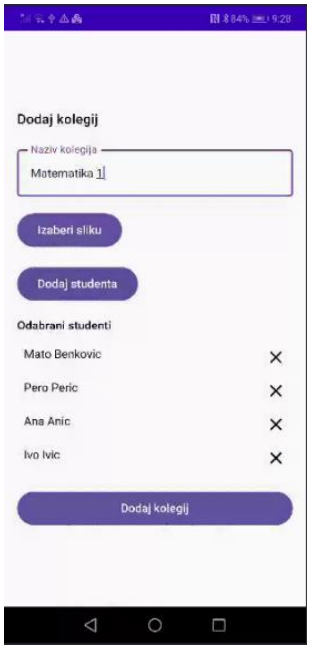
```
fun exportSubjectsToExcel(context: Context, subjects: List<Subject>, onCompletion: () -> Unit) {
    CoroutineScope(Dispatchers.IO).launch {
        try {
            val workbook = XSSFWorkbook()
            val sheet = workbook.createSheet( sheetname: "Kolegiji")

            val headerRow = sheet.createRow( rownum: 0)
            headerRow.createCell( columnIndex: 0).setCellValue("#")
            headerRow.createCell( columnIndex: 1).setCellValue("Naziv kolegija")
            headerRow.createCell( columnIndex: 2).setCellValue("Broj studenata")

            subjects.forEachIndexed { index, subject ->
                val row = sheet.createRow( rownum: index + 1)
                row.createCell( columnIndex: 0).setCellValue((index + 1).toDouble())
                row.createCell( columnIndex: 1).setCellValue(subject.name)
                row.createCell( columnIndex: 2).setCellValue(subject.numberOfStudents.toDouble())
            }

            val downloadsDir = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS)
            val fileName = "Kolegiji_${System.currentTimeMillis()}.xlsx"
            val file = File(downloadsDir, fileName)
            val outputStream = FileOutputStream(file)
            workbook.write(outputStream)
            outputStream.close()
            workbook.close()

            withContext(Dispatchers.Main) {
                Toast.makeText(context,  text: "Dokument exportiran: ${file.absolutePath}", Toast.LENGTH_LONG).show()
                onCompletion()
            }
        } catch (e: Exception) {
            withContext(Dispatchers.Main) {
                Toast.makeText(context,  text: "Greška pri izvozu dokumenta: ${e.message}", Toast.LENGTH_LONG).show()
                onCompletion()
            }
        }
    }
}
```

Slika 6.6. Funkcija za izvoz u excel

6.3.1. Zasloni za dodavanje kolegija i rukovanje sa studentima

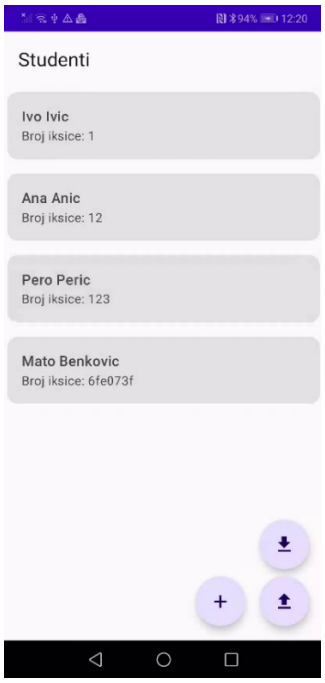
Zaslon za dodavanje kolegija (Slika 6.7.) omogućuje korisnicima kreiranje novog kolegija u aplikaciji, uključujući unos potrebnih informacija o kolegiju, dodavanje slike i povezivanje studenata s kolegijem. Ovaj zaslon koristi različite komponente za unos podataka, odabir i pregled informacija te pruža korisnicima intuitivan način za upravljanje kolegijima. Na početku zaslona nalazi se polje za unos naziva kolegija (*OutlinedTextField*) koje je označeno kao obavezno. Ako korisnik pokuša dodati kolegij bez unosa naziva, prikazuje se poruka o pogrešci koja naglašava da je polje obavezno. Zaslon, također, omogućuje korisnicima dodavanje slike kolegija pomoću gumba "Izaberi sliku". Nakon odabira slike, prikazuje se mala sličica (*Image*) kako bi korisnik mogao pregledati odabranu sliku. Za odabir slike koristi se *ActivityResultContracts.GetContent*

