Implementacija pametne brave

Gatarić, Jakov

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Jakov Gatarić

Implementacija pametne brave

DIPLOMSKI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Jakov Gatarić

Matični broj: 0016133101

Studij: Baze podataka i baze znanja

Implementacija pametne brave

DIPLOMSKI RAD

Mentor/Mentorica:

Prof. dr. sc. Ivan Magdalenić

Varaždin, rujan 2024.

Jakov Gatarić

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Ovaj diplomski rad bavi se razvojem pametne brave koja koristi biometrijsku autentifikaciju putem otisaka prstiju za kontrolu pristupa. Glavni cilj je osigurati visoku razinu sigurnosti uz jednostavnost korištenja kroz mobilnu aplikaciju za Android. Rad obuhvaća integraciju različitih tehnologija poput NodeMCU mikrokontrolera, AS608 senzora za otisak prsta, solenoidne brave, te MQTT protokola za komunikaciju između mobilne aplikacije i brave. Sigurnosni aspekti sustava osigurani su primjenom TLS enkripcije za prijenos podataka, kao i AES256 enkripcije sa soli za pohranu biometrijskih podataka. Poseban naglasak stavljen je na sigurnosne mjere kako bi se zaštitili osjetljivi podaci te spriječili neovlašteni pristupi. Kroz razradu tehničke i softverske implementacije, rad pruža detaljan uvid u proces izgradnje sigurnosnog sustava temeljenog na IoT tehnologiji. Rad također razmatra prednosti i izazove biometrijske autentifikacije u sigurnosnim sustavima te koristi MVVM arhitekturu za razvoj mobilne aplikacije, omogućujući modularnost i lakoću održavanja.

Ključne riječi: Internet of Things, Arduino, AS608, pametna brava, sigurnost, enkripcija android aplikacija,

Sadržaj

1. Uvod

Tema ovog diplomskog rada usmjerena je na razvoj pametne brave koja koristi biometrijske podatke za autentifikaciju korisnika. U današnjem svijetu, sigurnost i zaštita podataka postaju sve važniji faktori u svakodnevnom životu, pogotovo s obzirom na sve veću primjenu pametnih uređaja u domovima i poslovnim prostorima. Pametne brave pružaju višu razinu sigurnosti u odnosu na tradicionalne mehaničke brave, omogućujući korisnicima jednostavno i sigurno upravljanje pristupom. Odabir teme motiviran je potrebom za naprednijim sustavima kontrole pristupa koji kombiniraju praktičnost, tehnologiju i sigurnost. Implementacija biometrijske autentifikacije, u ovom slučaju putem otiska prsta, pruža dodatnu razinu sigurnosti, budući da se osobni biometrijski podaci teško mogu replicirati. Osim toga, integracija pametne brave s mobilnom aplikacijom omogućuje daljinsko upravljanje, što korisnicima pruža fleksibilnost i kontrolu nad sigurnosnim aspektima njihovog prostora.

Osobna motivacija za odabir ove teme proizlazi iz interesa za napredne tehnologije poput biometrijskih sustava i IoT (Internet of Things) uređaja, kao i želje za istraživanjem sigurnosnih izazova u njihovoj implementaciji. Također, razvoj mobilnih aplikacija predstavlja važan dio ovog rada jer omogućuje jednostavno korisničko sučelje za upravljanje sustavom, čime se osigurava intuitivno i praktično rješenje. Zainteresiranost za razvoj mobilnih aplikacija dodatno je motivirala pristup ovoj temi, s naglaskom na integraciju funkcionalnosti pametne brave i korisničkog iskustva putem Android platforme

2. Metode i tehnike rada

Prilikom izrade ovog diplomskog rada korištene su različite aplikacije i alati za razvoj softverskog i hardverskog dijela projekta. Arduino IDE korišten je za programiranje NodeMCU-a, mikrokontrolera odgovornog za upravljanje pametnom bravom. Razvoj Android aplikacije, napisane u Kotlinu, proveden je u razvojnom okruženju Android Studio, dok je alat MQTT Explorer korišten za pregled i testiranje komunikacije između uređaja i MQTT servera. Cijeli projekt razvijan je na operativnom sustavu Windows 11.

Teorijski dio rada izrađen je primjenom metode sinteze, kako bi se složena znanstvena istraživanja prezentirala i objasnila na jasan i razumljiv način. Ovaj pristup omogućava da se rad prikaže široj publici, uključujući one koji nisu detaljno upoznati s temom ili tehnologijama vezanim uz Internet stvari (IoT). U istraživanju literature za izradu teorijskog dijela korišteni su uglavnom strani izvori, uz nadopune domaće literature. Osim knjiga koje pokrivaju temu IoT-a, konzultirani su i relevantni znanstveni i stručni članci kako bi se osigurao sveobuhvatan i točan pregled područja.

2.1. Arduino IDE

Arduino IDE (Integrated Development Environment) je softverski alat koji omogućava pisanje, testiranje i upload programskog kôda na mikrokontrolere kao što su Arduino ili u ovom slučaju, NodeMCU. Razvojno okruženje Arduino IDE koristi se zbog svoje jednostavnosti, fleksibilnosti i široke podrške za različitim hardverskim platformama. Arduino IDE podržava C i C++ jezike, uz dodatak specifičnih funkcija koje olakšavaju rad s Arduino hardverom. Sama struktura programa u Arduino IDE-u sastoji se od dvije glavne funkcije: setup() i loop(). Funkcija setup() koristi se za inicijalizaciju varijabli i postavljanje početnih uvjeta, dok se funkcija loop() koristi za stalno izvršavanje glavnog programskog toka. Ova struktura je idealna za rad s ugrađenim sustavima jer omogućava neprekidan rad programa na mikrokontroleru (Söderby K, 2024).

U ovom diplomskom radu korištene su različite biblioteke unutar Arduino IDE-a kako bi se omogućila implementacija Wi-Fi komunikacije putem NodeMCU modula, kao i integracija senzora otiska prsta i solenoidne brave koji će biti kasnije objašnjeni u radu.

2.2. Android Studio

Android Studio je službeno razvojno okruženje (IDE) za Android aplikacije, koje pruža sve potrebne alate za izradu, testiranje i implementaciju aplikacija na Android platformi. Izgrađen je na IntelliJ IDEA platformi, a njegova glavna prednost leži u integraciji s Android SDK-om i širokoj podršci za različite Android uređaje (Android developers, 2024).

U ovom diplomskom radu, Android Studio je korišten za razvoj mobilne aplikacije koja omogućuje korisnicima daljinsko upravljanje pametnom bravom. Aplikacija je pisana u Kotlin programskom jeziku koji je službeno podržan za razvoj Android aplikacija od 2017. godine. Kotlin pruža jednostavniju i efikasniju sintaksu u odnosu na Javu, smanjujući potrebu za previše detaljnim pisanjem kôda, dok istovremeno omogućuje lakšu integraciju naprednih značajki. Prilikom izrade aplikacije, korištena je **MVVM** arhitektura, odnosno korišteni su **Model**, **View** i **ViewModel** za organizaciju i prikaz podataka u aplikaciji. Korištenjem ovakve arhitekture, aplikacija je modularna i lakša za održavanje.

2.3. MQTT Explorer

MQTT Explorer je napredan alat za vizualizaciju i upravljanje MQTT protokolom, razvijen kako bi olakšao pregled i testiranje komunikacije unutar MQTT mreže. Glavna prednost MQTT Explorera je njegova mogućnost da u realnom vremenu prikaže strukturu svih MQTT poruka unutar pojedine teme. Njegova sposobnost prikaza hijerarhije tema i povijesti poruka pomaže u brzom prepoznavanju i rješavanju problema u komunikaciji (Nordquist, 2024).

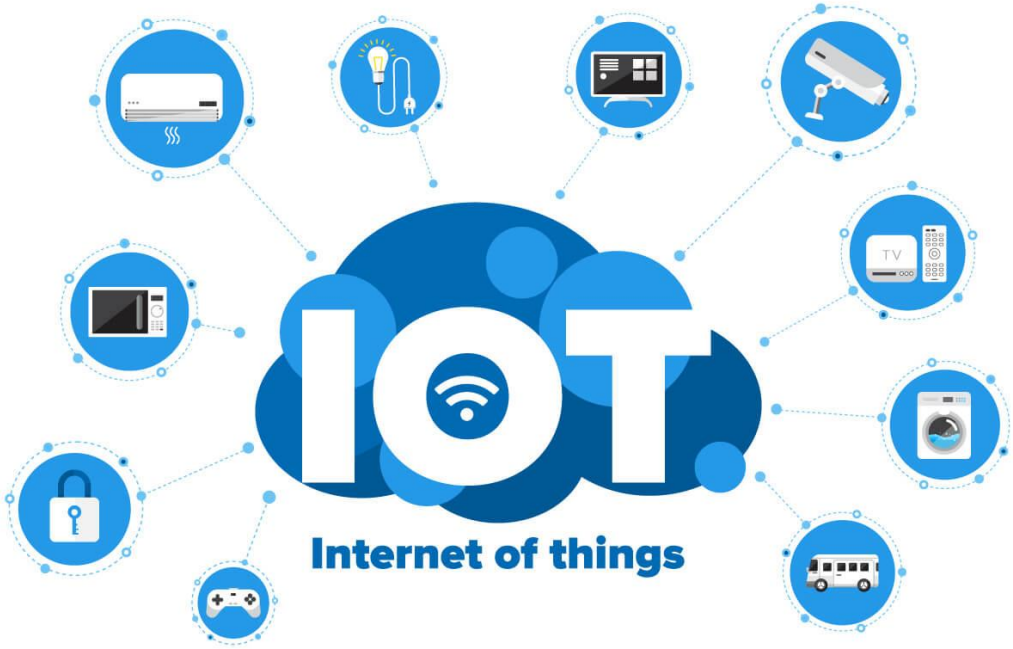
Korišten je tijekom razvoja ovog rada za testiranje i praćenje komunikacije između pametne brave, mobilne aplikacije i MQTT poslužitelja, prije nego su oba uređaja bila u potpunosti povezana i sposobna samostalno komunicirati bez greške.

3. Teorijska razrada sustava pametne brave

U ovom poglavlju razrađuje se cjelokupni sustav pametne brave koji kombinira biometrijsku autentifikaciju, Internet of Things tehnologiju te sigurnosne i komunikacijske protokole. Sustav je razvijen kako bi omogućio sigurnu kontrolu pristupa, koristeći biometrijske podatke korisnika, uz daljinsko upravljanje putem mobilne aplikacije. Kroz analizu arhitekture sustava, svih potrebnih komponenti i sigurnosnih aspekata, ovo poglavlje osigurava detaljan pregled implementacije i načina na koji sustav funkcionira.

3.1. Internet of Things

Internet of Things (IoT) predstavlja mrežu fizičkih uređaja, senzora, softvera i drugih tehnologija koji su međusobno povezani putem interneta kako bi razmjenjivali i analizirali podatke. Cilj IoT-a je omogućiti ovim uređajima neovisno donošenje odluka na temelju prikupljenih podataka, bez potrebe za ljudskom intervencijom. IoT se može smatrati ključnim tehnološkim okvirom koji omogućava sveobuhvatnu automatizaciju, povećanu učinkovitost te fleksibilnost u upravljanju uređajima. Ova tehnologija nalazi svoju primjenu u brojnim područjima, uključujući industrijsku automatizaciju, pametne domove, zdravstvo, transport i sigurnosne sustave (Lea, 2018).



Slika 1: Internet of Things (Globalsign; 2024; izvor: https://www.globalsign.com/en-sg/blog/what-internet-things-and-how-does-it-work)

U kontekstu sigurnosnih sustava, posebno pametnih brava, IoT omogućuje daljinsko upravljanje i nadzor nad pristupnim sustavima putem povezanih uređaja. Njegova arhitektura omogućuje, npr. Povezivanje pametnih brava s mobilnim aplikacijama, što korisniku omogućuje upravljane sigurnosnim mehanizmima u stvarnom vremenu, neovisno gdje se on nalazi. Prema knjizi *IoT Security: Advances in Authentication*, upravo ta povezanost uređaja donosi brojne sigurnosne izazove, poput mrežnih napada i rizika od neovlaštenog pristupa. Kod pametnih brava, IoT tehnologija pruža mogućnost korištenja biometrijskih sustava, daljinsko upravljanje i praćenje aktivnosti, čime se osigurava visoka razina sigurnosti i korisničke kontrole.

Jedan od glavnih sigurnosnih aspekata IoT-a u pametnim bravama jest zaštita prijenosa podataka između mobilnih uređaja i sustava brava. Kako bi se spriječilo neovlašteno pristupanje i manipulacija podacima, u ovom radu koristi se MQTT protokol u kombinaciji s TLS enkripcijom, čime se osigurava siguran kanal za komunikaciju između pametne brave i mobilne aplikacije.

MQTT (Message Queuing Telemetry Transport) je protokol koji je lagan za primjenu, posebno dizajniran za prijenos podataka između IoT uređaja, a njegova upotreba u sigurnosnim sustavima omogućava brzu i sigurnu razmjenu informacija (MQTT, 2024).

TLS (Transport Layer Security) je kriptografski protokol dizajniran za osiguranje privatnosti integriteta podataka koji se razmjenjuju između komunikacijskih aplikacija, najčešće između klijenata i poslužitelja na internetu. TLS koristi kombinaciju simetrične i asimetrične enkripcije kako bi se osiguralo da treće strane ne mogu presresti niti izmijeniti podatke tijekom prijenosa (Cloudfare, 2024).

