Izrada pametnog uređaja za praćenje kalorija sa senzorima i pripadajuće mobilne aplikacije

Sedlanić, Leon

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Leon Sedlanić

IZRADA PAMETNOG UREĐAJA ZA PRAĆENJE KALORIJA SA SENZORIMA I PRIPADAJUĆE MOBILNE APLIKACIJE

DIPLOMSKI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Leon Sedlanić

Matični broj: 35918/07–R

Studij: Informacijsko i programsko inženjerstvo

IZRADA PAMETNOG UREĐAJA ZA PRAĆENJE KALORIJA SA SENZORIMA I PRIPADAJUĆE MOBILNE APLIKACIJE

DIPLOMSKI RAD

Mentor:

Doc. dr. sc. Bogdan Okreša Đurić

Varaždin, rujan 2024.

Leon Sedlanić

Izjava o izvornosti

Izjavljujem da je ovaj 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.

Autorica potvrdila prihvaćanjem odredbi u sustavu FOI Radovi

Sažetak

Ovaj rad bavi se razvojem pametnog uređaja za praćenje kalorija koji koristi senzore i pripadajuću mobilnu aplikaciju. Glavni cilj rada bio je razviti sustav koji omogućuje korisnicima praćenje unosa kalorija putem integriranih senzora i algoritama strojnog učenja. Uređaj je sastavljen od komponenata kao što su ESP32 CAM, Arduino mikro kontroler i HX711 Load Cell. Sustav koristi MQTT protokol za komunikaciju između uređaja i mobilne aplikacije, omogućujući precizno praćenje i analizu nutritivnih vrijednosti unesenih namirnica. Kroz rad su detaljno opisane tehnologije korištene u izradi sustava, uključujući umjetnu inteligenciju, Internet stvari, strojno učenje i višeagentne sustave u okruženju interneta stvari. Posebna pažnja posvećena je implementaciji algoritama za prepoznavanje hrane te integraciji tih algoritama u mobilnu aplikaciju. Rezultati pokazuju da razvijeni sustav može uspješno pratiti unos kalorija, pružajući korisnicima korisne povratne informacije i preporuke za poboljšanje prehrambenih navika. .

Ključne riječi: IoT, strojno učenje, kalorije, prepoznavanje hrane, MQTT protokol, mobilna aplikacija.

Sadržaj

1. Uvod

U današnjem modernom društvu, gdje tehnologija prodire u sve aspekte života, održavanje zdravog načina života postalo je izazov i potreba. S obzirom na ubrzan tempo života i sveprisutnu dostupnost nezdrave hrane, postaje sve teže pratiti vlastiti unos kalorija i održavati uravnoteženu prehranu. Praćenje prehrambenih navika, unosa kalorija i nutritivnih vrijednosti hrane ključni su faktori u prevenciji raznih bolesti, održavanju zdrave tjelesne težine i postizanju općeg zdravlja. Međutim, većina ljudi nema pristup alatima koji bi im olakšali ovaj zadatak na jednostavan i efikasan način.

Pametni uređaji i Internet stvari predstavljaju revoluciju u ovoj domeni, omogućujući korisnicima jednostavnije praćenje i upravljanje vlastitim zdravljem. Ova tehnologija omogućuje povezivanje različitih uređaja i senzora koji mogu prikupljati i analizirati podatke o našem tijelu i prehrambenim navikama. Korištenjem naprednih senzora i algoritama strojnog učenja, moguće je razviti sustave koji mogu automatski prepoznati hranu, procijeniti njezinu nutritivnu vrijednost i pratiti unos kalorija bez potrebe za ručnim unosom podataka.

Ovaj rad istražuje razvoj pametnog uređaja za praćenje kalorija, koji koristi najnovije tehnološke inovacije kako bi korisnicima pružio jednostavan i učinkovit alat za upravljanje vlastitom prehranom. U radu se detaljno razmatraju različite komponente sustava, uključujući hardverske senzore, softverske algoritme i komunikacijske protokole, te se analiziraju njihovi učinci na preciznost i korisnost uređaja. Osim tehničkih aspekata, rad također istražuje potencijalne koristi ovakvih sustava za unapređenje zdravlja i kvalitete života korisnika.

2. Teorijska podloga

U svijetu gdje se tehnologija i zdravlje sve više isprepliću, nalazimo se na pragu revolucije u osobnom upravljanju zdravljem. Zamislimo svijet u kojem ručni sat nije samo mjerač vremena, već sofisticirani zdravstveni asistent koji prati svaki naš korak, svaki otkucaj srca i svaki zalogaj koji unesemo. Svijet u kojem kuhinja postaje pametni laboratorij, sposoban analizirati nutritivnu vrijednost obroka u stvarnom vremenu. Svijet u kojem pametni telefon postaje osobni nutricionistički savjetnik, pružajući personalizirane preporuke temeljene na jedinstvenim potrebama i ciljevima korisnika.

Internet stvari (eng. Internet of Things - IoT) predstavlja jednog od glavnih subjekata takvog svijeta. IoT čini mrežu fizičkih objekata opremljenih senzorima, softverom i ostalim tehnologijama za prikupljanje i razmjenu podataka s drugim uređajima te sustavima putem interneta. [1] Omogućava uređajima da promatraju, prepoznaju i razumiju okolinu bez oslanjanja na ljudsku pomoć. Primjene IoT uključuju pametne kućanske uređaje, vozila, termostate i monitore za bebe. Uz to, moguće je i proaktivno održavanje opreme, prepoznavanje kvara u vozilima, upravljanje zalihama u maloprodaji, te praćenje medicinskih uređaja i resursa.

IoT transformira svakodnevne predmete u pametne uređaje, stvarajući mrežu povezanih objekata koji komuniciraju, prikupljaju i dijele podatke, dok umjetna inteligencija sa svojim sposobnostima učenja, predviđanja i donošenja odluka dodaje novu dimenziju ovim povezanim uređajima. Zamislimo sustav koji ne samo da prati unos kalorija, već i predviđa prehrambene potrebe na temelju aktivnosti korisnika, razine stresa i čak hormonalnih ciklusa. Isto tako, umjetna inteligencija (eng. Artificial Intelligence - AI) igra sve značajniju ulogu u IoT aplikacijama i organizacijama, omogućavajući stvaranje novih poslovnih prilika i modela. AI se odnosi na sposobnost računala ili robota da obavljaju zadatke koji su obično povezani s ljudskom inteligencijom, kao što su prepoznavanje glasa, razumijevanje jezika, donošenje odluka i druge kognitivne funkcije. [2] Također, omogućava strojevima da uče iz iskustava, prilagođavaju se novim unosima i obavljaju zadatke nalik ljudskim. Primjene uključuju medicinske dijagnoze, personalizirane medicinske tretmane, virtualne kupovne asistente, analizu proizvodnih podataka, otkrivanje prijevara, kreditne ocjene i upravljanje podacima.