koji omogućava pristup lokalnom sustavu pohrane korisnika. Korisnik može dodati studente u kolegij pomoću gumba "Dodaj studenta", što otvara padajući izbornik (*DropdownMenu*) s popisom svih dostupnih studenata. Klikom na ikonu "Ukloni" pored imena studenta, korisnik može ukloniti studenta iz popisa. Na kraju zaslona nalazi se gumb "Dodaj kolegij" koji, kada se pritisne, pokreće validaciju podataka i dodaje kolegij u bazu podataka ako su svi podaci ispravno uneseni. Prilikom dodavanja kolegija koristi se funkcija *syncStudentSubjects* iz repozitorija *StudentSubjectRepository* za sinkronizaciju studenata s novim kolegijem. Ako proces traje dulje, prikazuje se *CircularProgressIndicator* koji vizualno informira korisnika da je operacija u tijeku.



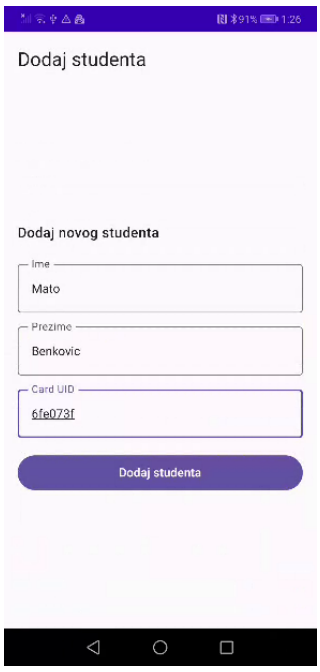
6.7. Izgled zaslona za dodavanje kolegija

Zaslon za prikaz liste studenata (Slika 6.8.) omogućuje profesorima pregled i upravljanje popisom studenata unutar aplikacije. Na početku se podaci o studentima dinamički učitavaju iz Firestore baze podataka pomoću klase *StudentRepository*, a korisnici mogu pregledavati informacije o svakom studentu kroz *StudentCard* komponentu.



Slika 6.8. Izgled zaslona za pregled studenata

Klikom na dugme "Dodaj studenta" korisnici mogu dodati novog studenta prelaskom na zaslon za dodavanje studenata (Slika 6.9.) , dok dugi pritisak na karticu studenta omogućuje brisanje studenta uz potvrdu putem dijaloškog okvira (*AlertDialog*).



Slika 6.9. Izgled zaslona za dodavanje studenta

Zaslon također podržava izvoz popisa studenata u Excel format i uvoz podataka iz CSV datoteke. Funkcija *exportStudentsToExcel* izvozi trenutni popis studenata u Excel format, a *importCSV* (Slika 6.10.) omogućava uvoz podataka o studentima iz odabrane CSV datoteke.

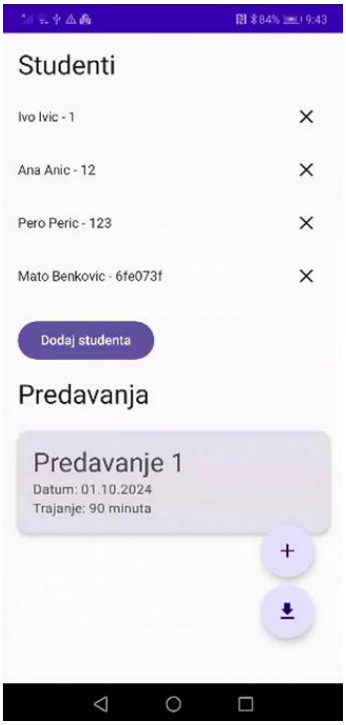
```
fun importCSV(context: Context, uri: Uri) {
    CoroutineScope(Dispatchers.IO).launch {
        try {
            val inputStream: InputStream? = context.contentResolver.openInputStream(uri)
            val reader = inputStream?.bufferedReader()
            val csvData = reader?.readLines()
            inputStream?.close()

            csvData?.let {
                val students = mutableListOf<Student>()
                val header = it.first().split( ...delimiters: ",")
                val rows = it.drop( n: 1)

                rows.forEach { row ->
                    val columns = row.split( ...delimiters: ",")
                    if (columns.size == header.size) {
                        val student = Student(
                            id = columns[0].trim(),
                            name = columns[1].trim(),
                            surname = columns[2].trim(),
                            cardUid = columns[0].trim()
                        )
                        students.add(student)
                    }
                }

                withContext(Dispatchers.Main) {
                    val studentRepo = StudentRepository()
                    studentRepo.addStudents(students)
                    Toast.makeText(context, text: "Uvoz CSV-a uspješan", Toast.LENGTH_LONG).show()
                }
            }
        } catch (e: Exception) {
            withContext(Dispatchers.Main) {
                Toast.makeText(context, text: "Greška pri uvozu CSV-a: ${e.message}", Toast.LENGTH_LONG).show()
            }
        }
    }
}
```