3.2. Primjena IoT tehnologije u pametnim bravama

Kod pametnih brava, IoT tehnologija omogućava više funkcionalnosti. Korisnici mogu daljinski otključavati i zaključavati brave, pratiti tko ulazi i izlazi iz zgrade te dobivati obavijesti o sigurnosnim incidentima u stvarnom vremenu. Takva integracija s mobilnom aplikacijom omogućava fleksibilnost prilikom upravljanja pristupom, a sustav je vrlo često povezan i s nekim drugim pametnim uređajem, poput kamere, alarmnog sustava ili senzora pokreta, čime se stvara složeniji sigurnosni okvir unutar pametnog doma ili poslovnog prostora.

Prema knjizi *Internet of Things for architect,* ključna prednost IoT tehnologije leži u njezinoj sposobnosti automatizacije i integracije s raznim sustavima. Pametne brave, koje su u konstantnoj vezi s mobilnom aplikacijom preko interneta, omogućuju korisnicima praćenje sigurnosnih zapisa te upravljanje pristupom, bez obzira na fizičku udaljenost. Osim toga, IoT omogućava povijesno praćenje aktivnosti unutar sustava, omogućujući korisnicima analiziranje tko i kada je imao pristup nekom od zaštićenih prostora. Međutim, uz brojne prednosti, IoT sigurnosni sustavi imaju i svoje izazove. Prema knjizi *IoT Security: Advances in Authentication*, jedan od ključnih izazova je osigurati povjerljivost podataka u okruženju gdje su uređaji često izloženi mrežnim prijetnjama. To zahtijeva primjenu naprednih metoda autentifikacije, enkripcije te upravljanje identitetom uređaja, što je ključno za održavanje cjelokupnog integriteta sustava.

U ovom radu, sustav pametne brave je implementiran kao dio šireg IoT sustava. Biometrijska autentifikacija pomoću otiska prsta omogućuje korisnicima jednostavan, ali siguran način ulaska u prostor, dok IoT infrastruktura osigurava korisnicima daljinsko upravljanje bravom putem mobilne aplikacije. Sigurnost podataka, kao i zaštita biometrijskih podataka, osigurana je putem sigurnosnih protokola i algoritama, što minimizira rizike od neovlaštenog pristupa i manipulacije podacima.

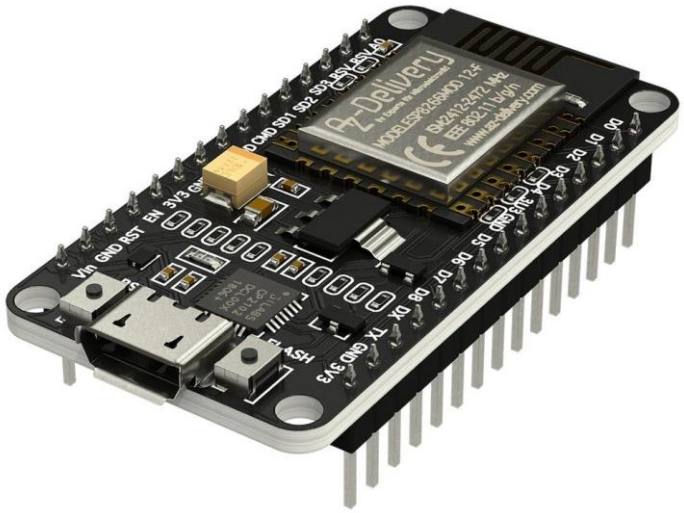
3.2.1. Implementacija NodeMCU i komponenti

U ovom će poglavlju biti opisane komponente potrebne za implementaciju sustava pametne brave korištenjem NodeMCU mikrokontrolera i ostalih ključnih komponenti. Glavna

funkcionalnost ovog sustava temelji se na autentifikaciji korisnika pomoću senzora za otisak prsta, upravljanjem solenoidnom bravom i dobivanjem povratne poruke na LCD ekranu. Sustav koristi NodeMCU kao centrali kontroler, dok su AS608 senzor za otisak prsta, solenoidna brava povezana s relejem i napajana 9V baterijom, te LCD ekran povezani s njim kako bi se omogućila sigurna i učinkovita kontrola pristupa. Kako će senzor za otisak prsta biti opisan kasnije, sada će ostale komponente biti detaljnije opisane.

3.2.1.1. NodeMCU

NodeMCU je mikrokontroler koji se temelji na ESP8266 Wi-Fi modulu, a posebno je popularan zbog svoje mogućnosti povezivanja s internetom putem Wi-Fi mreže. U sklopu sustava pametne brave, NodeMCU igra ključnu ulogu kao centrali kontroler koji upravlja komunikacijom između različitih hardverskih komponenti te omogućava povezivanje sustava s mobilnom aplikacijom i MQTT serverom (github, 2024).



Slika 2: NodeMCU (Hackster.io; 2024; izvor: https://www.hackster.io/javagoza/nodemcu-amica-v2-road-test-2e8bff)

Jedna od najvećih prednosti NodeMCU-a je njegova integracija s Wi-Fi- modulom, što omogućava bežičnu komunikaciju bez potrebe za dodatnim modulima ili perifernim uređajima. Zbog toga je idealan i vrlo popularan izbor za IoT projekte u kojima je ključna bežična komunikacija između uređaja. Programiranje kontrolera je moguće putem ArduinoIDE koji je prethodno opisan. ArduinoIDE podržava veliki broj biblioteka za rad s različitim senzorima i modulima, uključujući i MQTT, što znatno ubrzava razvoj IoT projekata.

Zahvaljujući svom niskom trošku, integriranoj Wi-Fi funkcionalnosti i podršci za protokole poput MQTT i TLS/SSL enkripcije, NodeMCU je postao jedan od najpopularnijih mikrokontrolera za IoT projekte, a posebno je pogodan za projekte gdje je potrebna bežična kontrola i daljinska komunikacija (github, 2024).

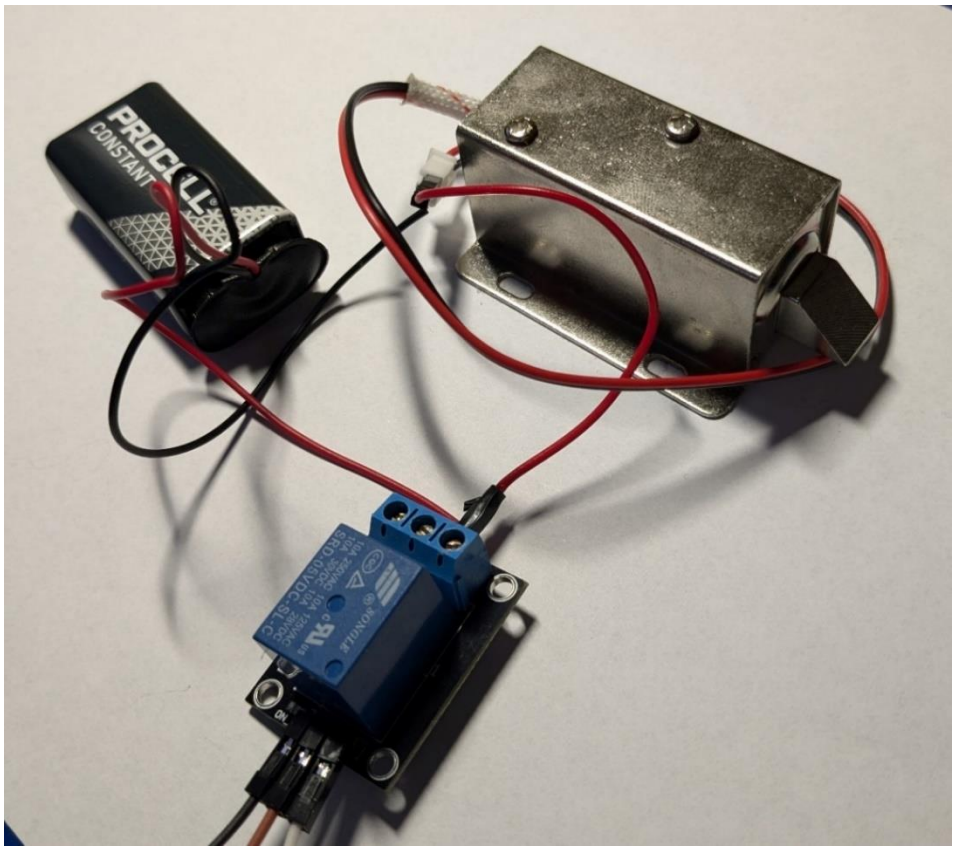
Sigurnost je važan aspekt prilikom korištenja NodeMCU-a, a podrška za TLS/SSL protokole omogućava siguran prijenos podataka između sustava i mobilne aplikacije putem MQTT-a. Implementacijom ovih protokola osigurava se da osjetljivi podaci, kao što su biometrijski podaci ili naredbe za upravljanje bravom, ne budu presretnuti ili manipulirani tijekom prijenosa preko Wi-Fi mreže.

Zbog svojih mogućnosti i fleksibilnosti NodeMCU je bio optimalan izbor za ovaj projekt, omogućavajući sigurnu i pouzdanu platformu za upravljanje bravom i ostalim komponentama.

3.2.1.2. Solenoidna brava

Solenoidna brava u ovom sustavu odgovorna je za fizičko otključavanje i zaključavanje vrata. Brava funkcionira na elektromagnetnom principu, odnosno, kada se struja provodi kroz solenoid, stvara se magnetno polje koje pokreće klip, čime se brava otključa. Ako pak struja prestane teći, opruga vraća klip u početni položaj i vrata se zaključavaju (ArduinoGetStarted, 2024).

Solenoidna brava obično koristi napajanje od 12V, 24V ili 48V, stoga se ne može direktno spojiti na kontroler (ArduinoGetStarted, 2024). Za ispravan rad brave, koristi se relej koji omogućava kontrolu protoka struje iz 9v baterije. Relej je povezan s mikrokontrolerom, a njegov je zadatak uključivanje i isključivanje strujnog kruga prema solenoidnoj bravi na temelju naredbi dobivenih od strane NodeMCU-a. Spajanje na relej osigurava da je solenoidna brava aktivirana samo kada je potrebno otvoriti vrata, čime se štedi energija i osigurava dugotrajan rad sustava. Osim releja, brava je također spojena s 9V baterijom koja osigurava napajanje što čini sustav neovisnim o vanjskom napajanju pa je tako idealan za mjesta gdje standardna električna mreža nije dostupna. Baterijsko napajanje također osigurava rad sustava tijekom nestanka struje, čime se povećava pouzdanost i sigurnost.



Slika 3: Solenoidna brava povezana s relejom i 9V baterijom (izvor: slikao autor, 2024)

3.2.1.3. LCD ekran

LCD ekran u sustav pametne brave koristi se za pružanje povratnih informacija korisniku. Na ekranu se prikazuju različite poruke koje informiraju korisnika o trenutnom stanju sustava. Jedna od funkcionalnosti ekrana je prikazivanje nasumično generirane šifre koja se koristi za povezivanje mobilne aplikacije s kontrolerom putem MQTT protokola. Kada korisnik pokuša uspostaviti vezu između mobilne aplikacije i pametne brave, na ekranu se prikazuje šifra koju korisnik mora unijeti u aplikaciju kako bi se ostvarila sigurna veza. Osim toga, LCD ekran pruža važne povratne informacije prilikom procesa autentifikacije. Nakon što korisnik postavi prst na AS608 senzor, na ekranu se prikazuje poruka koja potvrđuje može li osoba ući ili joj je zabranjen ulaz u prostor. U slučaju da je otisak prsta prepoznat, vrata se automatski otključavaju, a korisniku se prikazuje poruka o uspješno provedenoj autentifikaciji. Ako senzor ne uspije prepoznati otisak prsta, na ekranu se prikazuje obavijest kojom se korisnika traži da ponovno postavi prst na senzor.

Integracija LCD ekrana značajno poboljšava cjelokupno korisničko iskustvo jer korisnicima pruža jasne i pravovremene informacije o stanju sustava. Na ovaj način, korisnici

su uvijek svjesni je li pristup odobren ili odbijen, čime se povećava sigurnost sustava i smanjuje mogućnost nesporazuma ili sigurnosnih propusta uzrokovanih nedostatkom povratnih informacija.

3.2.1.4. MQTT

MQTT je jedan od najvažnijih protokola u svijetu IoT-a, a dizajniran je za prijenos podataka između uređaja s niskom propusnošću i ograničenim resursima (MQTT, 2024). Protokol nudi jednostavnu i učinkovitu komunikaciju putem modela „publish/subscribe“, što ga čini idealnim izborom za sustave poput pametne brave. U kontekstu ovog rada, MQTT je odabran kao glavni komunikacijski protokol između pametne brave i mobilne aplikacije zbog svoje sposobnosti brze i sigurne razmjene podataka uz minimalnu potrošnju energije i propusnosti.