Višeagentni sustavi (eng. Multi-agent systems – MAS) pak donose element suradnje i kolektivne inteligencije u ovu jednadžbu. Predočimo si ekosustav pametnih uređaja koji međusobno komuniciraju i koordiniraju svoje aktivnosti kako bi pružili holistički pogled na zdravlje. Pametna narukvica, pametna vaga i pametni hladnjak mogu postati tim agenata koji zajednički rade na optimizaciji zdravlja i prehrane ljudi.

No, ova vizija budućnosti donosi sa sobom i brojne izazove:

- Kako osigurati privatnost i sigurnost osobnih zdravstvenih podataka u svijetu gdje svaki uređaj postaje potencijalni kolektor informacija?
- Kako balansirati tehnološku sofisticiranost s jednostavnošću korištenja?
- Kako osigurati da ovi sustavi budu inkluzivni i dostupni svima, a ne samo tehnološki pismenim pojedincima?

2.1. Umjetna inteligencija stvari

Uloga umjetne inteligencije u revoluciji interneta stvari

Umjetna inteligencija predviđa obavljanje inteligentnih zadataka bez ljudske intervencije, dok IoT povezuje niz uređaja koji razmjenjuju podatke preko mreže. Kombinacija obiju tih tehnologija, umjetna inteligencija stvari (eng. Artificial Intelligence of Things - AIoT), omogućava analizu velikih količina podataka generiranih IoT uređajima, što pomaže u donošenju informiranih odluka. Ova kombinacija ima brojne prednosti: [3]

- Izbjegavanje neplaniranih zastoja: Određivanje u kojem trenutku je potrebno održavanje uređaja može smanjiti ekonomske štete neplaniranih zastoja, tzv. prediktivno održavanje.
- Povećanje operativne učinkovitosti: AI modeli mogu predvidjeti radne uvjete i identificirati potrebne prilagodbe za optimalne rezultate.
- Poboljšanje proizvoda i usluga: Omogućuju interakciju s uređajima, upravljanje flotom i automatizaciju, što poboljšava postojeće proizvode i usluge.
- Poboljšanje upravljanja rizicima: AI i IoT pomažu organizacijama u razumijevanju i predviđanju rizika te automatizaciji brzih odgovora na njih.

Stvarne primjene

Kombinacija AI i IoT već je uspješno primijenjena u mnogim stvarnim situacijama: [3]

- Autonomna vozila: Autonomna vozila predstavljaju revoluciju u transportu, koristeći kombinaciju AI i IoT tehnologija. Ova vozila opremljena su brojnim senzorima koji prikupljaju podatke o okolini u stvarnom vremenu, uključujući druge automobile, pješake, prometne znakove i uvjete na cesti. AI sustavi obrađuju ove podatke kako bi donijeli odluke o upravljanju vozilom, poput ubrzavanja, kočenja, skretanja ili promjene trake.
- Praćenje ugroženih vrsta: U zaštiti bioraznolikosti, kombinacija AI i IoT tehnologija igra ključnu ulogu. IoT uređaji poput GPS ogrlica, kamera s detekcijom pokreta i dronova koriste se za praćenje kretanja i ponašanja ugroženih životinja u njihovim prirodnim staništima. Ovi uređaji prikupljaju ogromne količine podataka o lokacijama životinja, obrascima kretanja i interakcijama s okolišem. AI algoritmi analiziraju ove podatke kako bi identificirali trendove, predvidjeli potencijalne prijetnje i optimizirali strategije očuvanja.
- Pametni termostati: Ovi uređaji povezani su s internetom i opremljeni su senzorima koji prate temperaturu, vlažnost i prisutnost ljudi u prostoru. AI komponenta analizira ove podatke zajedno s vanjskim faktorima poput vremenske prognoze i cijena energije. Sustav uči o preferencijama korisnika i obrascima korištenja prostora tijekom vremena. Na temelju ovih informacija, AI algoritmi optimiziraju postavke

grijanja i hlađenja kako bi postigli idealnu ravnotežu između udobnosti i energetske učinkovitosti.

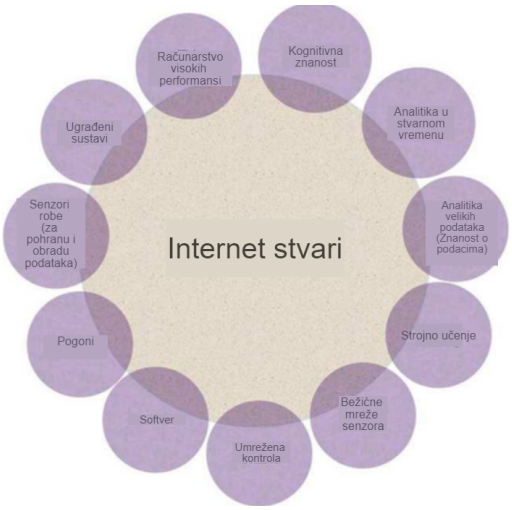
AI omogućuje IoT uređajima da postanu "pametni" kroz procese kao što su strojno učenje (eng. Machine Learning - ML) i analiza podataka (eng. Dana Analysis - DA). ML omogućuje uređajima da uče iz podataka i prilagođavaju svoje ponašanje, dok DA omogućuje analizu velikih količina podataka kako bi se identificirali obrasci i donijele bolje odluke u budućnosti. Pametni objekti (eng. Smart Object - SO) unutar IoT sustava mogu biti bilo što od pametnih telefona do složenih industrijskih strojeva. Ovi uređaji koriste senzore i komunikacijske tehnologije za prikupljanje podataka, dok algoritmi obrađuju te podatke kako bi donijeli autonomne odluke. Na primjer, pametni termostati mogu naučiti preferencije korisnika i automatski prilagoditi temperaturu u kući. No, implementacija AI u IoT-u nosi sa sobom izazove kao što su sigurnosni rizici i etička pitanja. Sigurnost je kritična jer su takvi uređaji često meta cyber napada. Također, etička pitanja vezana uz privatnost podataka i autonomnost odluka koje utječu na ljude zahtijevaju pažljivo razmatranje. [4]

Integracija umjetne inteligencije u internet stvari donosi značajne prednosti za poslovne i društvene procese. Organizacije koje uspiju iskoristiti potencijal ove integracije moći će unaprijediti operativnu učinkovitost, poboljšati proizvode i usluge, te bolje upravljati rizicima. Internet stvari i kibernetičko-fizički sustavi (eng. Cyber-physical system - CPS) predstavljaju nove paradigme gdje su fizičke i digitalne komponente međusobno povezane. Tehnologija umjetne inteligencije AI igra ključnu ulogu u razvoju ovih sustava, omogućujući automatsko donošenje odluka i analizu velikih količina podataka. Primjena AI u IoT-u može značajno unaprijediti performanse i efikasnost sustava, čineći svijet autonomnijim i pametnijim. [5]

Jedno istraživanje [5] bavilo se razvojem fleksibilnog i senzora za praćenje gutanja koje korisnik nosi na sebi, posebno korisnog za pacijente s disfagijom, koji često pate od raka glave i vrata. Ovaj senzor koristi se za detekciju aktivnosti gutanja i može razlikovati signale zdravih osoba od pacijenata s disfagijom s preciznošću od 86,4 posto. Ovi senzori su kombinirani s algoritmom strojnog učenja baziranim na L1-distance metodi kako bi prepoznali proces gutanja bolusa. Rezultati ukazuju na potencijal za neinvazivne i kućne sustave za praćenje funkcije gutanja, omogućujući kontinuirano praćenje pacijenata bez potrebe za stalnim kliničkim posjetima.