Slika 6.10. Funkcija za uvoz iz CSV datoteke

6.4. Zaslon detalja o kolegiju

Zaslon *SubjectDetailScreen* (Slika 6.11.) omogućava korisnicima pregled i upravljanje podacima o pojedinom kolegiju, uključujući popis predavanja i studenata povezanih s kolegijem. Ovaj zaslon koristi pripadajuće repository klase (*LectureRepository*, *StudentRepository*, *StudentSubjectRepository* i *AttendanceRepository*) kako bi dinamički učitao i prikazao relevantne

podatke iz Firestore baze podataka. Na vrhu zaslona prikazuje se popis studenata pridruženih kolegiju. Svaki student je predstavljen redom koji prikazuje njegovo ime, prezime i jedinstveni identifikator kartice (*Card UID*). Korisnici mogu ukloniti studente s kolegija klikom na ikonu uz ime studenta, što pokreće funkciju *removeStudentSubject*. Za dodavanje novih studenata koristi se gumb koji otvara padajući izbornik (*DropdownMenu*) s popisom svih dostupnih studenata. Ova funkcionalnost omogućuje jednostavno ažuriranje popisa studenata u stvarnom vremenu. Zaslon također omogućuje upravljanje predavanjima unutar kolegija. Prikazuju se sva predavanja u obliku kartica (*LectureItem*), koje sadrže naziv predavanja, datum i trajanje. Korisnici mogu dodavati nova predavanja pomoću plutajućeg akcijskog gumba (*FloatingActionButton*) ili brisati postojeća predavanja dugim pritiskom na karticu, uz potvrdu putem dijaloškog okvira (*AlertDialog*). Klikom na karticu predavanja, korisnik može pristupiti detaljima odabranog predavanja, omogućujući brzi pregled i administraciju predavanja.



Slika 6.11. Zaslon za detalje o kolegiju

Jedna od ključnih funkcionalnosti ovog zaslona je mogućnost izvoza podataka o prisutnosti studenata u Excel format. Funkcija *exportAttendancesToExcel* (Slika 6.12.) omogućava korisnicima izvoz svih relevantnih podataka o prisutnosti, uključujući popis studenata i postotak prisustvovanja na predavanjima.

```
val attendanceMap = mutableMapOf<String, MutableSet<String>>()

CoroutineScope(Dispatchers.IO).launch {
    try {
        // Build a map of student IDs to attended lectures
        for (student in students) {
            val attendances = attendanceRepository.getAttendancesByStudent(student.id)
            val attendedLectures = attendances.map { it.lectureId }.toMutableSet()
            attendanceMap[student.id] = attendedLectures
        }
```

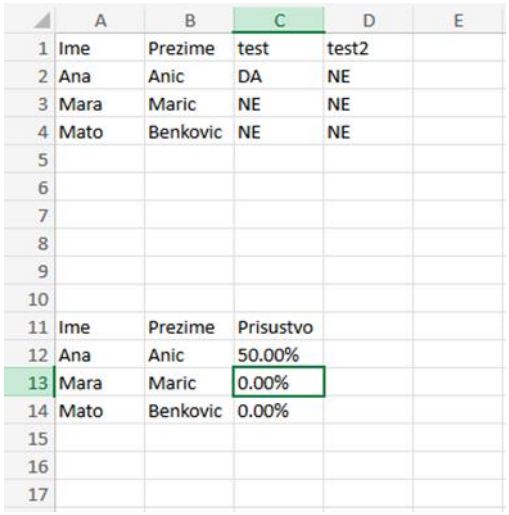
Slika 6.12. Isječak iz funkcije *exportAttendancesToExcel*