Prednosti MQTT-a su mnogobrojne. Ovaj protokol koristi klijent-server arhitekturu, gdje klijent, u ovom slučaju NodeMCU, šalje i prima podatke putem posredničkog sustava poznatog kao MQTT broker. Na taj način, pametna brava može slati informacije o stanju brave, dok mobilna aplikacija šalje naredbe za otključavanje ii zaključavanje vrata. Ključna prednost ovog modela u odnosu na tradicionalne protokole kao što je HTTP leži u tome što MQTT omogućuje slanje samo nužnih podataka, čime se izbjegava stalno održavanje veze i nepotrebna potrošnja resursa (Lea, 2018). Ovaj pristup rezultira nižom potrošnjom energije, što je ključno za uređaje poput pametnih brava koje često rade na baterije.

Također je važno napomenuti kako je korišten MQTT Explorer kao alat za testiranje komunikacije pametne brave i mobilne aplikacije. Ovaj alat omogućava pregled svih poruka poslanih preko MQTT-a kako bi se osiguralo da su svi podaci ispravno preneseni i da nema gubitka informacija tijekom prijenosa. Ovakvi alati omogućavaju detaljan nadzor nad komunikacijom unutar IoT mreža, što je ključno za pravovremeno otkrivanje potencijalnih sigurnosnih propusta.

3.2.1.5. Sigurnosni aspekti

Sigurnost podataka igra ključnu ulogu u svakom sustavu koji se susreće s biometrijskom autentifikacijom, a u ovom slučaju posebna pažnja posvećena je zaštiti osjetljivih biometrijskih informacija i komunikaciji između pametne brave i mobilne aplikacije. Za zaštitu komunikacije koristi se TLS protokol unutar MQTT-a, koji osigurava enkripciju podataka prilikom slanja i primanja informacija (Vacca, 2017). TLS štiti podatke od napada presretanja, poznatijih kao „main-in-the-middle“ napadi, te osigurava siguran i zaštićeni prijenos podatka između NodeMCU-a, mobilne aplikacije i MQTT brokera.

Osim zaštite komunikacijskog kanala, vrlo je bitna i sama zaštita pohranjenih biometrijskih podataka, točnije, otisaka prstiju. U ovom sustavu koristi se AES256 enkripcija s dodatkom soli za kriptiranje otisaka prstiju na senzoru. AES256 (Advanced Encryption Standard) je algoritam simetrične enkripcije koji koristi 256-bitni ključ, što ga čini jednim od najpouzdanijih kriptografskih standarda u industriji (Vacca, 2017). Ovaj je algoritam odabran zbog svoje robusnosti i otpornosti na napade brute-force metodom, s obzirom na veličinu ključa koji se koristi za šifriranje podataka. Korištenje ovakvog nivoa enkripcije značajno smanjuje mogućnost dešifriranja podataka, čak i ako bi došlo do fizičkog pristupa uređaju ili njegovim komponentama.

Ukoliko ovaj algoritam ne bi bio dovoljan, dodana je i sol pri enkripciji koja dodatno pojačava sigurnost. Sol je slučajni niz podataka koji se dodaje izvornoj vrijednosti otiska prsta prije nego što se isti šifrira (Vacca, 2017). Ovaj postupak sprječava različite napade jer svaki pohranjeni otisak prsta, čak i ako je identičan s nekim drugim otiskom, ima jedinstveni enkriptirani izlaz zbog dodane soli. Kombinacija AES256 enkripcije i soli osigurava visok nivo sigurnosti i zaštite od naprednih tehnika kriptoanalize.

Prednosti korištenja AES256 enkripcije sa soli je višestruka. Prvotno, AES256 pruža visoku otpornost na brute-force napade. Zahvaljujući 256-bitnom ključu, broj mogućih kombinacija je izuzetno velik, što čini ovakve napade neučinkovitima i vremenski neisplativima. Drugo, korištenje AES256 osigurava zaštitu osjetljivih podataka, poput otisaka prstiju, koji zahtijevaju dodatnu sigurnost zbog svoje nepromjenjive prirode. Za razliku od lozinki koje je moguće resetirati, biometrijski podaci moraju ostati trajno zaštićeni, a AES256 enkripcija omogućuje da ti podaci ostanu sigurni čak i u slučaju fizičkog pristupa uređaju. Treće, enkripcija soli osigurava da svaki enkriptirani izlaz bude jedinstven. To znači da se čak i identični otisci prstiju šifriraju različito, čime se sprječavaju napadi koji se oslanjaju na unaprijed dekriptirane podatke, kao što su napadi uz pomoć rainbow tablica (Katz J., 2020).

Kombinacija TLS protokola za sigurnost komunikacije i AES256 sa soli za pohranu biometrijskih podataka osigurava višeslojnu zaštitu sustava. Svi ovi sigurnosni mehanizmi implementirani su kako bi se osiguralo da biometrijski podaci budu zaštićeni, čak i u slučaju naprednih napada. Kako je navedeno u *Computer and Information Security Handbook*, takav slojevit pristup sigurnosti jedan je od najučinkovitijih načina osjetljivih podataka u IoT sustavima, pogotovo kada su u pitanju biometrijske informacije.

3.3. Biometrijska autentifikacija

Biometrijska autentifikacija odnosi se na tehnologije koje omogućuju identifikaciju i verifikaciju osoba na temelju jedinstvenih biometrijskih značajki, poput otiska prsta, prepoznavanja lica, glasa, pa čak i načina hodanja. U ovom radu, biometrijska autentifikacija implementirana je pomoću senzora za otisak prsta. Ova metoda autentifikacije postala je popularna zbog svoje pouzdanosti i sigurnosti u usporedbi s tradicionalnim metodama kao što su lozinka ili PIN.

3.3.1. Sigurnost biometrijske autentifikacije

Sigurnost biometrijske autentifikacije predstavlja ključno pitanje u današnjem svijetu gdje se sve više koristi za zaštitu osjetljivih informacija i resursa. Ova se metoda temelji na prikupljanju i analizi fizičkih ili bihevioralnih karakteristika pojedinca, poput otisaka prstiju, lica, šarenice oka ili glasa.

Prema knjizi *Biometric Systems: Technology, Design and Performance Evaluation*, biometrijska autentifikacija nudi znatno veću razinu sigurnosti od tradicionalnih metoda poput lozinki, koje su podložne zaboravljanju, krađi ili kompromitiranju. Biometrijski podaci su jedinstveni za svaku osobu i teže se repliciraju, čime je smanjen rizik od neovlaštenog pristupa.

Iako biometrijska autentifikacija nudi značajne prednosti u odnosu na tradicionalne metode, ona sa sobom također vuče i određene izazove koji zahtijevaju posebnu pažnju. Jedan od ključnih aspekata je nepovratnost podataka. Naime, biometrijski podaci poput otisaka prstiju ili prepoznavanja lica se ne mogu jednostavno mijenjati kao na primjer lozinke. Iako je to istovremeno i prednost jer su biometrijski podaci trajni, u slučaju kompromitacije, taj podatak se ne može jednostavno zamijeniti kao što bi se mogla promijeniti lozinka. Iz tog razloga presudno je implementirati napredne metode šifriranja i zaštite tih podataka.

Otpornost biometrijskih sustava na različite vrste napada je također važan sigurnosni aspekt. U teoriji, biometrijska autentifikacija je otpornija na napade kao što su brute force napad, phishing ili krađa identiteta, jer bi napadač morao pribaviti fizički pristup biometrijskim podacima osobe. Korištenjem naprednih senzora otisaka prsta, moguće je detektirati pokušaje zloupotrebe ili lažiranja biometrijskih podataka, što dodatno povećava sigurnost sustava.

Otisci prstiju, kao jedan od najčešćih oblika biometrijske autentifikacije, pružaju visoku razinu sigurnosti zahvaljujući svojim jedinstvenim značajkama. Svaki otisak prsta sastoji se

od različitih uzoraka koji su jedinstveni za svaku osobu, što otežava njihovo kopiranje ili repliciranje. Prema istraživanjima, čak ni identični blizanci nemaju potpuno identične otiske prstiju, što ovu metodu čini izuzetno pouzdanom (Jain, 2007).

Jedan od najvažnijih sigurnosnih izazova u biometrijskim sustavima jest zaštita biometrijskih podataka. Prema knjizi *Biometric Systems: Technology, Design and Performance Evaluation,* biometrijski podaci nikada ne bi smjeli biti pohranjeni u svom originalnom obliku. Umjesto toga, koriste se hash funkcije i šifriranje kako bi se osiguralo da se podaci ne mogu kompromitirati. U ovom radu, podaci o otiscima prstiju se šifrirano pohranjuju i uspoređuju s hash kôdovima pohranjenim u sustavu, čime se osigurava dodatna razina zaštite protiv neovlaštenog pristupa.

U sigurnosnim sustavima, jedan od ključnih pokazatelja uspješnosti biometrijske autentifikacije je stopa lažnog prihvaćanja (FAR) i stopa lažnog odbijanja (FRR). FAR predstavlja vjerojatnost da će sustav pogrešno prihvatiti neovlaštenog korisnika, dok FRR predstavlja vjerojatnost da će sustav odbiti ovlaštenog korisnika. Ove dvije stope ključno su balansirane kako bi se osigurala pouzdanost i sigurnost sustava. U kontekstu pametnih brava, FAR i FRR imaju važnu ulogu u osiguravanju da sustav bude dovoljno pouzdan, bez prekomjernih lažnih odbijanja koja bi mogla frustrirati korisnika (Jain, 2007).

Šifriranje i sigurnosni protokoli imaju ključnu ulogu u osiguravanju zaštite biometrijskih podataka tijekom prijenosa između uređaja i sustava za autentifikaciju. U ovom radu korišten je MQTT protokol s TLS-om(Transport Layer Security), koji osigurava siguran prijenos podataka između pametne brave i mobilne aplikacije.

Uz sve ove sigurnosne mjere, biometrijska autentifikacija nudi brojne prednosti, no zahtijeva spretnu implementaciju kako bi se osigurala maksimalna zaštita podataka korisnika.

3.3.2. Implementacija biometrijske tehnologije

Implementacija biometrijske tehnologije, posebno otisaka prstiju, u sustave autentifikacije postala je široko prihvaćena zahvaljujući pouzdanosti i visokoj razini sigurnosti koju ova tehnologija pruža.

Senzori otisaka prstiju koriste se za prikupljanje i analizu jedinstvenih značajki prstiju pojedinca, a te se značajke zatim uspoređuju s prethodno pohranjenim uzorcima kako bi se utvrdilo podudaranje i valjanost identiteta korisnika. Različiti senzori otisaka prsta se koriste kako bi se omogućilo učitavanje i prepoznavanje tih jedinstvenih obrazaca. Postoji nekoliko

vrsta senzora, a svaki od njih primjenjuje različitu tehnologiju kako bi što preciznije očitao otisak prsta (Aratek, 2023).

Optički senzori su među najstarijim tehnologijama u području očitavanja otisaka prstiju. Ovi senzori koriste svjetlo za stvaranje slike otiska prsta. Kada korisnik postavi prst na staklenu površinu, ispod koje se nalazi izvor svjetlosti i kamera, refleksija svjetla s površine prsta bilježi se kako bi se dobila slika otiska. Njihova prednost je što pružaju visoku rezolucije slike i mogu očitati otiske čak i ako osoba ima oštećenu površinu kože, što ih čini korisnima u situacijama gdje je potrebna jednostavna i pouzdana identifikacija kao što su sustavi za kontrolu pristupa u uredima ili drugim javnim prostorima. Iako pružaju visoku razlučivost, optički senzori mogu biti podložni sigurnosnim rizicima jer se oslanjaju na refleksiju svjetla s površine prsta, što znači da se otisak može potencijalno replicirati uz pomoć slike ili maske (Wayman J, 2005) . Senzor za otisak prsta, AS608, je također optički senzor.

Slika 4: Optički senzor (Aratek; 2024; izvor: https://www.aratek.co/news/the-4-fingerprint-sensor-types)

Kapacitivni senzori najčešće se koriste u mobilnim uređajima i predstavljali su standard u biometrijskoj tehnologiji pametnih telefona do prije nekoliko godina. Oni funkcioniraju na princip mjerenja promjena električne vodljivosti između prsta i senzorske površine. Kada prst dodirne senzor, električna se svojstva između točaka na senzoru i površine prsta mijenjaju, omogućujući senzoru da stvori preciznu sliku otiska prsta. Za razliku

od optičkih senzora, kapacitivni senzori ne koriste vizualni prikaz prsta, što otežava pokušaje lažiranja. Glavna smetnja za očitavanje točnosti je vlažan ili prljav prst. Unatoč tom nedostatku, njihova kompaktna veličina, pouzdanost i otpornost na vizualne obmane učinili su ih popularnim rješenjem za biometrijsku autentifikaciju u pametnim telefonima i prijenosnim uređajima (Aratek, 2023).

Slika 5: Kapacitivni senzor (Aratek; 2024; izvor: https://www.aratek.co/news/the-4-fingerprint-sensor-types)

Ultrazvučni senzori koriste zvučne valove kako bi snimili trodimenzionalni prikaz prsta. Prste se postavlja na senzor, a ultrazvučni valovi putuju kroz prst i reflektiraju se natrag prema senzoru, čime se dobiva detaljan prikaz strukture grebena i udubljenja otiska. Budući da ultrazvučni valovi prodiru ispod površine kože, ovi senzori mogu očitati i unutarnje slojeve otiska, čineći ih iznimno sigurnima i otpornima na pokušaje lažiranja. Oni su idealni za primjenu u okruženjima s visokim sigurnosnim zahtjevima, poput vojnih i vladinih ustanova, te u aplikacijama koje zahtijevaju najvišu razinu zaštite. Prednost ultrazvučnih senzora je njihova sposobnost rada i u teškim uvjetima, poput vlage ili prljavštine na prstima, no njihova složena tehnologija ih čini skupljima u usporedbi s drugim vrstama senzora, što ograničava njihovu širu primjenu (Aratek, 2023).