Drugo istraživanje [6] se fokusiralo na implementaciju AIoT uređaja za detekciju bolesti kod jagoda koristeći TinyML tehnologiju. U ovom istraživanju korištena je ST B-L4S5I-IOT01A ploča za evaluaciju kombinacije hardvera (ST B-L4S5I-IOT01A ploča) i softvera (TinyML okvir). TinyML omogućava ugrađeno duboko učenje na mikro-kontrolerima, što uređajima omogućava lokalnu obradu podataka i donošenje odluka u realnom vremenu. Istraživanje je pokazalo da AIoT uređaj može detektirati bolesti jagoda efikasno i sa niskom potrošnjom energije. Usprkos brzom napretku AIoT tehnologija, izazovi kao što su sigurnost, kašnjenje, interoperabilnost i pouzdanost senzorskih podataka ostaju ključni za dalji razvoj i primjenu ovih tehnologija u digitalnim blizancima, autonomnim vozilima i industriji 4.0.

Na slici 1 može se vidjeti kružni dijagram koji ilustrira različite komponente i tehnologije vezane uz Internet stvari.



Slika 1: Komponente interneta stvari (Prema: [4])

Integracija AI unutar IoT-a predstavlja revoluciju koja će promijeniti način na koji živimo i radimo. Iako postoje izazovi i rizici, prednosti u smislu automatizacije, efikasnosti i inteligentnog donošenja odluka su ogromne. Kontinuirani napredak u ovim tehnologijama obećava svijet gdje će autonomni sustavi značajno unaprijediti kvalitetu života i produktivnost.

2.2. Strojno učenje

Strojno učenje podskup je umjetne inteligencije usmjeren na razvoj algoritama i statističkih modela koji omogućuju računalima izvršavanje zadataka bez izričitih uputa. Umjesto korištenja tih uputa, ovi modeli uče obrasce i donose odluke na temelju podataka. Cilj ML-a je omogućiti strojevima da uče iz iskustva, poboljšaju svoje performanse i prilagode se novim podacima. Strojno učenje uključuje stvaranje algoritama koji mogu učiti iz podataka i donositi zaključke na temelju podataka. Također, strojno učenje obuhvaća različite tehnike, od jednostavne linearne regresije do složenih neuronskih mreža. Glavna ideja je omogućiti računalima da uče iz podataka, identificiraju obrasce i donose odluke uz minimalnu ljudsku intervenciju. [7]

Opći cilj ML-a je prepoznavanje obrazaca u podacima, koji informiraju način na koji se nevidljivi problemi tretiraju. To znači da obrasci prepoznati u podacima pružaju informacije ili upute o tome kako se nositi s problemima koji nisu izravno vidljivi ili očiti. Na primjer, u vrlo složenim sustavima kao što je samo vozeći automobil, goleme količine podataka koji dolaze iz senzora moraju se pretvoriti u odluke o tome kako upravljati automobilom pomoću računala koje je "naučilo" prepoznati obrazac "opasnosti". Kada govorimo o "obrascu opasnosti", ne mislimo na jedan specifičan obrazac, već na mnoštvo različitih obrazaca koji mogu ukazivati na potencijalno opasnu situaciju. Ovi obrasci mogu uključivati: [8]

- Specifične konfiguracije objekata na cesti
- Određene brzine i smjerove kretanja vozila ili pješaka
- Vremenske uvjete i vidljivost

- Stanje ceste
- Neobično ponašanje drugih sudionika u prometu

Koncept strojnog učenja seže iz 1950-ih kada je Arthur Samuel stvorio program za igranje dame koji se s vremenom poboljšavao učeći iz svojih iskustava. Od tada se to područje značajno razvilo, posebno dolaskom snažnijih računala, velikih skupova podataka i naprednih algoritama. [7]

Tehnike strojnog učenja općenito se kategoriziraju u tri vrste: [7]

- Nadzirano učenje: U nadziranom učenju, model se trenira na označenom skupu podataka, što znači da je svaki primjer obuke uparen s izlaznom oznakom. Algoritam uči mapirati ulaze u izlaze na temelju ovih podataka o obuci. Uobičajeni algoritmi nadziranog učenja uključuju linearnu regresiju, logističku regresiju, vektorske strojeve podrške i neuronske mreže.
- Nenadzirano učenje: Za razliku od nadziranog učenja, algoritmi nenadziranog učenja koriste se kada podaci nemaju označene izlaze. Cilj je identificirati obrasce ili intrinzične strukture u ulaznim podacima. Grupiranje (npr. K-srednje vrijednosti, hijerarhijsko grupiranje) i pridruživanje (npr. Apriori algoritam) tipični su primjeri tehnika učenja bez nadzora.
- Poticano učenje: Ova vrsta učenja uključuje osposobljavanje agenta za donošenje niza odluka nagrađujući ga za dobre postupke i kažnjavajući ga za loše. Učenje s pojačanjem obično se koristi u robotici, igranju igara i drugim domenama.

Strojno učenje ima širok raspon primjena u raznim područjima: [9]

1. Zdravstvo: Predviđanje izbijanja bolesti, personalizirana medicina, analiza medicinske slike.
2. Financije: otkrivanje prijevara, algoritamsko trgovanje, kreditno bodovanje.
3. E-trgovina: sustavi preporuka, segmentacija kupaca, predviđanje potražnje.
4. Prijevoz: Autonomna vozila, optimizacija ruta, predviđanje prometa.
5. Društveni mediji: preporuka sadržaja, analiza raspoloženja, analiza društvenih mreža.

Izvedba ML modela uvelike ovisi o kvaliteti i količini podataka. Nepotpuni, šumoviti ili pristrani podaci mogu dovesti do loše izvedbe modela. Složeni modeli, kao što su duboke neuronske mreže, često djeluju kao "crne kutije", što otežava tumačenje njihovih odluka. [5]

Konvergencija strojnog učenja i IoT-a otvara put budućem napretku u učinkovitosti, točnosti, produktivnosti i ukupnim uštedama za IoT uređaje s ograničenim resursima. Kada algoritmi strojnog učenja i IoT rade zajedno, mogu se postići poboljšane performanse za komunikaciju i računanje, bolju upravljivost i poboljšano donošenje odluka. Ova poboljšanja proizlaze iz sposobnosti strojnog učenja da optimizira korištenje resursa, inteligentno filtrira podatke

i prilagođava se promjenjivim uvjetima. Algoritmi mogu analizirati obrasce u podacima koje generiraju IoT uređaji, omogućujući prediktivno održavanje i učinkovitiju komunikaciju. Implementacija strojnog učenja na samim uređajima (edge computing) dodatno smanjuje potrebu za stalnim prijenosom podataka, poboljšavajući brzinu odziva i energetsku učinkovitost. [10]

Unutar istraživanja [11] predložen je automatizirani sustav praćenja prehrane koji podržava IoT. Autori predlažu algoritam za predviđanje obroka koristeći detaljnu analizu Bayesovih mreža. Kako bi se istražila prehrambena ravnoteža, predlaže se još jedan algoritam koji koristi modele dubokog učenja temeljene na neuronskim mrežama perceptrona. Ovaj računalni model koristi senzorsku infrastrukturu i aplikaciju za praćenje kalorija i komunikaciju s korisnikom.