Prvo se iz baze podataka prikupljaju podaci o prisutnosti svakog studenta za sva predavanja na koja je student mogao prisustvovati. Podaci se pohranjuju u mapu (*attendanceMap*), gdje je ključ ID studenta, a vrijednost je skup (*Set*) svih ID-ova predavanja na kojima je student prisustvovao.

```
students.forEachIndexed { studentIndex, student ->
    val row = sheet.createRow( rownum: percentageTableStartRow + studentIndex + 1)

    val attendedLecturesCount = attendanceMap[student.id]?.size ?: 0
    val totalLectures = lectures.size
    val attendancePercentage = if (totalLectures > 0) {
        (attendedLecturesCount.toDouble() / totalLectures) * 100
    } else {
        0.0
    }
```

Slika 6.13. Računanje postotka prisutnosti

Nakon što su podaci o prisutnosti prikupljeni, funkcija započinje izračunavanje postotka prisustvovanja svakog studenta (Slika 6.13.). Za svakog studenta se računa omjer između broja predavanja na kojima je student bio prisutan i ukupnog broja predavanja. Ovaj omjer se množi sa 100 kako bi se dobio postotak. Podaci se organiziraju u dvije odvojene tablice unutar jedne Excel datoteke (Slika 6.14.): prva tablica sadrži detalje prisutnosti po predavanjima, dok druga prikazuje ukupan postotak prisustvovanja svakog studenta.



	A	B	C	D	E
1	Ime	Prezime	test	test2	
2	Ana	Anic	DA	NE	
3	Mara	Maric	NE	NE	
4	Mato	Benkovic	NE	NE	
5					
6					
7					
8					
9					
10					
11	Ime	Prezime	Prisustvo		
12	Ana	Anic	50.00%		
13	Mara	Maric	0.00%		
14	Mato	Benkovic	0.00%		
15					
16					
17					

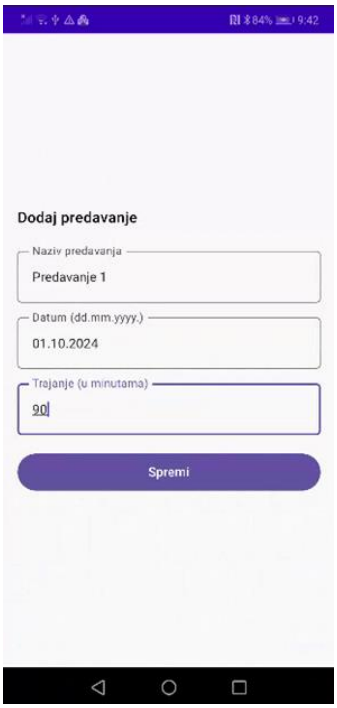
Slika 6.14. Izgled stvorene tablice

6.4.1. Zaslon za dodavanje predavanja

Zaslon za dodavanje predavanja (Slika 6.15.) omogućuje korisnicima unos detalja o novom predavanju koje žele dodati za određeni predmet. Ovaj zaslon sadrži nekoliko polja za unos podataka, kao što su naziv predavanja, datum i trajanje predavanja te koristi osnovne provjere valjanosti kako bi se osigurala točnost informacija prije pohrane u bazu podataka.

Korisnik unosi naziv predavanja, datum (u formatu "dd.mm.yyyy.") i trajanje predavanja (u minutama) koristeći komponente *OutlinedTextField*. Svako od ovih polja ima ugrađenu provjeru valjanosti unosa: provjerava se da li je naziv predavanja unesen, da li datum odgovara očekivanom formatu koristeći regularne izraze (*regex*) te da li je trajanje unijeto kao brojčana vrijednost. Ako bilo koji unos nije valjan, ispod polja prikazuje se odgovarajuća poruka o pogrešci.