Termički senzori se rijetko koriste u usporedbi s drugim vrstama senzora, ali su korisni po tome što mjere temperaturne razlike između grebena i udubljenja otiska prsta. Grebeni, koji su u izravnom kontaktu sa senzorom, prenose više topline u odnosu na udubljenja, stvarajući temperaturnu mapu otiska. Njihova prednost je niska cijena i otpornost na lažiranje pomoću slika, budući da se oslanjaju na temperaturne razlike između grebena i

udubljenja otiska prsta. Međutim, nedostatak termičkih senzora je što ne pružaju visoku razlučivost u usporedbi s drugim tipovima senzora, što ograničava njihovu primjenu na sustave s nižim sigurnosnim zahtjevima, poput jeftinijih uređaja za kontrolu pristupa ili potrošačke elektronike (Aratek, 2023).

Odabir vrste senzora ovisi o specifičnim zahtjevima aplikacije, sigurnosnim potrebama i financijskim ograničenjima. Svaka tehnologija nudi jedinstvene prednosti, ali i nedostatke, stoga je važno odabrati onaj senzor koji najbolje odgovara određenoj situaciji.

3.3.3.Implementacija AS608 senzora otiska prsta

Senzor otiska prsta AS608 spada u kategoriju optičkih senzora koji koriste svjetlosni izvor za snimanje slike otiska prsta. Ovaj senzor je često korišten u Arduino projektima zbog svoje jednostavnosti implementacije i mogućnosti prilagodbe. Kao što je prethodno spomenuto, glavni princip rada optičkih senzora, pa tako i AS608, temelji se na refleksiji svjetlosti s prsta postavljenog na površini senzora.Ovaj pristup omogućuje precizno snimanje detalja otiska, a zatim i usporedbu s unaprijed pohranjenim biometrijskim uzorcima kako bi se potvrdio identitet korisnika.

Slika 6: AS608 senzor otiska prsta (Electropeak; 2024; izvor: https://electropeak.com/as608-optical-fingerprint-reader-module-sensor)

AS608 ima sposobnost pohranjivanja do 128 otisaka prstiju u svoju internu memoriju, što ga čini izuzetno korisnim za promjene gdje je potreban veći broj korisnika, kao što su pametne brave ili kontrola pristupa. Senzor komunicira s mikrokontrolerom putem serijske komunikacije, što olakšava njegovu integraciju u različite IoT projekte. Ovaj serijski protokol omogućava slanje i primanje naredbi za prepoznavanje i pohranu otisaka prstiju. Uz jednostavnu implementaciju, AS608 nudi visok stupanj prilagodljivosti, posebno u sustavima gdje je potrebno upravljanje na daljinu putem mobilnih aplikacija ili web sučelja. Kada je pak riječ o sigurnosti pohrane podataka, AS608 nudi osnovnu zaštitu jer pohranjuje podatke o otiscima prstiju direktno u svoju memoriju, umjesto da ih šalje na vanjske poslužitelje. Pohranjeni podaci nisu slike otisaka, već su hashirane matematičke reprezentacije, što dodatno osigurava da biometrijski podaci nisu lako dostupni i podložni zloupotrebi. Ovo je vrlo bitno jer čak i da dođe do neovlaštenog pristupa senzoru, stvari podaci o otiscima prstiju se ne mogu rekonstruirati (Shojaei, 2023).

Iako senzor nudi osnovnu razinu sigurnosti, potrebne su dodatne mjere kako bi se osigurala potpuna zaštita biometrijskih podataka. Ključna tehnika koja se koristi u takvim sustavima su enkripcija podataka između senzora i kontrolera, kao i korištenje sigurnih protokola za prijenos podataka, poput MQTT-a i TLS enkripcije u našem slučaju. Enkripcija podataka osigurava da se biometrijski podaci ne mogu presresti ili kompromitirati tijekom prijenosa. Osim toga, hashiranje podataka omogućava dodatnu razinu zaštite, jer čak i u slučaju kompromitacije podataka, stvari podaci o otisku nisu direktno dostupni.

Što se tiče modula pametnih brava, AS608 senzor može poslužiti kao ključna komponenta sigurnosnog sustava. Njegova sposobnost prepoznavanja i pohrane otisaka prstiju izravno na uređaju, zajedno s dodatnom sigurnosnom mjerom kao što je enkripcija i sigurnosni protokol, čine ga odličnim izborom za rad u IoT okruženju.

3.4. Android mobilna aplikacija

Aplikacija za upravljanje pametnom bravom razvijena je u programskom jeziku Kotlin koristeći Android Studio razvojno okruženje. Fokusa razvoja je bio na sigurnom, fleksibilnom i intuitivnom korisničkom sučelju koje omogućuje upravljanje pristupom putem pametne brave. Aplikacija korisniku pruža mogućnost otključavanja i zaključavanja vrata, dodavanje novih uređaja i registraciju novih otisaka prstiju.

3.4.1. Arhitektura aplikacije

Mobilna aplikacija koristi MVVM (Model-View-ViewModel) arhitekturu koja se u posljednje vrijeme nametnula kao preferirana arhitektura u razvoju mobilnih aplikacija zbog svoje modularnosti, lakoće testiranja i održavanja. Ovaj arhitektonski obrazac omogućuje odvajanje odgovornosti između slojeva aplikacije, što pridonosi boljoj organizaciji kôda, njegovoj skalabilnosti i testiranju. Prema knjizi *How to build Android apps with Kotlin*, MVVM je posebno učinkovit u aplikacijama koje zahtijevaju dinamičko ažuriranje korisničkog sučelja i rad s više izvora podataka jer odvaja poslovnu logiku od prikaza podataka.

Model u MVVM arhitekturi upravlja podacima i poslovnom logikom aplikacije. U ovom projektu, Model predstavlja podatke o pametnoj bravi, uređajima i otiscima prstiju. Ove podatke aplikacija prikuplja putem MQTT komunikacije. Uloga Modela je osigurati da podaci budu dostupni ViewModelu, koji ih zatim dostavlja View-u, bez da aplikacija izravno manipulira podacima unutar View-a (Forrester, 2021).

View je sloj koji predstavlja korisničko sučelje i sadrži sve komponente koje korisnik može vidjeti i s kojima može stupiti u interakciju. U ovoj aplikaciji, View uključuje gumb za otključavanje i zaključavanje brave, popis povezanih uređaja ili registriranih prstiju te opciju za dodavanjem novog otiska prsta ili uređaja. View također obrađuje interakcije korisnika, kao što je pritisak gumba ili odabir iz popisa uređaja ili otisaka prstiju.

ViewModel služi kao posrednik između View-a i Modela, osiguravajući da podaci iz Modela budu pravilno prikazani u View-u, ali i da se promjene koje se događaju u View-u (npr. Otključavanje brave) reflektiraju u Modelu. Ovaj obrazac sprječava prekomjerno zagušivanje aktivnosti ili fragmenta samom logikom aplikacije (Forrester, 2021).

Prema knjizi *How to build Android apps with Kotlin,* jedna od prednosti MVVM arhitekture je u tome što se omogućuje ponovno korištenje kôda, odnosno njena modularnost. ViewModeli su neovisni o View komponentama i ne komuniciraju izravno s View-om, čime se osigurava da ViewModel može biti testiran bez ovisnosti o složenim korisničkim sučeljima. U kontekstu aplikacije iz ovog rada, to znači da se ViewModel može testirati i koristiti neovisno o različitim korisničkim sučeljima (npr. različitim fragmentima za upravljanje uređajima i otiscima prstiju).

Android aplikacije koriste fragmente kako bi bile modularnije i fleksibilnije. Oni se koriste za prikaz različitih funkcionalnosti aplikacije, kao što su prikaz popisa uređaja, popis otisaka prstiju i početnog ekrana s gumbom za upravljanje bravom. Prednost korištenja fragmenata u odnosu na aktivnosti je što fragmenti omogućuju višekratno korištenje istih dijelova korisničkog sučelja, čime se smanjuje složenost aplikacije. Fragmenti su optimalni

za aplikacije koje koriste više prikaza i trebaju se dinamički prilagođavati korisničkim potrebama (Forrester, 2021).

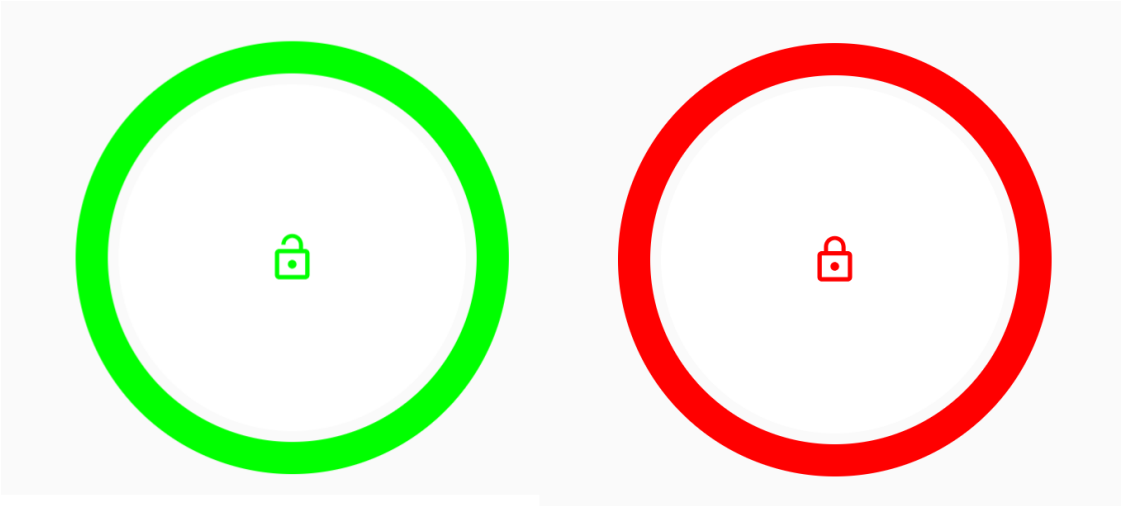
Komunikacija između mobilne aplikacije i pametne brave odvija se putem MQTT protokola koji je posebno dizajniran za IoT sustave. MQTT protokol integriran je s TLS enkripcijom kako bi se osigurala zaštita prijenosa podataka. Prema Forresteru, korištenje MQTT-a u aplikacijama za pametne uređaje ključno je zbog njegovih prednosti u minimiziranju potrošnje energije i učinkovitog upravljanja mrežnim resursima.

U zaključku, MVVM arhitektura pruža čvrstu osnovu za razvoj modularnih i skalabilnih Android aplikacija. Primjenom ove arhitekture, razvoj aplikacije za pametnu bravu omogućio je jasno odvajanje odgovornosti između poslovne logike, korisničkog sučelja i upravljanja podacima, čime se osigurala održivost i skalabilnost aplikacije u budućem razvoju.

3.4.2. Korisničko sučelje

Korisničko sučelje aplikacije za upravljanje pametnom bravom dizajnirano je s naglaskom na jednostavnost, intuitivnost i funkcionalnost. Glavni cilj pri dizajnu korisničkog sučelja bio je omogućiti korisnicima jednostavno i brzo upravljanje pametnom bravom, uz pružanje jasnih povratnih informacija o trenutnom statusu sustava. Prema istraživanjima, jednostavna i čista korisnička sučelja potiču bolju interakciju korisnika s aplikacijama, osobito u kontekstu pametnih uređaja, gdje je potrebno minimizirati složenost kako bi se osigurao nesmetan protok radnih tokova (Forrester, 2021). Imajući to na umu, koristi se minimalistički dizajn s fokusom na funkcionalnost i jasnu navigaciju.

Glavni ekran aplikacije dizajniran je tako da sadrži ključne funkcionalnosti pametne brave, omogućujući korisniku upravljanje osnovnim opcijama bez potrebe za dubokim istraživanjem aplikacije. Na početku, korisnik je dočekan ekranom koji sadrži veliki bijeli gumb s obrubom u boji koji pokazuje trenutno stanje brave. Ukoliko je brava otvorena, gumb ima zeleni obrub i zeleni otvoreni lokot u sredini gumba, a ukoliko je brava zatvorena, gumb ima crveni obrub i crveni zatvoreni lokot u sredini. Prikaz stanja brave omogućuje korisniku trenutan uvid u status uređaja i osigurava intuitivan način za promjenu tog stanja.



Slika 7: Stanja gumba za pametnu bravu (Izvor: Slika autora, 2024)

Fokus je stavljen na to da korisničko sučelje bude što jednostavnije, a sam gumb predstavlja element oko kojeg su grupiranje dodatne funkcionalnosti. Na primjer, kada korisnik pritisne gumb za otključavanje ili zaključavanje, vrata se otvaraju ili zatvaraju, a gumb se dinamički mijenja kako bi vizualno reflektirao promjenu stanja. Ovaj način interakcije koristi se za pružanje jednostavnog iskustva, a prema istraživanjima, upotreba dinamičkih elemenata u sučelju poboljšava korisničku percepciju responzivnosti i funkcionalnosti aplikacije.