Strojno učenje revolucioniralo je brojna polja omogućivši automatizaciju na temelju podataka. S pojavom prijenosnog učenja, potencijal za iskorištavanje postojećeg znanja za rješavanje novih problema značajno se proširio. Kako se polje nastavlja razvijati, nedvojbeno će nastati sve sofisticiraniji modeli i tehnike, dodatno ugrađujući strojno učenje u strukturu tehnologije i društva.

2.3. Prijenosno učenje

Prijenosno učenje (eng. Transfer Learning - TL) je tehnika u kojoj se znanje stečeno treniranjem modela na jednom zadatku koristi za poboljšanje performansi modela na drugom, srodnom zadatku. [12] Ova metoda je posebno korisna kada je dostupno malo označenih podataka za novi zadatak, jer omogućuje korištenje pred treniranih modela koji su već naučili neke značajke iz velikih skupova podataka. Također je i tehnika koja je korištena za izradu uređaja unutar ovog rada.

Prijenosno učenje uključuje dvije glavne faze: pred treniranje i fino podešavanje. U fazi pre-treniranja, model se trenira na velikom skupu podataka iz izvorne domene. U fazi finog podešavanja, model se prilagođava specifičnostima ciljnog zadatka pomoću manjeg skupa podataka specifičnog za taj zadatak. Prijenosno učenje se može klasificirati u četiri glavne kategorije: [12]

- Instance-based transfer learning (prijenosno učenje temeljeno na instancama) : Korištenje instanci iz izvorne domene s odgovarajućim težinama.
- Mapping-based transfer learning (prijenosno učenje temeljeno na mapiranju): Mapiranje instanci iz dvije domene u novi prostor podataka s boljom sličnošću.
- Network-based transfer learning (prijenosno učenje temeljeno na mreži): Ponovno korištenje dijela mreže pretrenirane u izvornoj domeni.
- Adversarial-based transfer learning (prijenosno učenje temeljeno na kontradiktornosti): Korištenje kontradiktorne tehnologije za pronalaženje prijenosnih značajki prikladnih za obje domene.

Primjena

TL je široko primjenjiv u raznim područjima, uključujući obradu prirodnog jezika, analizu sentimenta, prepoznavanje slika, prepoznavanje govora i kibernetičku sigurnost. U medicini, TL omogućuje korištenje pred treniranih modela za klasifikaciju bolesti, analizu medicinskih slika i predikciju rizika od bolesti. [12] Ova tehnika značajno smanjuje potrebu za velikim količinama označenih podataka, što je često izazov u medicinskom području. Također omogućuje učinkovito korištenje prethodno stečenog znanja za rješavanje novih zadataka s minimalnim potrebama za treniranjem, čineći ga vrijednim alatom za širok spektar aplikacija u strojnog učenja.

U izvoru [13] opisana je upotreba prijenosnog učenja za prepoznavanje jela koristeći EfficientNet-B0 arhitekturu, nakon usporedbe EfficientNet-B0, VGG-16 i ResNet-50. Glavni cilj tog istraživanja je poboljšanje točnosti prepoznavanja jela iz slike primjenom naprednih tehnika dubokog učenja i transfernog učenja. Model EfficientNet-B0 je postigao najbolje rezultate u usporedbi s VGG-16 i ResNet-50 modelima. Na osnovu postignutih rezultata, razvijena je mobilna aplikacija za prepoznavanje vijetnamskih jela pod nazivom Vietnamese Dishes Recognition (VDR). Aplikacija je izrađena pomoću frameworka Flutter, što omogućava njeno funkcioniranje na više platformi, uključujući sustave Android i iOS.

Inception-V3

Inception-v3 je pred trenirana konvolucijska neuronska mreža koja ima 48 slojeva, te je verzija mreže koja je već uvježbana na više od milijun slika iz ImageNet baze podataka. Ova pred trenirana mreža može klasificirati slike u 1000 kategorija objekata, kao što su tipkovnica, miš, olovka i mnoge životinje. Kao rezultat toga, mreža je naučila prepoznavati širok raspon slika. Mreža ima ulaznu veličinu slike od 299 x 299. Model izdvaja opće značajke iz ulaznih slika u prvom dijelu i klasificira ih na temelju tih značajki u drugom dijelu. Inception-v3 osposobljen je za ImageNet Large Visual Recognition Challenge koristeći podatke iz 2012. Inception v3 široko je korišten model prepoznavanja slike za koji se pokazalo da postiže veću od 78,1 posto točnosti na skupu podataka ImageNet i oko 93,9 posto točnosti u prvih 5 rezultata. [14]

Autori u [12] opisuju korištenje skupa podataka koji sadrži slike 16 različitih klasa hrane, uglavnom južnoindijske kuhinje. Slike su prikupljene putem kamere, mobilnih uređaja, internetskih stranica i drugih izvora. Prije procesa treniranja, slike su prilagođene tako da sve imaju jedinstvenu rezoluciju kako bi se održala konzistentnost, smanjila složenost i povećala efikasnost modela. Rezultati pokazuju da je model napravljen na bazi Inception-V3 postigao točnost klasifikacije od 96,27 posto tijekom faze testiranja. Ova točnost uspoređena je s drugim suvremenim tehnikama i pokazala se superiornom. Metoda konvolucijske neuronske mreže (CNN) uključuje i fazu predobrade, gdje su slike obrezane ili prilagođene da odgovaraju predefiniranoj veličini. Nakon toga, slike su prošle kroz nekoliko slojeva CNN, uključujući slojeve za izvlačenje značajki, potpuno povezane slojeve i softmax funkciju za klasifikaciju.

Ova istraživanja potvrđuju da prijenos učenja, posebno s modelom Inception-V3, može značajno poboljšati točnost klasifikacije slika hrane. Također, pokazalo se da integracija posebno segmentiranih značajki s prilagođenim modelom prije faze klasifikacije dodatno povećava sposobnost prepoznavanja važnih značajki potrebnih za točnu klasifikaciju.