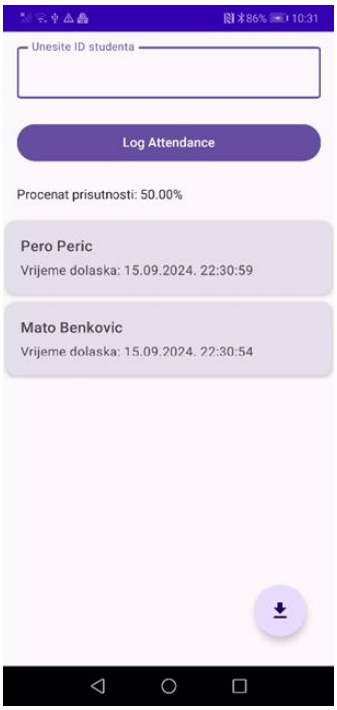
Nakon što korisnik unese sve podatke i oni prođu provjere valjanosti, omogućuje mu se klik na gumb "Spremi" kako bi dodao novo predavanje. Pritiskom na gumb, aplikacija koristi korutine za asinkrono izvršavanje mrežnih operacija kako bi dohvatila podatke o predmetu i kreirala novo predavanje. Novo predavanje se zatim dodaje u Firestore bazu podataka koristeći klasu *LectureRepository*. Ako je operacija uspješna, korisnik dobiva poruku o uspješnom dodavanju predavanja, nakon čega se aplikacija vraća na prethodni zaslon.



Slika 6.15. Zaslon za dodavanje predavanja

6.5. Zaslon detalja o predavanju

Zaslon za detalje predavanja (Slika 6.16.) omogućuje korisnicima pregled i upravljanje informacijama vezanim uz određeno predavanje, uključujući evidenciju prisustva studenata. Ovaj zaslon nudi funkcionalnosti kao što su pregled prisutnosti, bilježenje dolazaka putem Bluetooth uređaja, ručno dodavanje prisutnosti te izvoz podataka u Excel format za daljnju obradu i analizu.



Slika 6.16. Zaslon za detalje predavanja

Kako bi se uspostavila Bluetooth veza između aplikacije i NFC čitača (ESP32), koristi se standardni SPP UUID (*MY_UUID*). Funkcija *connectToBluetoothDevice* (Slika 6.17.) prvo pretražuje popis uparenih Bluetooth uređaja kako bi pronašla uređaj s nazivom "ESP32_NFC".

Kada se uređaj pronađe, aplikacija inicijalizira socket za komunikaciju pomoću *device.createRfcommSocketToServiceRecord(MY_UUID)* i pokušava uspostaviti vezu pomoću metode *connect()*.

```
fun connectToBluetoothDevice(deviceName: String) {
    bluetoothAdapter?.bondedDevices?.find { it.name == deviceName }?.let { device ->
        coroutineScope.launch(Dispatchers.IO) {
            try {
                bluetoothSocket = device.createRfcommSocketToServiceRecord(MY_UUID)
                bluetoothSocket?.connect()
                listenForUID(bluetoothSocket!!)
            } catch (e: Exception) {
                withContext(Dispatchers.Main) {
                    Toast.makeText(
                        context,
                        text: "Povezivanje s bluetooth uređajem nije uspjelo",
                        Toast.LENGTH_SHORT
                    ).show()
                }
                e.printStackTrace()
            }
        }
    }
}
```

Slika 6.17. Funkcija za povezivanje s bluetooth uređajem

Nakon što je veza uspostavljena, funkcija *listenForUID* (Slika 6.18.) sluša dolazne podatke iz Bluetooth uređaja koristeći ulazni tok (*inputStream*). Podaci koji se primaju putem Bluetootha obično predstavljaju jedinstveni identifikacijski broj (UID) RFID kartice. Kada aplikacija primi UID, poziva funkciju *logAttendance* kako bi zabilježila prisustvo studenta u sustavu.

```
fun listenForUID(socket: BluetoothSocket) {
    CoroutineScope(Dispatchers.IO).launch {
        try {
            val inputStream: InputStream = socket.inputStream
            val buffer = ByteArray( size: 1024)
            var bytes: Int

            while (true) {
                bytes = inputStream.read(buffer)
                val incomingMessage = String(buffer,  offset: 0, bytes)
                receivedUID = incomingMessage.trim()

                logAttendance(receivedUID)
            }
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }
}
```