Ispod glavnog gumba, nalaze se dva gumba: „Uređaji“ i „Otisci“. Ove opcije omogućuju korisniku pristup dodatnim funkcijama, kao što je upravljanje povezanim uređajima i otiscima prstiju koji su registrirani u sustavu. Klikom na „Uređaji“ prikazuje se popis povezanih uređaja ispod gumba „Uređaji“, a iznad gumba „Otisci“. Ovdje korisnik može dodavati nove uređaje, brisati postojeće ili uređivati imena već registriranih uređaja. Svaki uređaj se dodaje na način da aplikacija komunicira s NodeMCU-om putem MQTT-a, pri čemu je potrebno unijeti nasumično generirani kôd koji se prikazuje na LCD ekranu pametne brave. Ovaj kôd osigurava sigurno uparivanje uređaja i smanjuje mogućnost neovlaštenog dodavanja uređaja. Ako korisnik klikne na opciju „Otisci“, otvara se ekran koji prikazuje sve registrirane otiske prstiju. U ovom ekranu korisnik može dodavati nove otiske, uređivati postojeće ili brisati nevažeće otiske. Proces dodavanja novog otiska započinje pritiskom na gumb „Dodaj novi otisak“, pri čemu se aktivira biometrijski senzor AS608, koji provodi proces registracije novog otiska.

Prema preporukama za dizajn mobilnih aplikacija, korisničko sučelje mora biti prilagođeno korisniku i prilagođeno raznim scenarijima uporabe (Material Design Guidelines,

2024). U ovom projektu, dizajn sučelja slijedi pravila koja omogućuju dosljedno korisničko iskustvo na različitim Android uređajima. Korisničko sučelje koristi komponente iz Android Jetpack biblioteka, uključujući RecyclerView za dinamičko prikazivanje popisa uređaja i otisaka, te ViewModel za upravljanje podacima i osiguranje da se podaci pravilno prikazuju i ažuriraju u sučelju. Korištenje RecyclerView-a omogućuje jednostavno upravljanje popisima i pruža mogućnost skaliranja u slučaju da korisnik poveže veći broj uređaja ili registrira više otisaka prstiju. Prema knjizi *How to build Android apps with Kotlin*, korištenje komponenti kao što su RecyclerView i ViewModel poboljšava performanse i osigurava glatko korisničko iskustvo čak i kod složenih interakcija s podacima.

3.4.3. Sigurnosni aspekti aplikacije

Sigurnost je ključni prioritet u razvoju aplikacije koja upravlja pametnom bravom, osobito zbog prirode uređaja i njegove uloge u zaštiti pristupa fizičkom prostoru. Aplikacija koristi nekoliko sigurnosnih mehanizama kako bi osigurala da svi podaci koji se prenose između mobilne aplikacije i pametne brave budu zaštićeni od presretanja i neovlaštenog pristupa.

Glavni sigurnosni mehanizam koji se aplikacija oslanja je TLS protokol za šifriranje komunikacije između aplikacije i kontrolera. TLS omogućuje siguran prijenos podataka putem zaštićenog kanala, sprječavajući bilo kakvo presretanje ili izmjenu podataka tijekom prijenosa. Ovaj protokol osigurava enkripciju svih komandi, poput naredbi za otključavanje ili zaključavanje vrata, kao i razmjenu podataka vezanih uz dodavanje novih uređaja ili verifikaciju korisnika. Time je osigurana cjelokupna komunikacija, čime se jamči da samo ovlašteni uređaji i korisnici mogu komunicirati sa sustavom pametne brave.

Korisnički podaci, poput korisničkog imena i lozinke, koji se koriste za povezivanje s kontrolerom preko TLS-a, pohranjuju se u shared preferences unutar aplikacije. Ova lokalna pohrana omogućuje da aplikacija jednostavno i sigurno pristupi potrebnim podacima za autentifikaciju prilikom uspostavljanja sigurne veze. Podaci u shared preferences nisu lako dostupni drugim aplikacijama, čime se dodatno smanjuje rizik od curenja osjetljivih informacija.

Također je važno naglasiti da aplikacije ne pohranjuje niti prenosi biometrijske podatke. Sve operacije vezane uz biometrijske podatke, uključujući njihovu pohranu i obradu, odvijaju se na senzoru AS608. Svi biometrijski podaci pohranjuju se lokalno na senzoru, što smanjuje rizik od potencijalnog curenja informacija tijekom prijenosa. Aplikacija samo koristi sučelje za upravljanje procesom registracije ili verifikacije, bez direktnog pristupa

biometrijskim podacima. Drugim riječima, pametnu bravu nije moguće otvoriti putem senzora za otisak prsta na mobitelu.

Jedna od sigurnosnih mjera unutar sustava je i upotreba nasumičnog broja za povezivanje novih uređaja. Kada korisnik pokuša dodati novi uređaj, mikrokontroler prikazuje nasumično generirani broj na LCD ekranu. Korisnik mora unijeti taj broj u aplikaciju kako bi uspostavio sigurnu vezu između uređaja i sustava. Oba broja su poslana na MQTT, te se povežu samo ako korisnik upiše isti broj koji je mikrokontroler generirao.

Korištenje TLS-a u kombinaciji s lokalnim pohranjivanjem osjetljivih podataka poput korisničkog imena značajno smanjuje rizik od presretanja i neovlaštenog pristupa u IoT sustavima. Ova arhitektura pruža pouzdano i sigurno okruženje za prijenos i obradu podataka unutar sustava pametnih uređaja.

4. Implementacija sustava pametne brave

U sklopu diplomskog rada, cilj je bio razviti cjeloviti sustav pametne brave koji se oslanja na biometrijske podatke za autentifikaciju korisnika i omogućava daljinsko upravljanje putem mobilne aplikacije. Razvoj pametne brave uključuje integraciju više tehnoloških komponenti, uključujući hardverske module poput senzora otiska prsta, solenoidne brave i LCD ekrana, kao i softverske aspekte poput mobilne aplikacije i različite vrste sigurnosnih aspekata. S obzirom na rastući interes za sustavima pametnih kuća i sve većom potrebom za povećanjem sigurnosti, ovakav projekt ima široku primjenu, kako u privatnim, tako i u poslovnim prostorima.

Pametne brave pružaju dodatni sloj sigurnosti jer omogućuju kontrolu pristupa korištenjem biometrijskih podataka koji su teži za falsifikaciju u odnosu na tradicionalne ključeve ili šifre. Ovaj sustav koristi biometrijsku tehnologiju uz pomoć senzora za otisak prsta AS608, čime se osigurava jednostavna i sigurna autentifikacija korisnika. Korištenjem solenoidne brave, koja se otključava i zaključava na temelju prepoznatog otiska prsta, eliminira se potreba za fizičkim ključevima, što također smanjuje rizik od krađe ili gubitka ključeva. Osim toga, sustav uključuje i mogućnost daljinskog upravljanja, čime korisnici mogu otvoriti i zatvoriti vrata preko mobilne aplikacije. Korištenjem MQTT protokola, omogućena je sigurna komunikacija između kontrolera pametne brave i mobilne aplikacije. Daljinsko upravljanje pruža korisnicima dodatnu razinu fleksibilnosti, omogućujući im da kontroliraju pristup svojem domu ili poslovnom prostoru bez obzira na fizičku udaljenost.

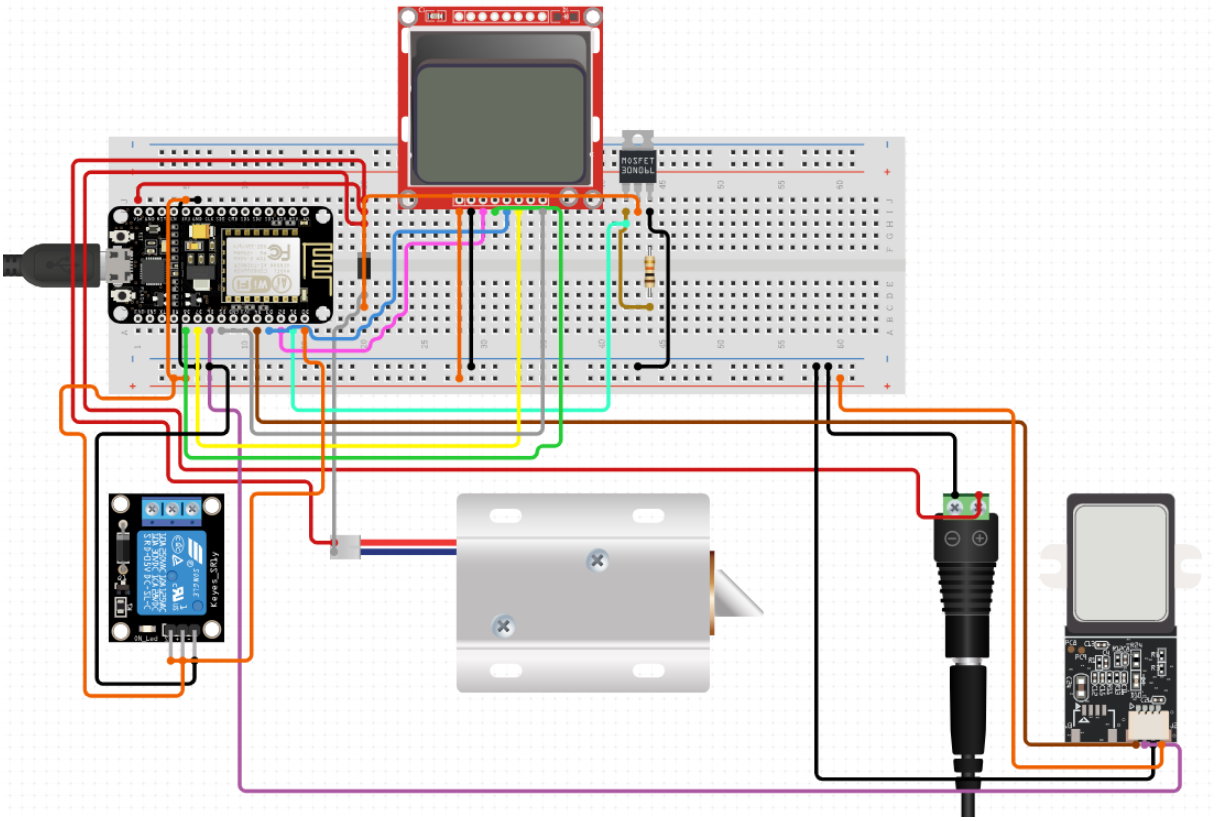
Jedan od ključnih komponenti ovog projekta je osiguravanje sigurnosti podataka, posebice biometrijskih podataka. Implementacijom TLS enkripcije unutar MQTT komunikacije, osigurano je da svi podaci koji se razmjenjuju između aplikacije i kontrolera budu zaštićeni od neovlaštenog pristupa. Dodatno, otisci prstiju su kriptirani koristeći AES-256 sa soli, što dodatno povećava razinu zaštite biometrijskih podataka. Drugim riječima, biometrijski podaci nisu lako dostupni čak ni ako se fizički pristupi uređaju, što čini cijeli sustav izuzetno sigurnim.

Korisnost ovakvog sustava leži u njegovoj primjeni u svakodnevnim situacijama gdje je potreban pouzdan i siguran način autentifikacije i kontrole pristupa. Uvođenjem biometrijske autentifikacije i daljinskog upravljanja, pametna brava postaje idealno rješenje za moderne pametne kuće, kuće za odmor, ali i poslovne prostore. Osim sigurnosti, sustav nudi praktičnost jer eliminira potrebu za fizičkim ključevima, čime se smanjuju rizici povezani s ključevima.

U narednim odjeljcima bit će detaljno opisani svi aspekti implementacije, kao što su hardverske komponente sustava, razvoj samog rješenja, razvoj mobilne aplikacije i primjena sigurnosnih mjera.

4.1. Integracija hardverskih komponenti

U implementaciji sustava pametne brave, komponente su međusobno povezane putem NodeMCU mikrokontrolera koji služi kao središnji dio sustava.



Slika 8: Shema spojenih komponenti (Izrada autora preko circuito; 2024; izvor:https://www.circuito.io/app?components=10168,10218,11792,360216,842876,3061987)

NodeMCU je spojen s eksperimentalnom pločicom, koja omogućava jednostavno povezivanje i napajanje više komponenti. Eksperimentalna pločica, kao osnovni alat, služi za distribuciju napajanja te povezivanje različitih dijelova sustava bez potrebe za složenim lemljenjem ili trajnim spajanjima. Kao što je vidljivo na slici, NodeMCU ne koristi samo svoje naponske i zemljane pinove, već je povezan i putem drugih pinova koji omogućuju komunikaciju s ključnim elementima sustava. Na eksperimentalnoj pločici, putem naponske

(3v3) i zemljane (GND) linije, povezane su komponente poput LCD ekrana, relaja i AS608 senzora. Ove linije osiguravaju stabilno napajanje za sve komponente, dok NodeMCU preko svojih drugih digitalnih pinova uspostavlja komunikaciju s njima. Konkretno, AS608 senzor otiska prsta je povezan s NodeMCU-om putem D1 i D2 pinova, što omogućava dvosmjernu komunikaciju između senzora i mikrokontrolera, što je nužno za čitanje biometrijskih podataka i upravljanje bravom.