2.4. Višeagentni sustavi u IoT okruženju

Višeagentni sustavi značajan su napredak u području umjetne inteligencije, pružajući kompaktna rješenja za složene, distribuirane i dinamičke probleme. MAS domene primjene obuhvaćaju različita područja uključujući e-trgovinu, logistiku, upravljanje opskrbnim lancem, telekomunikacije, zdravstvenu skrb i proizvodnju. [16] Oni su ključni u razvoju modela i teorija u distribuiranim sustavima velikih razmjera i sustavima usmjerenim na čovjeka, promičući demokratizaciju u korištenju tehnologije. Demokratizacija tehnologije odnosi se na proces kojim se složena tehnološka rješenja čine pristupačnima i razumljivima širem krugu korisnika, neovisno o njihovom tehničkom predznanju. MAS su osmišljeni kako bi pojednostavili kompleksne procese i sučelja, čineći ih pristupačnijima prosječnom korisniku pružanjem dosljednih i logičnih načina interakcije. Ovi sustavi omogućuju laicima, odnosno osobama bez specijaliziranog znanja u području tehnologije ili računalstva, da jednostavno i učinkovito koriste sofisticirane tehnologije.

Inteligentni agent, glavni subjekt MAS sustava, je autonomni entitet sposoban percipirati svoje okruženje i na njega djelovati, razmišljati o svojim ciljevima i poduzimati radnje za postizanje tih ciljeva. [16] To uključuje interakciju s drugim agentima, bilo umjetnim ili ljudskim, i ukorijenjeno je u glavnom cilju umjetne inteligencije stvaranja entiteta koji pokazuju inteligentno ponašanje. MAS su specifični po svojoj decentraliziranoj prirodi i autonomiji svojih komponenti. Svaki agent slijedi vlastite ciljeve. Iako svaki agent ima autonomiju u svojim odlukama i akcijama, oni ne djeluju u izolaciji, već u međusobno povezanom okruženju. Ciljevi agenata često su isprepleteni ili utječu jedni na druge, što zahtijeva usklađivanje akcija kako bi se izbjeglo međusobno ometanje i optimizirali ukupni rezultati. Nedostatak centralizirane kontrole čini MAS posebno prikladnim za aplikacije koje zahtijevaju distribuiranu i paralelnu obradu podataka.

Višeagentni sustavi pokazuju se kao obećavajući pristup za rješavanje izazova u IoT okruženjima. Kao prvo, pružaju načine za bavljenje autonomijom i heterogenošću, što je od ključne važnosti u IoT okruženju. MAS omogućuju heterogenim komponentama da djeluju autonomno, ali i da surađuju kada je to potrebno. Na primjer, senzori na uređaju mogu autonomno prikupljati podatke, dok mobilna aplikacija može samostalno analizirati te podatke i donositi preporuke.

Nadalje, MAS uvode koncept kolektivne inteligencije, što je posebno korisno u kontekstu praćenja kalorija. Umjesto oslanjanja samo na pojedinačne senzore ili algoritme, sustav može kombinirati podatke i uvide iz više izvora - senzora na uređaju, korisničkih unosa u aplikaciju, pa čak i agregiranih podataka od mnoštva korisnika. Ovo omogućuje preciznije praćenje kalorija i personalizirane preporuke temeljene na kolektivnom znanju. Također se u [17] naglašava važnost protokola i normi u interakciji između agenata. To bi uključivalo definiranje standardiziranih protokola za razmjenu podataka između senzora i mobilne aplikacije, kao i uspostavljanje normi za privatnost i sigurnost korisničkih podataka. Ovi protokoli i norme osiguravaju da različiti dijelovi sustava mogu učinkovito komunicirati i surađivati, istovremeno štiteći interese korisnika.

Jedno istraživanje [18] bavi se dinamičkom ko-simulacijom komponenti interneta stvari (IoT) korištenjem višeagentnog sustava. Ova metoda omogućava simulaciju IoT sustava, gdje

svaka komponenta IoT sustava ima svog agenta koji reprezentira simulaciju te komponente u zasebnom alatu. Takav sustav omogućava dinamičko uključivanje i isključivanje komponenti tijekom trajanja simulacije, čime se ostvaruje koncept "Plug-and-Simulate". To znači da se nove komponente mogu dodati u sustav simulacije dok je on u radu, čime se precizno odražava dinamika stvarnog IoT sustava. Evaluacija predloženog koncepta kroz prototip je pokazala da sustav može efikasno simulirati heterogene IoT komponente i omogućiti njihovu interakciju u realnom vremenu, čime se podržava donošenje odluka, optimizacija sustava i prediktivno održavanje.

Još jedno istraživanje opisano u [19] fokusira se na razvoj algoritma višeagentnog sustava za stvaranje učinkovitog sustava za preporuku unutar IoT okruženja. Ovaj sustav koristi decentraliziranu i samo organizirajuću strategiju gdje kibernetički agenti (agenti koji su više fokusirani na znanost o kontrolnim sustavima i teoriji informacija – često uključuju naprednije mehanizme za adaptaciju i učenje) upravljaju i razmjenjuju "opisnike stvari" - bitne vektore koji predstavljaju pametne objekte - na temelju hash funkcija koje očuvaju lokalitet. Hash funkcije koje očuvaju lokalitet su posebna vrsta hash funkcija koje imaju svojstvo da slični ulazni podaci proizvode slične hash vrijednosti. Kibernetički agenti međusobno djeluju u peer-to-peer načinu, organizirajući se kako bi poboljšali performanse sustava. Preliminarni rezultati istraživanja pokazuju da ovaj pristup omogućava otkrivanje relevantnih i korisnih stavki grupiranjem sličnih opisnika, čime se poboljšava točnost i učinkovitost preporuka u IoT aplikacijama kao što su pametni gradovi i zdravstvo. Ovaj pristup demonstrira potencijal višeagentnih sustava da se nose s kompleksnošću i dinamikom IoT okruženja, nudeći skalabilno rješenje za personalizirane preporuke

Svi prethodno navedeni aspekti MAS sustava pružaju snažan okvir za razvoj pametnog uređaja za praćenje kalorija koji je robustan, prilagodljiv i sposoban za pružanje personaliziranih uvida korisnicima u njihove prehrambene navike.

2.5. Internet stvari i pametni uređaji za praćenje zdravlja

Internet stvari (IoT) i pametni uređaji za praćenje zdravlja postali su ključni elementi u razvoju modernih zdravstvenih sustava. Prema [20] procjenjuje se da je do 2020. godine globalno bilo u upotrebi više od 162 milijarde IoT uređaja namijenjenih zdravstvenoj skrbi. Ovaj podatak ukazuje na ogroman potencijal i rastuću važnost pametnih uređaja u kontekstu praćenja zdravlja i općenito zdravstvene skrbi.

Pametni senzori koji se mogu nositi ili ugraditi omogućuju prikupljanje podataka u stvarnom vremenu o navikama korisnika, njihovom kretanju i korištenju uređaja. Ti se podaci zatim obrađuju pomoću tehnika strojnog učenja kako bi se otkrili skriveni obrasci u podacima i pratilo stanje korisnika. Ovo je posebno relevantno za razvoj uređaja za praćenje kalorija, jer bi takav uređaj mogao kontinuirano prikupljati podatke o aktivnostima korisnika, njihovom metabolizmu i unosu hrane.