Slika 6.18. *listenForUID* funkcija

Bluetooth integracija omogućava aplikaciji da automatski bilježi prisutnost studenata na temelju podataka primljenih iz NFC čitača. Funkcija *logAttendance* provjerava UID kartice studenta i bilježi dolazak u bazu podataka ako je kartica valjana. Ova funkcionalnost značajno smanjuje potrebu za ručnim unosom i minimizira mogućnost grešaka, čineći proces evidencije prisustva bržim i efikasnijim.

7. ZAKLJUČAK

Cilj ovog završnog rada bio je izraditi mobilnu aplikaciju za evidenciju prisutnosti studenata koristeći RFID/NFC tehnologiju i Android platformu. Tijekom razvoja, primijenjene su moderne tehnologije, kao što su Firebase za autentifikaciju i pohranu podataka te Jetpack Compose za izradu korisničkog sučelja. Rezultati istraživanja postojećih aplikacija pokazali su da na tržištu postoji značajan broj aplikacija za evidenciju prisutnosti, no malo njih nudi integraciju RFID/NFC tehnologije u svrhu brze i pouzdane identifikacije studenata, što je bila jedna od glavnih motivacija za razvoj ove aplikacije.

Aplikacija razvijena u ovom radu implementira nekoliko ključnih funkcionalnosti, uključujući autentifikaciju korisnika putem Firebase servisa, upravljanje podacima o studentima, predmetima i predavanjima te evidenciju prisutnosti studenata na predavanjima pomoću NFC tehnologije. Korisničko sučelje aplikacije je intuitivno i prilagođeno različitim ulogama korisnika (student, profesor, administrator), čime se omogućuje lako upravljanje svim aspektima sustava. Funkcionalnost aplikacije dodatno je obogaćena mogućnošću generiranja izvještaja i izvoza podataka u Excel format, što omogućuje lakše praćenje prisutnosti i administraciju podataka.

Razvoj aplikacije temeljio se na MVVM (Model-View-ViewModel) arhitekturi, što je omogućilo jasno razdvajanje poslovne logike od korisničkog sučelja i značajno olakšalo održavanje i proširivost aplikacije. Korištenje Jetpack Compose-a omogućilo je bržu izradu prilagodljivih i responzivnih korisničkih sučelja, dok je integracija s Firebase-om pružila pouzdano rješenje za pohranu podataka i autentifikaciju korisnika.

Jedan od glavnih izazova tijekom razvoja bio je osigurati sigurnu i učinkovitu komunikaciju između hardverskih i softverskih komponenti. Integracija RFID/NFC tehnologije omogućila je jednostavan i brz proces evidencije prisutnosti, ali je zahtijevala i dodatnu pažnju u pogledu sigurnosti podataka i zaštite privatnosti korisnika. Bluetooth tehnologija, korištena za komunikaciju s RFID čitačem, pružila je stabilnu platformu za povezivanje uređaja i olakšala proces evidencije prisutnosti.

Zaključno, ovaj završni rad predstavlja inovativno rješenje za digitalnu evidenciju prisutnosti studenata, koje koristi modernu tehnologiju i usmjereno je prema poboljšanju učinkovitosti i točnosti evidencije. Aplikacija je uspješno implementirana i već sada nudi značajne prednosti u usporedbi s klasičnim metodama evidencije. Međutim, prepoznati su i potencijalni pravci daljnjeg razvoja, kao što su unaprjeđenje dizajna korisničkog sučelja, dodatne funkcionalnosti za analizu

podataka te bolja integracija s drugim sustavima za upravljanje obrazovnim procesima. Ovim daljnjim unapređenjima, aplikacija bi mogla postati još relevantnija i korisnija