Pin D3 na NodeMCU-u koristi se za kontrolu releja, koji upravlja napajanjem solenoidne brave. Relej služi kao sklopka koja omogućuje uključivanje ili isključivanje solenoidne brave na temelju signala primljenog s NodeMCU-a. Pritom je solenoidna brava direktno spojena na bateriju od 9V, koja osigurava njezino napajanje, dok je relej spojen između brave i baterije te upravlja protokom struje. LCD ekran je povezan na NodeMCU putem D5 i D6 pinova. Ova veza omogućuje kontroleru slanje podataka na ekran, kao što su poruke o statusu sustava, nasumično generirani broj za povezivanje s mobilnom aplikacijom i obavijesti o uspjehu ili neuspjehu autentifikacije otiska prsta.

Spajanje preko eksperimentalne pločice omogućuje fleksibilnost u radu, a korištenje žica za povezivanje komponenti sa svakom od linija osigurava jednostavnu zamjenu ili nadogradnju elemenata sustava. Na slici je vidljivo kako su sve komponente povezane putem žica koje omogućuju neometanu distribuciju struje i signalizaciju između kontrolera i senzora u sustavu. Ovaj način povezivanja omogućuje učinkovitu integraciju svih elemenat u funkcionalnu cjelinu.

Nakon što je opisano kako su igle povezane, potrebno je isto definirati u samom programskom rješenju, a to izgleda ovako:

```
#define relayPin D3
#define rxPin D1
#define txPin D2
#define lcdPin1 D5
#define lcdPin2 D6
SoftwareSerial mySerial(rxPin, txPin);
Adafruit_Fingerprint finger = Adafruit_Fingerprint(&mySerial);
```

Kôd predstavlja definiciju pinova te su postavljeni na svoja mjesto onako kako je prikazano na slici. Osim definiranja igla, definira se *SoftwareSerial* za senzor otiska prsta, te posljednja linija predstavlja kreiranje instance Adafruit Fingerprint senzora. No prije definicije igla na kontroleru, potrebno je uključiti biblioteke koje se koriste, a one se dodaju tako da se u ArduinoIDE odaberu biblioteke, pronađe se biblioteka koju želimo koristiti i pritisnemo „Install“. Nakon toga, možemo ju dodati u program na sljedeći način:

```
#include <ESP8266WiFi.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
<Adafruit_SSD1306.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
#include <AESLib.h>
#include <Adafruit_Fingerprint.h>
#include <ArduinoJson.h>
#include <SoftwareSerial.h>
```

Biblioteka *ESP8266WiFi* služi za Wi-Fi povezivanje kako bi komunikacija na MQTT protokolu bila moguća. Sljedeća biblioteka jest *Wire* koja je potrebna za I2C komunikaciju s ekranom. *Adafruit_SSD106* biblioteka je biblioteka za specifične ekrane, dok je *Adafruit_GFX* grafička biblioteka koja je također potrebna za rad s ekranom. Sljedeća biblioteka, *WiFiClientSecure* služi za TLS/SSL podršku kako bi podaci između kontrolera i mobilne aplikaciji bili zaštićeni. Kada govorimo o komunikaciji u IoT sustavu, ne možemo izbjeći MQTT protokol, te smo ga povezali s našim kôdom u 6. liniji, odnosno bibliotekom *PubSubClient*. Kako bi spremanje otisaka prsta bilo što sigurnije, potrebna je enkripcija, a nju smo dobili dodavanjem *AESLib* biblioteke. Za sam rad sa senzorom, odnosno, AS608 modulom, potrebna je biblioteka naziva *Adafruit_Fingerprint*. Osim toga, za softversku serijsku komunikaciju koristio sam *SoftwareSerial*. Posljednja biblioteka koja je bila korištena jest *ArduinoJson* koja omogućava čitanje i pisanje poruka u JSON formatu koje završe u MQTT protokolu.

Nakon uvođenja biblioteka i definiranja koje komponente koriste koju iglu, potrebno je spojiti kontroler na Internet, ali i MQTT.

```
const char* ssid = "naziv_wifi_mreže";
const char* password = "šifra_wifi_mreže";
const char* mqtt_server = "test.mosquitto.org";
const int mqtt_port = 8883;

WiFiClientSecure espClient;
PubSubClient client(espClient);

void setup_wifi() {
delay(10);
Serial.print("Povezivanje na ");
Serial.println(ssid);
```

```
WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}
Serial.println("WiFi povezano");
}
```

Prvih 4 linije kôda predstavljaju deklariranje MQTT i WiFi podataka. Podaci kao ssid i password predstavljaju naziv i lozinku WiFi mreže na koju se želimo spojiti. Nakon što se spojimo na WiFi, možemo se spojiti i na MQTT te smo u ovom projektu odabrali server koji piše u kôdu, te je port postavljen na 8883 što ukazuje da se koristi MQTT preko TLS/SSL jer je to standardni port za MQTT preko sigurnog kanala. Kao što je već rečeno, taj port omogućuje da se komunikacija odvija preko enkriptiranog kanala, što osigurava privatnost podataka tijekom prijenosa. Kreirana je varijabla *espClient* koristeći klasu *WiFiClientSecure,* što omogućava sigurne veze putem TLS-a. Zatim je definiran objekt *client* koristeći klasu *PubSubClient*, koja se koristi za komunikaciju s MQTT brokerom. Funkcija *setup_wifi()* zadužena je za spajanje NodeMCU uređaja na Wi-Fi mrežu. Unutar te funkcije, nakon kratkog odgađanja, ispisuje se poruka „Povezivane na“ zajedno s nazivom mreže. Nakon toga započinje funkcija *WiFi.begin(),* koja započinje postupak povezivanja s Wi-Fi mrežom koristeći uneseni ssid i lozinku. Petlja while koristi se za čekanje dok se NodeMCU ne poveže s Wi-Fi mrežom. Svakih 500 milisekundi ispisuje se točka na serijski monitor kako bi korisnik znao da je povezivanje u tijeku. Kada je veza uspješno uspostavljena, ispisuje se poruka „Wifi povezano“, što potvrđuje da je NodeMCU spojen na mrežu i spreman je za daljnju komunikaciju putem MQTT protokola.

```
byte aes_key[] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                   0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
                   0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
                   0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F };

byte iv[16] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
               0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F};

void generateSalt(byte* salt, int length) {
  for (int i = 0; i < length; i++) {
    salt[i] = random(0, 256);
```

```
  }
}

int addPadding(char* input, int length) {
  int paddingSize = 16 - (length % 16);
  for (int i = length; i < length + paddingSize; i++) {
    input[i] = paddingSize;
  }
  return length + paddingSize;
}
```

Ovaj dio kôda je osmišljen kako bi osigurao sigurnu enkripciju podataka koristeći AES-256 algoritam u kombinaciji s inicijalizacijskim vektorom i soli. U kôdu se definira 256-bitni AES ključ pomoću niza *aes_key[]*, koji sadrži 32 bajta predstavljena heksadecimalnim vrijednostima. Ovaj niz služi kao enkripcijski ključ, a osigurava 256-bitnu zaštitu. Ključ mora ostati povjerljiv i ne smije biti pohranjen na nesigurnim mjestima. Inicijalizacijski vektor se koristi za dodavanje slučajnosti enkripcijskom procesu, kako bi se osiguralo da isti ulazni podaci enkriptirani istim ključem uvijek rezultiraju različitim izlazom. On se definira kao niz od 16 bajtova, te mora biti jedinstven za svaku enkripciju, iako ne mora biti tajan. Najčešće se prenosi zajedno s enkriptiranim podacima, jer je potreban za dekripciju. Sol se koristi kao dodatni mehanizam sigurnosti za sprečavanje napada unaprijed pripremljenim tablicama i osigurava da isti podaci, kada se više puta enkriptiraju, daju različit rezultat. Funkcija *generateSalt()* generira nasumičnu veličinu soli definiranu parametrom *length*. Unutar petlje, svaki element niza se popunjava slučajnim brojem između 0 i 255 koristeći funkciju *random()*. AES algoritam radi u blokovima od 16 bajtova, što znači da ulazni podaci moraju biti višekratnici od 16 bajtova. U slučaju da podaci nisu te duljine, potrebno je dodati padding kako bi se ispunili zahtjevi blok veličine. Funkcija *addPading()* upravo i implementira taj postupak. Ona izračunava veličinu potrebnog paddinga tako da zbroji razliku između duljine trenutnih podataka i najbližeg višekratnika broja 16. Svaki dodani bajt ispunjava se vrijednošću koja odgovara veličini paddinga, što omogućuje kasnije uklanjanje paddinga nakon dekripcije.

```
int encryptFingerprintData(char* data, int dataLength, byte* salt, char*
encryptedData) {
  for (int i = 0; i < 4; i++) {
    data[dataLength + i] = salt[i];
  }

  int paddedLength = addPadding(data, dataLength + 4);
```

```
  aesLib.encrypt(encryptedData, data, paddedLength, aes_key, 256, iv);

  return paddedLength;
}
```

Prethodno napisani dio kôda implementira funkciju *encryptFingerprintData*, koja je zadužena za enkripciju podataka o otisku prsta. Funkcija koristi sol za dodatnu zaštitu podataka, a nakon toga enkriptira podatke pomoću AES-256 algoritma u CBC načinu rada. U *for* petlji se dodaje sol na kraj neenkriptiranih podataka. Kopira se svaki bajt soli na kraj niza *data,* počevši od indeksa koji odgovara trenutnoj duljini podataka (*datalength).* Nakon što su podaci pripremljeni dodavanjem soli i padding-a, dolazi do same enkripcije. Ovdje se koristi AES_256 enkripcija u CBC načinu rada. Funkcija vraća duljinu enkriptiranih podataka, koja je jednaka duljini podataka nakon dodavanja padding-a. Ova se duljina koristi prilikom pohrane ili slanja enkriptiranih podataka. Nakon implementacije enkripcije, potrebno je registrirati otisak prsta.

```
void enrollFingerprint() {
  int id = 1;
  display.clearDisplay();
  display.setCursor(0, 0);
  display.println("Registracija otiska ID:");
  display.print(id);
  display.display();

  while (finger.getImage() != FINGERPRINT_OK) {
    display.clearDisplay();
    display.setCursor(0, 0);
    display.println("Stavite prst");
    display.display();
    delay(1000);
  }

  if (finger.image2Tz(1) != FINGERPRINT_OK) {
    display.println("Greška pretvorbe");
    display.display();
    return;
  }

  display.println("Uklonite prst");
```

```
  display.display();
  delay(2000);

  while (finger.getImage() != FINGERPRINT_OK) {
    display.clearDisplay();
    display.setCursor(0, 0);
    display.println("Ponovno prst");
    display.display();
    delay(1000);
  }

  if (finger.image2Tz(2) != FINGERPRINT_OK) {
    display.println("Greška pretvorbe");
    display.display();
    return;
  }

  if (finger.createModel() != FINGERPRINT_OK) {
    display.println("Greška kreiranja");
    display.display();
    return;
  }

  byte salt[4];
  generateSalt(salt, sizeof(salt));

  char dataToEncrypt[32];
  sprintf(dataToEncrypt, "Fingerprint ID: %d", id);

  char encryptedData[48];
  int encryptedLength = encryptFingerprintData(dataToEncrypt,
strlen(dataToEncrypt), salt, encryptedData);

  display.clearDisplay();
  display.setCursor(0, 0);
  display.println("Podaci enkriptirani");
  display.display();

  if (finger.storeModel(id) == FINGERPRINT_OK) {
    display.println("Otisak pohranjen");
```

```
  } else {
    display.println("Greška pohrane");
  }
  display.display();
}
```

Ovaj dio programskog rješenja implementira proces registracije otiska prsta na uređaju, te uključuje korake za enkripciju podataka o otisku prije njegove pohrane na senzor. Funkcija *enrollFingerprint* provodi nekoliko koraka: skeniranje otiska prsta, pretvaranje slike u uzorak, kreiranje modela i pohrana otiska prsta.