Arhitektura sustava za pametno praćenje zdravlja obično se sastoji od: [20]

- Razina rubnih čvorova (eng. Edge Nodes) - gdje se prikupljaju podaci sa IoT senzora na tijelu. Ovdje se odvija osnovna obrada podataka na prijenosnim uređajima poput pametnih satova, pametnih telefona ili ugrađenih uređaja.
- Razina maglenih čvorova (eng. Fog Nodes) - gdje se prikupljaju podaci s terenskih senzora ili rubnih uređaja. Ovdje se vrši pohrana i lokalna obrada podataka koristeći servere ili računala.
- Razina obrade u oblaku - gdje se svi podaci prikupljaju i pohranjuju. Ovdje se odvija napredna obrada, uključujući primjenu sofisticiranih algoritama i analizu podataka.

Istraživanje [20] također naglašava važnost primjene strojnog učenja i dubokog učenja u obradi podataka s IoT uređaja za zdravstvenu skrb. Ove tehnike omogućuju otkrivanje složenih obrazaca u podacima i mogu značajno poboljšati točnost praćenja i predviđanja zdravstvenog stanja korisnika.

U istraživanju [21] provedenom na Sveučilištu u Kaliforniji, IoT sustav je korišten za praćenje prehrambenih navika starijih osoba. Senzori postavljeni na kuhinjske uređaje i posuđe prikupljali su podatke o konzumaciji hrane, koji su analizirani kako bi se identificirali obrasci prehrane i potencijalni zdravstveni rizici. Rezultati su pokazali da je ovakav sustav pomogao u ranom otkrivanju prehrambenih nedostataka i poboljšanju zdravstvene skrbi korisnika.

Integracija IoT tehnologije u sustave za praćenje prehrane može značajno poboljšati zdravstvene preporuke. Unutar istraživanja opisanog u [22] predložen je sustav koji koristi IoT uređaje za praćenje unosa hrane i pružanje personaliziranih prehrambenih savjeta. Sustav prikuplja podatke o prehrani putem senzora i koristi algoritme strojnog učenja za analizu tih podataka, što omogućuje prilagođene preporuke temeljem individualnih potreba korisnika.

Važno je napomenuti da razvoj takvih pametnih zdravstvenih sustava donosi i određene izazove, posebno u pogledu sigurnosti i privatnosti podataka. Stoga je pri razvoju uređaja za praćenje kalorija potrebno posebnu pažnju posvetiti zaštiti osjetljivih zdravstvenih podataka korisnika.

2.6. MQTT protokol

Mreža pametnih uređaja unutar interneta stvari olakšava komunikaciju izravno (uređaj-uređaj) ili putem posredničkih platformi koristeći različite protokole.

Među tim protokolima, protokol „Telemetrijski prijenos poruka kroz red čekanja“ (eng. Message Queuing Telemetry Transport – MQTT) ističe se zbog svoje male težine i učinkovitosti u okruženjima s ograničenom propusnošću i procesorskom snagom. MQTT je posebno dizajniran za rješavanje potreba IoT uređaja i sustava, nudeći pouzdanu isporuku poruka u izazovnim mrežnim uvjetima. [23]

Osnovni koncepti MQTT [20]

Model objavljivanja/pretplate: MQTT radi na modelu slanja poruka preko objavljivanja ili pretplata. Objavljivači šalju poruke o određenim temama, dok pretplatnici primaju poruke o

temama koje ih zanimaju. Ovo odvaja proizvodnju informacija od njihove potrošnje, dopuštajući skalabilne i fleksibilne komunikacijske obrasce.

Teme i pretplate: Teme u MQTT-u djeluju kao subjekti ili kanali za poruke. Klijenti se mogu pretplatiti na određene teme kako bi primali relevantne informacije. MQTT podržava hijerarhijske teme i pretplate sa zamjenskim znakovima, omogućujući učinkovito širenje podataka kroz povezane teme.

Razine kvalitete usluge (QoS); MQTT podržava tri razine osiguranja isporuke poruka:

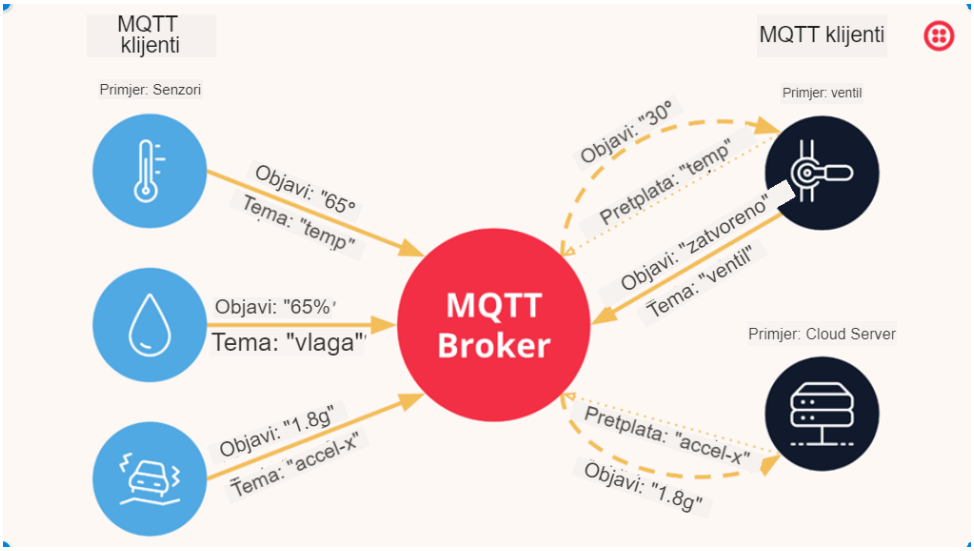
- QoS 0 (Najviše jednom): Poruke se isporučuju najviše jednom, bez potrebe za potvrdom. Ova je razina prikladna za aplikacije u kojima je prihvatljiv povremen gubitak poruka.
- QoS 1 (barem jednom): Zajamčeno je da će poruke biti isporučene barem jednom, s mogućim duplikatima. Ova razina pruža ravnotežu između pouzdanosti i performansi.
- QoS 2 (Točno jednom): Zajamčeno je da će poruke biti isporučene točno jednom, korištenjem procesa rukovanja u četiri koraka. Ova se razina koristi u aplikacijama u kojima se mora izbjeći dupliciranje poruka.

Zadržane poruke: MQTT brokeri mogu zadržati zadnju poruku poslanu na temu. Kada se novi klijent pretplati na tu temu, zadržana poruka se šalje odmah, osiguravajući da novi pretplatnici dobiju najnovije informacije.

Čiste sesije i pouzdane veze: Klijenti mogu odrediti treba li njihova sesija biti trajna (oznaka čiste sesije postavljena na false) ili prolazna (oznaka čiste sesije postavljena na true). Trajne sesije zadržavaju pretplate i neisporučene poruke, olakšavajući pouzdanu komunikaciju za povremeno povezane uređaje.