LITERATURA

1. [1] Clever.Apps (2024), Attendy [mobilna aplikacija], dostupno na: https://play.google.com/store/apps/details?id=fr.gym.mattnissartdevelopment.attendanceApp [13.09.2024.]
2. [2] Jibble Group (2024), Attendance Tracker by Jibble [mobilna aplikacija], dostupno na: https://play.google.com/store/apps/details?id=io.jibble.androidclient.jibble&hl=en [13.09.2024.]
3. [3] Dasduino (2023), Dasduino ESP32 Connectplus [mikrokontroler], dostupno na: https://soldered.com/product/dasduino-connectplus/?srsltid=AfmBOorVjC6JOyyk7OYLXKWtiGekX7T4SptW1D6Z2XBR_5-wUdPTPVR- [14.09.2024.]
4. [4] EasyC (2023), EasyC adapter [adapter za I2C komunikaciju], dostupno na: https://soldered.com/product/easyc-adapter/ [14.09.2024.]
5. [5] Elechouse (2023), PN532 NFC RFID Module V4 [RFID/NFC modul], dostupno na: https://www.elechouse.com/product/pn532-nfc-rfid-module-v4/ [14.09.2024.]
6. [6] Arduino (2024), Arduino IDE (Verzija 2.3.2), dostupno na: https://www.arduino.cc/en/software [14.09.2024.]
7. [7] Google (2024), Firebase [platforma za razvoj aplikacija], dostupno na: https://firebase.google.com/ [14.09.2024.]
8. [8] Google (2023), Android Studio (Verzija 2024.1.2 "Koala"), dostupno na: https://developer.android.com/studio [14.09.2024.]
9. [9] JetBrains (2024), Kotlin (Verzija 2.0.20), dostupno na: https://kotlinlang.org/ [14.09.2024.]
10. [10] Google (2024), Jetpack Compose (Verzija 1.7.1.), dostupno na: https://developer.android.com/jetpack/compose [14.09.2024.]

SAŽETAK

Ovaj završni rad prikazuje razvoj mobilne Android aplikacije za evidenciju prisutnosti studenata koristeći RFID/NFC tehnologiju. Cilj je bio kreirati rješenje koje automatizira i ubrzava proces praćenja prisutnosti na predavanjima, koristeći studentske identifikacijske kartice. Aplikacija je izrađena u Android Studio razvojnom okruženju koristeći Kotlin jezik i Jetpack Compose za izradu korisničkog sučelja. U sklopu aplikacije implementirane su ključne funkcionalnosti poput upravljanja korisnicima, predmetima i predavanjima te evidencije prisutnosti putem integracije s Firebase platformom za autentifikaciju i pohranu podataka. Bluetooth veza omogućuje komunikaciju s NFC čitačem, olakšavajući proces identifikacije. Aplikacija također omogućuje generiranje izvještaja i izvoz podataka u Excel format, što dodatno pojednostavljuje administrativne zadatke. Razvoj aplikacije temeljio se na MVVM arhitekturi, osiguravajući razdvajanje poslovne logike od korisničkog sučelja i time olakšavajući održavanje i proširivost. Glavni izazovi su bili osigurati sigurnu i učinkovitu komunikaciju između hardverskih i softverskih komponenti, ali uz pravilnu integraciju, aplikacija pruža pouzdano i inovativno rješenje za evidenciju prisutnosti studenata.

Ključne riječi: Android, evidencija, NFC

ABSTRACT

Title: Mobile Android Application for Student Attendance Records Using RFID/NFC Reader

This final thesis presents the development of a mobile Android application designed for recording student attendance using RFID/NFC technology. The primary goal of the project was to create an efficient and automated solution for tracking student attendance during lectures by leveraging student ID cards equipped with RFID/NFC tags. The application was built in the Android Studio environment using the Kotlin programming language and Jetpack Compose for the user interface. It incorporates key functionalities such as managing users, subjects, and lectures, as well as tracking attendance through a Firebase-based system for authentication and data storage. The Bluetooth connection facilitates communication with the NFC reader, streamlining the identification process. Additional features include generating reports and exporting data to Excel format, which simplifies administrative tasks. The development followed the MVVM architecture, ensuring a clear separation of business logic from the user interface, thereby enhancing maintainability and scalability. The main challenges involved securing seamless communication between hardware and software components, but the final implementation successfully provides a reliable and modern solution for managing student attendance.

Keywords: Android, attendance, NFC