Funkcija započinje s ID-om za otisak prsta koji se registrira. Na LCD zaslonu se prikazuje poruka „ Registracija otiska ID: „ nakon čega slijedi ID koji se trenutno registrira. Ovo korisniku daje vizualni prikaz trenutnog statusa procesa registracije. Petlja zatim čeka da korisnik stavi prst na senzor. Funkcija *finger.getImage()* pokušava uhvatiti sliku otiska prsta. Ako slika nije uspješno uhvaćena, zaslon će kontinuirano prikazivati poruku „Stavite prst“. Nakon što je otisak uspješno skeniran, prelazi se na sljedeći korak, a to je pretvaranje slike u uzorak. Funkcija *finger.image2Tz(1)* pretvara sliku otiska prsta u uzorak koji senzor može analizirati. Ako dođe do pogreške u ovom procesu, korisnik se o istom obavještava, a funkcija se prekida. Nakon što je otisak skeniran i pretvoren, korisniku se prikazuje poruka da ukloni prst, nakon čega se cijeli postupak skeniranja i pretvaranja ponavlja po drugi put. Nakon uspješnog skeniranja oba otiska prsta, senzor stvara model otiska kombiniranjem oba uzorka pomoću *finger.createModel()*. Ako model ne uspije kreirati model, funkcija ispisuje poruku o grešci. Nakon što se model uspješno kreira, isti se kriptira, te se rezultat pohranjuje u *encryptedData*. Nakon što su podaci enkriptirani, korisnik je obaviješten da je proces dovršen. Funkcija *finger.storeModel(id)*pohranjuje kreirani model otiska prsta u senzor pod navedenim ID-om. Nakon uspješne pohrane, prikazuje se poruka „Otisak pohranjen“, odnosno prikazuje se „Greška pohrane“ u suprotnom. Sada kada imamo registrirani otisak prsta, možemo ga koristiti za otvaranje vrata.

```
void openDoor() {
  display.clearDisplay();
  display.setCursor(0, 0);
  display.println("Stavite prst za provjeru");
  display.display();

  while (finger.getImage() != FINGERPRINT_OK) {
    delay(1000);
  }
```

```
  if (finger.image2Tz() != FINGERPRINT_OK) {
    display.clearDisplay();
    display.setCursor(0, 0);
    display.println("Greška pretvorbe");
    display.display();
    return;
  }
  if (finger.fingerFastSearch() != FINGERPRINT_OK) {
    display.clearDisplay();
    display.setCursor(0, 0);
    display.println("Otisak nije pronađen");
    display.display();
    return;
  }

  display.clearDisplay();
  display.setCursor(0, 0);
  display.println("Otisak pronađen");
  display.println("Otključavanje vrata...");
  display.display();

  digitalWrite(relayPin, HIGH);
  delay(5000);
  digitalWrite(relayPin, LOW);

  display.clearDisplay();
  display.setCursor(0, 0);
  display.println("Vrata zaključana");
  display.display();
}
```

Ovaj dio kôda implementira funkciju *openDoor*, koja provodi autentifikaciju korisnika pomoću senzora za otisak prsta, a nakon uspješne identifikacije otvara vrata kontrolom releja. Proces uključuje skeniranje otiska prsta, provjeru podudarnosti sa spremljenim otiscima i aktiviranje releja za otključavanje vrata. Na samom početku, LCD ekran prikazuje poruku „Stavite prst za provjeru“ kako bi obavijestio korisnika na radnju radi provjere. Naredba *clearDisplay()* briše prethodni sadržaj, dok *setCursor()* postavlja pokazivač za ispis teksta na odgovarajuće mjesto na zaslonu, u našem slučaju, gornji lijevi kut. Funkcija *finger.getImage()* kontinuirano pokušava uhvatiti sliku otiska prsta s uređaja. Ako otisak nije

uspješno skeniran, funkcija ponovno pokušava nakon 1000 milisekundi, odnosno 1 sekunde, sve dok ne dođe do uspjeha. Nakon što je slika otiska uspješno skenirana, koristi se funkcija *finger.image2Tz()* koja pretvara sliku otiska u karakteristike koje senzor može koristiti za pretraživanje pohranjenih otisaka. Ako dođe do pogreške tijekom pretvorbe, ispisuje se poruka „Greška pretvorbe“ na zaslonu i funkcija se prekida pomoću *return*. Ako je pretvorba uspješna funkcija *finger.fingerFastSearch()* pretražuje senzor za pohranjene otiske prsta kako bi pronašla podudaranje s trenutno skeniranim otiskom. Ako podudaranje nije pronađeno, prikazuje se poruka „Otisak nije pronađen“ i funkcija se prekida. Nakon što senzor pronađe podudarni otisak prsta, na zaslonu se prikazuje poruka „Otisak pronađen“ i „Otključavanje vrata…“. Na taj način korisnik je obaviješten da je otisak prsta pronađen i da može ući u prostor. Relej koji kontrolira solenoidnu bravu aktivira se pomoću *digitalWrite(relayPin, High),*što znači da se šalje signal za otključavanje vrata. Funkcija *delay(5000)* održava vrata otključanima 5 sekundi, nakon čega se relej deaktivira pomoću *digitalWrite(relayPin, Low)*. Nakon što se vrata zaključaju, na zaslonu se prikazuje poruka „Vrata zaključana“.

Za otključavanje vrata putem mobilne aplikacije, čita poruke u temi „Gataric“, te kada dođe poruka „Action: open door“, vrata se otvaraju samo na način da relaj otvori vrata, te ih u ovom slučaju drži otvorenima 30 sekundi. Kako se većina kôda ponavlja, prikaz razgovora s MQTT protokol prikazat ću na primjeru slanja nasumičnog broja za dodavanje uređaja u mobilnu aplikaciju.

```
String generateRandomCode() {
  int randomCode = random(0, 10000);
  char codeStr[5];
  sprintf(codeStr, "%04d", randomCode);
  return String(codeStr);
}

void sendMqttMessage(String randomCode) {
  StaticJsonDocument<200> doc;
  doc["Action"] = "add device";
  doc["Code"] = randomCode;

  char buffer[256];
  size_t n = serializeJson(doc, buffer);

  if (client.connected()) {
    client.publish("Gataric", buffer, n);
```

```
    Serial.print("Poruka poslana: ");
    Serial.println(buffer);

    display.clearDisplay();
    display.setCursor(0, 0);
    display.println("Kod generiran:");
    display.println(randomCode);
    display.display();
  } else {
    display.println("MQTT klijent nije povezan.");
  }
}

void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Poruka primljena [");
  Serial.print(topic);
  Serial.print("]: ");
  String message = "";
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];
  }
  Serial.println(message);
  StaticJsonDocument<200> doc;
  DeserializationError error = deserializeJson(doc, payload, length);
  if (error) {
    Serial.println("Greška u parsiranju JSON poruke");
    return;
  }
  const char* action = doc["Action"];
  if (strcmp(action, "add device") == 0) {
    display.println("Generiranje nasumičnog koda za dodavanje uređaja...");
    String randomCode = generateRandomCode();
    sendMqttMessage(randomCode);
  } else if (strcmp(action, "door unlock") == 0) {
    display.println("Otključavanje vrata...");
    digitalWrite(relayPin, HIGH);
    delay(30000);
    digitalWrite(relayPin, LOW);
    display.clearDisplay();
```

```
    display.setCursor(0, 0);
    display.println("Vrata otključana!");
    display.display();
  }
else if (strcmp(action, "register fingerprint") == 0) {
    display.println("Pokretanje registracije otiska prsta...");
    enrollFingerprint();
  } }
```

Prethodno navedeni kôd prikazuje funkcionalnost za generiranje nasumičnog kôda, slanje MQTT poruke,te obradu primljenih poruka koje se koriste za dodavanje uređaja ili novog otiska prsta te samo otključavanje vrata. Korištenje MQTT protokola omogućava komunikaciju između različitih uređaja putem objavljivanja i primanja poruka na određene teme. Funkcija *generateRandomCode()* generira nasumični broj između 0 i 9999 pomoću funkcije *random()*. Kako bi se taj broj formatirao kao niz od četiri znamenke, koristi se *sprintf()*, pri čemu se dodaju vodeće nule ako je broj manji od 1000(npr. ako je broj 12, bit će prikazani kao „0012“). Funkcija vraća rezultat u obliku stringa koji predstavlja nasumični četveroznamenkasti broj. Sljedeća vrlo bitna funkcija je *sendMqttMessage()* koja prima nasumično generirani broj i šalje ga putem MQTT-a. Unutar funkcije se kreira JSON dokument koristeći *StaticJsonDocument*, gdje se definiraju polja „Action“ koji sadrži vrijednost „add device“, što označava akciju dodavanja uređaja i „Code“ koji sadrži nasumično generirani broj. Funkcija „serializeJson() pretvara JSON dokument u niz znakova koji se zatim šalje MQTT brokeru ako je veza uspostavljena. Poruka se šalje uz pomoć *client.publish()* na temu „Gataric“ s podacima u JSON formatu. Na ekranu se zatim prikazuje nasumično generirani kod. Osim slanja poruka, kontroler mora i primati poruke, a to je opisano u funkciji *callback()* koja obrađuje MQTT poruke koje dolaze na registrirane teme. Ova se funkcija poziva svaki put kada se primi poruka na neku od pretplaćenih tema. U početku se ispisuje tema na kojoj je poruka primljena, a zatim se konstruira string *message* koji sadrži cijeli payload poruke. Za parsiranje primljenih poruka u JSON format koristi se *deserializeJson()*. Ako dođe do pogreške kod parsiranja, ispisuje se poruka o pogrešci, a funkcija se prekida. Kontroler ima nekoliko radnji koje obavlja, a one su opisane u *action* polju iz JSON dokumenta. Ako je akcija „add device“, generira se novi nasumični kod pomoću funkcije *generateRandomCode()*, a zatim se ta vrijednost šaljem pomoću *sendMqttMessage().* Ako je akcija „door unlock“, vrata se otključavaju aktiviranjem releja pomoću *digitalWrite(relayPin, HIGH),* pri čemu relej ostaje aktivan 30 sekundi (*delay(30000)),* nakon čega se vrata zaključavaju. Ako je akcija „register fingerprint“, pokreće se proces registracije otiska prsta koja je prethodno opisana.

Implementacija sustava pametne brave obuhvaća učinkovitu integraciju hardverskih komponenti poput senzora otiska prsta, releja, LCD ekrana i NodeMCU kontrolera. Kombinacija tih komponenti, uz sigurnu komunikaciju putem MQTT protokola i enkripciju podataka, ostvarena je visoka razina sigurnosti i pouzdanosti u radu pametne brave, čime je postignuta jednostavna i sigurna korisnička interakcija.

4.2. Razvoj Kotlin Android aplikacije

Aplikacija koja je razvijena u sklopu ovog diplomskog rada osmišljena je kako bi omogućila jednostavno i sigurno upravljanje pametnom bravom putem mobilnog uređaj. S obzirom na to da pametne brave postaju sve popularnije zbog svojih naprednih sigurnosnih značajki, nužno je osigurati jednostavno korisničko sučelje koje olakšava svakodnevno korištenje ovih uređaja. Aplikacija pruža korisnicima mogućnost daljinskog upravljanja pametnom bravom, omogućujući im da otključavaju i zaključavaju vrata te dodaju nove uređaje ili biometrijske otiske prstiju na siguran način.

Razvoj aplikacije u programskom jeziku Kotlin, koji je trenutno standard za izradu Android aplikacija, omogućava optimizirano i efikasno programiranje, prilagođeno modernim mobilnim uređajima. Kotlin je odabran zbog svoje jednostavnosti, snage i interoperabilnosti s postojećim Java bibliotekama, ali i zbog vlastitog interesa za razvojem mobilnih aplikacija. Integracija aplikacije s pametnom bravom putem MQTT protokola dodatno povećava njezinu korisnost, omogućujući sigurnu komunikaciju s bravom u realnom vremenu te pružajući korisnicima intuitivnu kontrolu nad sigurnosnim aspektima svog doma ili poslovnog prostora.

Aplikacija se sastoji od nekoliko ključnih dijelova koji omogućuju jednostavno i intuitivno upravljanje sigurnosnim funkcijama. Osnovna struktura aplikacije organizirana je u tri glavna paketa: **auth, device** i **fingerprint**, zajedno s glavnim komponentama kao što su MainActivity i MainFragment. U nastavku će biti opisani samo najbitniji dijelovi za rad aplikacije i komunikacije s kontrolerom.

```
class AuthViewModel(application: Application) :
AndroidViewModel(application) {

    private val _isLoggedIn = MutableLiveData<Boolean>()
    val isLoggedIn: LiveData<Boolean> get() = _isLoggedIn

    private val _isRegistered = MutableLiveData<Boolean>()
    val isRegistered: LiveData<Boolean> get() = _isRegistered
```

```
    private val sharedPreferences =
application.getSharedPreferences("MQTT_PREFS", Context.MODE_PRIVATE)

    fun login(username: String, password: String) {
        val savedUsername = sharedPreferences.getString("username", "")
        val savedPassword = sharedPreferences.getString("password", "")
        _isLoggedIn.value = username == savedUsername && password ==
savedPassword
    }

    fun register(username: String, password: String) {
        sharedPreferences.edit().putString("username", username).apply()
        sharedPreferences.edit().putString("password", password).apply()
        _isRegistered.value = true
    }
}
```

Ovaj kôd implementira AuthViewModel, klasu koja upravlja procesima autentifikacije, odnosno prijave i registracije, unutar Android aplikacije. Koristi se ViewModel i LiveData za upravljanje i praćenje stanja autentifikacije. Osim toga, koristi se i **SharedPreferences** za spremanje korisničkih podataka na uređaju. Klasa *AuthViewModel* nasljeđuje *AndroidViewModel*, što omogućava pristup aplikacijskom kontekstu unutar ViewModel-a, a korisno je kada se trebaju koristiti resursi poput **SharedPreferences**. Za praćenje promjena u statusu prijave koristi se *_isLoggedIn* koji je *MutableLiveData* i označava je li korisnik prijavljen ili ne. Varijabla *isLoggedIn* je *LiveData* koja omogućuje samo čitanje stanja izvana, što osigurava da se status prijave može mijenjati samo unutar ViewModel-a. Slična struktura se koristi i za praćenje status registracije. Privatna varijabla *sharedPreferences* omogućuje pohranu i čitanje jednostavnih podatka, kao što su korisničko ime i lozinka. Ovdje se koristi *MQTT_PREFS* kao naziv datoteke u kojoj se pohranjuju podaci, a mod je *Context.MODE_PRIVATE*, što označava da samo aplikacija može pristupiti tim podacima. Funkcija *login()* dohvaća spremljeno korisničko ime i lozinku iz **SharedPreferences** te ih uspoređuje s vrijednostima koje je korisnik unio. Ako su unijete vrijednosti jednake spremljenima, *_isLoggedIn.Value* se postavlja na *true*, što označava uspješnu prijavu. Sličan postupak je i prilikom registracije. U nastavku je prikazana interakcija korisnika putem unosa korisničkog imena i lozinke te pokretanje procesa autentifikacije.

```
loginButton.setOnClickListener {
    val username = usernameEditText.text.toString()
    val password = passwordEditText.text.toString()
```

```
    if (username.isNotBlank() && password.isNotBlank()) {
        authViewModel.login(username, password)

        authViewModel.isLoggedIn.observe(viewLifecycleOwner) { isLoggedIn ->
            if (isLoggedIn == true) {

findNavController().navigate(R.id.action_loginFragment_to_mainFragment)
            } else {
                Toast.makeText(context, "Login failed",
Toast.LENGTH_SHORT).show()
            }
        }
    } else {
        Toast.makeText(context, "Please enter both username and password",
Toast.LENGTH_SHORT).show()
    }
}
```