Oporuke: Klijenti mogu definirati oporuku koju broker šalje ako klijent neočekivano prekine vezu. Ova je značajka osobito korisna u scenarijima koji zahtijevaju trenutnu obavijest o kvarovima ili prekidima veze s uređajem.

Na slici 2 prikazana je arhitektura MQTT sustava gdje senzori kao klijenti objavljuju podatke na određene teme, primjerice senzor temperature na temu "temp" s vrijednošću "65°", senzor vlage na temu "vlaga" s vrijednošću "65 posto", te akcelerometar na temu "accel-x" s vrijednošću "1.8g". MQTT broker, koji je središnji dio sustava, prima te podatke i prosljeđuje ih drugim klijentima koji su pretplaćeni na određene teme. Na primjer, ventil je pretplaćen na teme "temp" i "ventil", pa prima podatke s teme "temp" i objavljuje povratnu informaciju "zatvoreno" na temu "ventil", dok cloud server prima podatke s teme "accel-x".



Slika 2: Primjer MQTT brokera (Prema: [23])

MQTT arhitektura

MQTT arhitektura sastoji se od dvije glavne komponente: klijenata i brokera. [20]

Klijent: Klijent može biti ili izdavač ili pretplatnik. Odgovoran je za uspostavljanje veze s brokerom, objavljivanje poruka u temama, pretplatu na teme i odjavu s tema.

Broker: Broker djeluje kao posrednik, upravljajući distribucijom poruka između klijenata. Prima poruke od izdavača, filtrira ih na temelju pretplata i prosljeđuje odgovarajućim pretplatnicima. Broker je također odgovoran za upravljanje klijentskim sesijama, zadržanim porukama i osiguravanje sigurnosti komunikacijskog procesa.

MQTT brokeri

MQTT broker središnji je dio rada protokola, osiguravajući učinkovito usmjeravanje i isporuku poruka. Kratkim pretraživanjem na internetskom pregledniku može se pronaći nekoliko MQTT brokera, svaki s jedinstvenim značajkama i ograničenjima:

Mosquitto: Broker otvorenog koda koji podržava sve razine MQTT QoS i nudi značajke kao što su autentifikacija, premošćivanje i WebSockets. Pogodan je za različite aplikacije, ali ne podržava istodobne veze s autentifikacijom.

RSMB (Really Small Message Broker): Kompaktni broker koji podržava QoS razine i dinamičke teme. Nedostaju mu značajke kao što su klasteriranje i WebSockets.

MQTT.js: Broker napisan u JavaScriptu, koji pruža API klijent/poslužitelj i podržava značajke kao što su WebSockets i SSL. Idealan je za web aplikacije, ali ima ograničenja.

Primjene MQTT-a

Slika 3: Važnost MQTT u IOT (Prema: [24])

Na slici 3 prikazane su važnosti MQTT protokola u kontekstu IoT tehnologije. Navedeni aspekti ističu MQTT kao ključan alat u optimizaciji i pouzdanom radu IoT sustava, doprinoseći boljoj povezivosti i komunikaciji između uređaja. Primjene MQTT možemo pronaći: [25]

Zdravstvo: Omogućuje daljinsko praćenje pacijenata dopuštajući medicinskim uređajima prijenos dijagnostičkih podataka pružateljima zdravstvenih usluga u stvarnom vremenu. Time se smanjuje potreba za čestim odlascima u bolnicu i omogućuju pravovremene medicinske intervencije.

Energija i komunalije: Koristi se u aplikacijama pametne mreže za praćenje i upravljanje potrošnjom energije. Pametna brojila opremljena MQTT klijentima mogu prijaviti podatke o korištenju u središnji sustav, olakšavajući učinkovitu distribuciju energije i upravljanje potrošnjom.

Društveno umrežavanje: Može poboljšati performanse platformi društvenih mreža smanjenjem latencije u isporuci poruka. Podržava značajke komunikacije u stvarnom vremenu kao što su izravne poruke i obavijesti, poboljšavajući korisničko iskustvo. [25]

3. Višeagentni sustav za praćenje kalorija

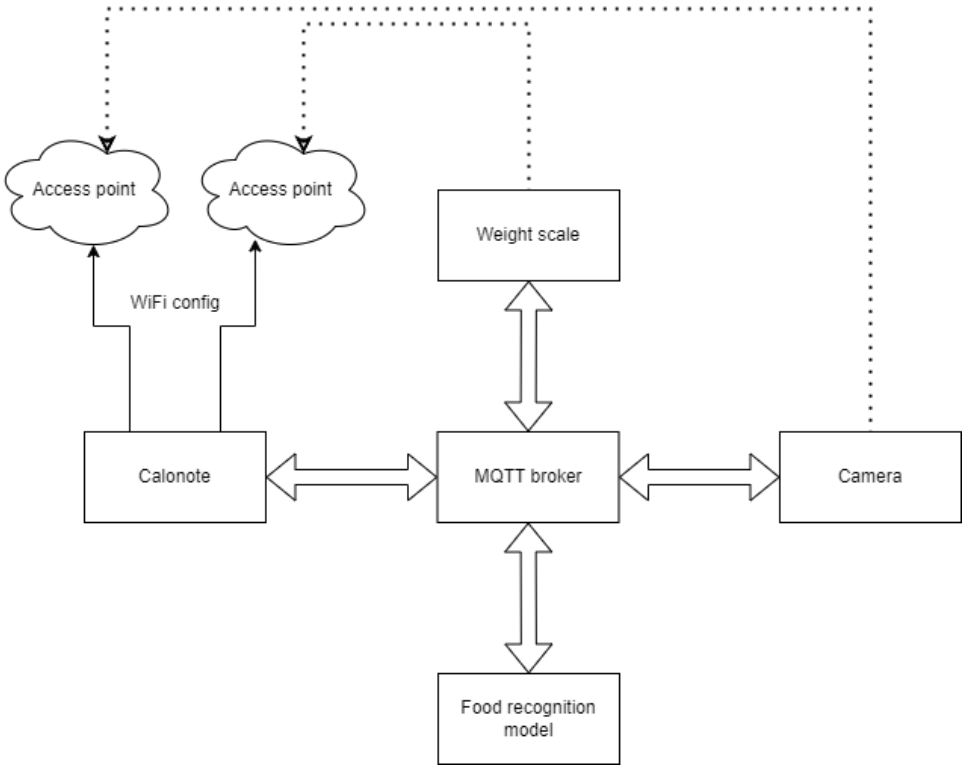
Pametni uređaj za praćenje kalorija kombinira napredne tehnologije poput senzora, umjetne inteligencije i Interneta stvari kako bi korisnicima omogućio praćenje unosa kalorija i nutritivnih vrijednosti hrane na jednostavan i učinkovit način. Sustav koristi višeagentni pristup gdje različite komponente djeluju zajedno kako bi pružile precizne i pravovremene informacije korisniku.