Kada korisnik pritisne gumb za prijavu(*loginButton),* postavlja se *OnClickListener*koji započinje proces. Za dohvaćanje teksta koji je korisnik unio u polja za korisničko ime i lozinku koriste se *usernameEditText.text.toString()* i *passwordEditText.text.toString()* koji tekst pretvaraju u stringove. Nakon toga, provjera se je li korisnik unio vrijednosti u oba polja, dok funkcija *isNotBlank()* osigurava da polja nisu prazna ili ispunjena razmacima. Ako je barem jedno od polja prazno, prikazuje se obavijest korisniku. Ako su oba polja ispunjena, poziva se funkcija *login()* unutar *authViewModel* koja provjerava jesu li uneseni podaci točni koristeći pohranjene podatke. Metoda *observe()* omogućuje promatranje promjena vrijednosti *isLoggedIn*, koja je definirana kao LiveData u ViewModel-u. Kada se vrijednost *isLoggedIn* promijeni, poziva se lambda funkcija s novom vrijednošću. Ako je *isLoggedIn* postavljena na istinu, korisnik je uspješno prijavljen te odlazi na početni zaslon, odnosno glavni fragment. Ako je pak prijava neuspješna, odnosno vrijednost *isLoggedIn* je neistinita, prikazuje se obavijest korisniku.

```
private fun setupMqttConnection() {
    val serverUri = "tcp://test.mosquitto.org:8883"
    val clientId = MqttClient.generateClientId()
    mqttClient = MqttClient(serverUri, clientId, MemoryPersistence())

    val mqttOptions = MqttConnectOptions().apply {
        isAutomaticReconnect = true
        isCleanSession = true
```

```
        socketFactory = SSLSocketFactory.getDefault()
    }

    try {
        mqttClient.connect(mqttOptions)
        mqttClient.setCallback(object : MqttCallback {
            override fun connectionLost(cause: Throwable?) {
                Log.e("MQTT", "Veza s brokerom izgubljena")
            }

            override fun messageArrived(topic: String?, message:
MqttMessage?) {
                Log.d("MQTT", "Poruka primljena: ${message.toString()} na
topic: $topic")
            }

            override fun deliveryComplete(token: IMqttDeliveryToken?) {
                Log.d("MQTT", "Poruka isporučena")
            }
        })
        Log.d("MQTT", "Povezan na MQTT broker")

    } catch (e: MqttException) {
        Log.e("MQTT", "Neuspješno povezivanje: ${e.message}")
    }
}
```

Nakon uspješne prijave, potrebno je postaviti vezu s MQTT brokerom koristeći TLS enkripciju. Za povezivanje se koristi **Eclipse Paho** MQTT biblioteka, koja omogućava jednostavnu implementaciju komunikacije putem MQTT protokola. Prvo se definira **URI MQTT** brokera koristeći protokol **tcp** i port **8883**, što označava da se koristi sigurnosni sloj TLS/SSL za enkripciju komunikacije. Generira se jedinstveni ID klijenta pomoću funkcije *MqttClient.generateClient()*, koji služi za identifikaciju uređaja prilikom povezivanja na broker. Instanca klijenta kreira se pomoću klase *MqttClient*, pri čemu se koristi memorijska pohrana za poruke tijekom sesije. Za povezivanje na broker postavljaju se opcije putem klase *MqttConnectOptions*. Unutar tih opcija omogućeno je automatsko ponovno povezivanje klijenta na broker u slučaju prekida veze, te je omogućen način rada s "čistom sesijom", što znači da klijent ne pamti pretplate ili poruke nakon prekida veze. Za uspostavu sigurne komunikacije, koristi se SSL socket putem zadane postavke *SSLSocketFactory*, čime se omogućuje da svi podaci koji se šalju i primaju budu enkriptirani. Nakon definiranja opcija,

klijent pokušava uspostaviti vezu s MQTT brokerom koristeći *connect()* funkciju. Ako je povezivanje uspješno, postavlja se **callback** funkcija koja definira ponašanje klijenta prilikom važnih događaja. Funkcija *connectionLost()* se poziva kada dođe do prekida veze s brokerom i bilježi gubitak veze. Funkcija *messageArrived()* obrađuje dolazne poruke, ispisujući njihovu temu i sadržaj, dok funkcija *deliveryComplete()* potvrđuje da je poruka uspješno dostavljena brokeru. U slučaju uspješnog povezivanja, u logovima se bilježi poruka koja potvrđuje uspostavu veze s brokerom, dok se eventualne greške prilikom povezivanja hvataju pomoću **MqttException** i bilježe u logovima. Ovaj pristup osigurava pouzdanu i sigurnu komunikaciju u IoT sustavima, osiguravajući automatsku obradu poruka i mogućnost automatskog ponovnog povezivanja klijenta u slučaju prekida veze. Osim toga, ovakva implementacija olakšava testiranje jer su svi ključni događaji, poput uspostave ili gubitka veze te primanja poruka, detaljno zabilježeni u logovima.

```
lockButton.setOnClickListener {
    if (isDoorLocked) {
        sendMqttMessage("Gataric", "{\"Device_ID\": 1, \"Door\": 
\"Open\"}")
        lockButton.setBackgroundResource(R.drawable.open_button)
    } else {
        sendMqttMessage("Gataric", "{\"Device_ID\": 1, \"Door\": 
\"Closed\"}")
        lockButton.setBackgroundResource(R.drawable.closed_button)
    }
    isDoorLocked = !isDoorLocked
}
```

Nakon uspješnog povezivanja s MQTT brokerom potrebno je prikazati i njegov rad u aplikaciji, odnosno kako se otključavaju i zaključavaju vrata putem poruka. Gumb *lockButton* postavlja se na *OnClickListener*, što znači da se radnja pokreće kada korisnik pritisne gumb. Funkcionalnost je povezana s promjenom stanja vrata i slanjem odgovarajuće poruke MQTT brokeru kako bi se ažuriralo stanje uređaja. Kada je gumb pritisnut, provjerava se trenutni status vrata koristeći varijablu *isDoorLocked*. Ako su vrata zaključana, pokreće se proces otključavanja tako da funkcija *sendMqttMessage()* šalje MQTT poruku s informacijom o otvaranju vrata na definiranu temu „Gataric“. Poruka se šalje u JSON formatu, pri čemu sadrži polje „Device_ID“ i „Door“, gdje se vrijednost „Door“ postavlja na „Open“, što označava da su vrata otključana. Nakon toga, vizualni prikaz gumba se mijenja pomoću funkcije *setBackgroundResource()*, koja postavlja novu sliku gumba, simbolizirajući otvorena vrata. U suprotnom slučaju, kada su vrata otključana, korisnik pritiskom na gumb pokreće proces zaključavanja vrata. U tom slučaju, također se šalje MQTT poruka, pri čemu se polje "Door"

postavlja na "Closed" kako bi se označilo da su vrata zaključana. Slika gumba se ažurira kako bi prikazivala zaključana vrata.

Na kraju, varijabla *isDoorLocked* mijenja svoje stanje u suprotno kako bi se omogućila izmjena stanja prilikom sljedeće interakcije korisnika. Ovakva implementacija omogućuje jednostavno upravljanje vratima putem korisničkog sučelja, dok se svaka promjena automatski prenosi putem MQTT protokola, osiguravajući da je stanje vrata uvijek sinkronizirano s drugim dijelovima sustava.

Razvoj Android aplikacije omogućuje sigurnu, pouzdanu i intuitivnu kontrolu pametne brave putem mobilnog uređaja. Korištenje Kotlin jezika omogućava jednostavnije i optimizirano programiranje, dok implementacija MVVM arhitekture pomoću LiveData i ViewModel-a osigurava da korisnici mogu upravljati sigurnosnim funkcijama uz minimalan napor i jasan pregled stanja sustava. Uvođenje SharedPreferences za pohranu osjetljivih podataka pruža jednostavno i sigurno rješenje za lokalno spremanje informacija o prijavi. Korištenje MQTT protokola s TLS/SSL enkripcijom osigurava sigurnu komunikaciju između aplikacije i pametne brave, smanjujući rizik neovlaštenog pristupa. Jednostavno sučelje omogućuje korisnicima brzu interakciju, dok se svaka promjena statusa vrata sinkronizira u stvarnom vremenu, čime se povećava sigurnost sustava. Ova integracija čini aplikaciju učinkovitim i sigurnim rješenjem za upravljanje pametnom bravom, uz jednostavno korisničko iskustvo.

5. Zaključak

Sigurnost u digitalnom dobu postaje prioritet, posebno kada se radi o zaštiti fizičkih prostora. Integracija biometrijskih sustava s IoT tehnologijom predstavlja značajan korak prema ostvarivanju naprednih rješenja koja kombiniraju visoku razinu sigurnosti i praktičnost. Ovaj sustav pametne brave temelji se na tehnologiji prepoznavanja otiska prsta, čime se osigurava da samo ovlašteni korisnici mogu pristupiti osjetljivim prostorima. Senzor otiska prsta, AS608, omogućuje visoko preciznu biometrijsku autentifikaciju, dok je upotreba AES-256 enkripcije s nasumičnim soli dodana kako bi se osigurala zaštita pohranjenih biometrijskih podataka. Na ovaj način, biometrijski podaci ostaju zaštićeni čak i u slučaju pokušaja neovlaštenog pristupa. U kombinaciji s MQTT protokolom za komunikaciju između sustava i mobilne aplikacije, osigurana je sigurna i efikasna razmjena podataka. Razvoj mobilne aplikacije u Kotlinu omogućio je jednostavno i intuitivno korisničko iskustvo, s naglaskom na sigurnost i funkcionalnost. Aplikacija omogućuje daljinsko upravljanje sustavom pametne brave, pri čemu je svaki aspekt dizajniran tako da omogući maksimalnu kontrolu i sigurnost korisniku. Korištenjem TLS enkripcije u komunikaciji, osigurana je potpuna zaštita podataka tijekom prijenosa između aplikacije i pametne brave.

Cjelokupni sustav, razvijen kroz integraciju biometrijske autentifikacije, naprednih sigurnosnih protokola i IoT tehnologije, uspješno je demonstrirao kako se suvremeni sigurnosni sustavi mogu učinkovito primijeniti u stvarnim životnim situacijama. Poseban naglasak stavljen je na osiguravanje visoke razine sigurnosti, ali i na jednostavnost uporabe, čime se omogućuje široka primjena ovog rješenja u svakodnevnom okruženju.

6. Popis literature

Android developers. (9 2024). *Android developers*. Dohvaćeno iz Meet Android Studio: https://developer.android.com/studio/intro

Aratek. (1 2023). Dohvaćeno iz The Fingerprint File: 4 Fingerprint Sensor Types : https://www.aratek.co/news/the-4-fingerprint-sensor-types

ArduinoGetStarted. (2024). Dohvaćeno iz Arduino - Solenoid Lock: https://arduinogetstarted.com/tutorials/arduino-solenoid-lock

Cloudfare. (2024). Dohvaćeno iz What is Transport Layer Security (TLS)?: https://www.cloudflare.com/en-gb/learning/ssl/transport-layer-security-tls/

Forrester, A. (2021). *How to build Android apps with Kotlin.* Packt Publishing.

github. (2 2024). Dohvaćeno iz NodeMCU.

Jain, A. F. (2007). *Handbook of Biometrics.* Springer.

Katz J., L. Y. (2020). *Introduction to Modern Cryptography.* CRC Press.

Lea, P. (2018). *Internet of things for architects.* Packt Publishing.

Material Design Guidelines. (2024). Dohvaćeno iz Material design: https://m3.material.io/

MQTT. (2024). Dohvaćeno iz MQTT: The Standard for IoT Messaging: https://mqtt.org/

Nordquist, T. (2024). *MQTT Explorer*. Dohvaćeno iz MQTT client overview: https://mqtt-explorer.com/

Shojaei, A. M. (2023). *Electropeak*. Dohvaćeno iz Interfacing AS608 Optical Fingerprint Sensor Module with Arduino: https://electropeak.com/learn/interfacing-fpm10a-as608-optical-fingerprint-reader-sensor-module-with-arduino/

Söderby K, H. J. (9 2024). *Arduino docs*. Dohvaćeno iz Getting Started with Arduino IDE: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2/

Vacca, J. R. (2017). *Computer and information security handbook.* Cambridge.

Wayman J, J. A. (2005). *Biometric Systems.* Springer.

7. Popis slika