ESP32 CAM i ESP32 mikro kontroleri odgovorni su za prikupljanje podataka sa senzora, poput težine hrane i slika hrane putem kamere. Slika se zatim prosljeđuje Python aplikaciji, koja koristi trenirani model za prepoznavanje vrste hrane. Nakon što se hrana prepozna, podaci se prenose na Android aplikaciju, koja obrađuje informacije i procjenjuje nutritivne vrijednosti hrane, uključujući i broj kalorija.

Komunikacija između uređaja odvija se putem MQTT protokola, gdje MQTT broker omogućuje nesmetanu razmjenu podataka između različitih komponenti sustava. Ovaj višeagentni sustav omogućuje svakom dijelu da specijalizirano obavlja svoju funkciju, a sve zajedno omogućuje korisniku detaljno i precizno praćenje prehrane u realnom vremenu, uz pružanje korisnih povratnih informacija i preporuka za poboljšanje prehrambenih navika. Višeagentnost omogućuje fleksibilnost i efikasnost, jer svaki agent može neovisno obavljati svoje zadatke dok međusobno surađuju u postizanju zajedničkog cilja – unapređenju zdravlja korisnika.

3.1. Arhitektura

Arhitektura ovog sustava je prilično jednostavna. Središnja komponenta koja povezuje sve ostale jest MQTT broker koji upravlja kanalima komunikacije između agenata. Android aplikacija pomaže korisniku da prebaci konfiguracijske podatke za spajanje na WiFi vagi i kameri. Sve komponente su povezane na Internet nakon konfiguracije i bez internetske veze sustav ne radi.



Slika 4: Arhitektura sustava

3.2. Pregled komponenti sustava

Sljedeće komponente su potrebne za spajanje sustava:

- ESP32 CAM
- ARDUINO mikro kontroler sa WiFi
- HX711 Load Cell
- Programer za ESP32 CAM
- Konektorski kabeli
- Kutija (proizvoljne dimenzije), drvena daska (proizvoljne dimenzije, najbolje malo šire od kutije i debljine oko 1-2 cm), dva plastična čepa jednake veličine
- 2 USB-C (M) USB-A (M) kabela
- Pametni telefon
- Breadboard (opcionalno, moguće je i spajanje lemilicom ili drugi načini)

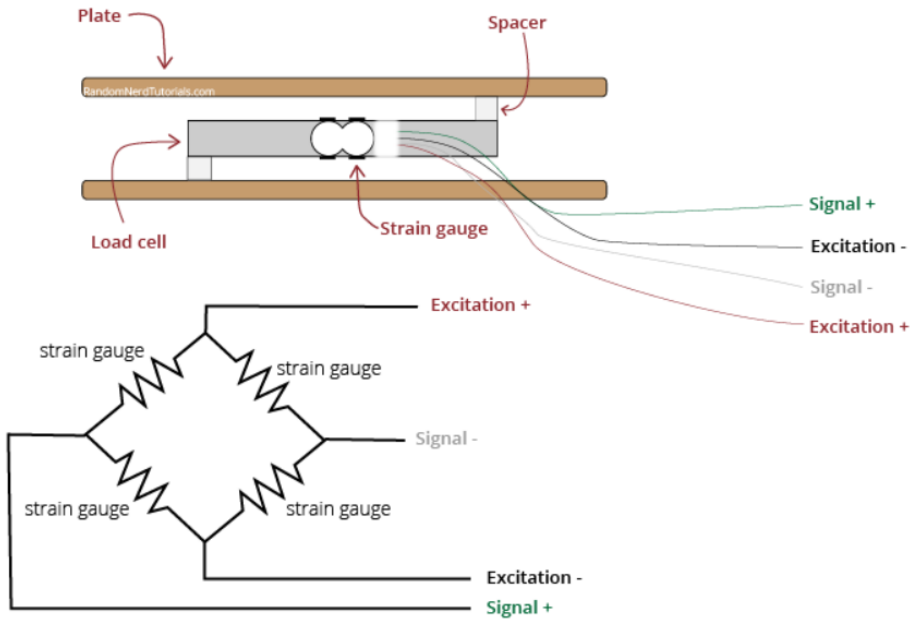
Tehnologije:

- Bilo koji MQTT broker (ovdje je korišten EMQX online broker `https://www.emqx.com/en`)

- Python
  - tensorflow
  - numpy
  - PIL
  - json
  - paho.mqtt.client
- Android studio
- Arduino IDE

3.2.1. Vaga

Prva komponenta sustava (a i najveća) je vaga sastavljena od kutije, daske, dva plastična čepa, nekoliko šarafa, Arduino mikro kontrolera i Load Cell-a. Load cell ima sljedeću osnovnu arhitekturu:



Slika 5: Load Cell (Prema: [26])

Mjerna ćelija (Load Cell) je elektromehanički senzor koji se koristi za mjerenje sile ili težine. Ima jednostavan, ali učinkovit dizajn koji se oslanja na dobro poznati prijenos između primijenjene sile, deformacije materijala i protoka električne energije. Mjerne ćelije su nevjerojatno svestrani uređaji koji nude preciznu i robusnu izvedbu u raznim primjenama. Postali su ključni za mnoge industrijske i komercijalne procese, od automatizacije proizvodnje automobila do vaganja kupnje na blagajni. Kako tehnologija napreduje, pojavljuju se mnoge nove i uzbudljive primjene koje također mogu imati koristi od korištenja mjernih ćelija. Neki od primjera su nova postignuća u robotici i medicinskim protezama koji trebaju učinkovite načine mjerenja sile

i težine. Nove vrste mjernih ćelija kontinuirano se dizajniraju kako bi zadovoljile potrebe ovog tržišta koje se stalno mijenja. [27]

Kada koristimo mjerne ćelije, jedan kraj je obično pričvršćen za okvir ili bazu, dok je drugi kraj slobodan za pričvršćivanje utega ili nosivog elementa.

Kada se na tijelo mjerne ćelije primijeni sila, ono se lagano savija pod naprezanjem. To je slično onome što se događa sa štapom za pecanje kada ribar upeca ribu. Ribar će držati štap u svojim rukama dok riba primjenjuje vučnu silu na drugom kraju strune. Rezultat ove sile je da se štap za pecanje savija više, kod veće, jače ribe, zbog čega je savijanje ekstremnije. Kada se to dogodi senzoru opterećenja, deformacija je vrlo suptilna i nije vidljiva golim okom. Za mjerenje deformacije, mjerači naprezanja čvrsto su spojeni na tijelo mjerne ćelije na unaprijed određenim točkama, uzrokujući njihovu deformaciju u skladu s tijelom. Rezultirajuće kretanje mijenja električni otpor mjerača naprezanja proporcionalno količini deformacije uzrokovane primijenjenim opterećenjem. [27]

Korištenjem elektronike za kondicioniranje signala, električni otpor mjerača naprezanja može se izmjeriti s rezultirajućim signalom koji se šalje kao očitanje težine ili sile.

Mjerne ćelije imaju sljedeće žice:

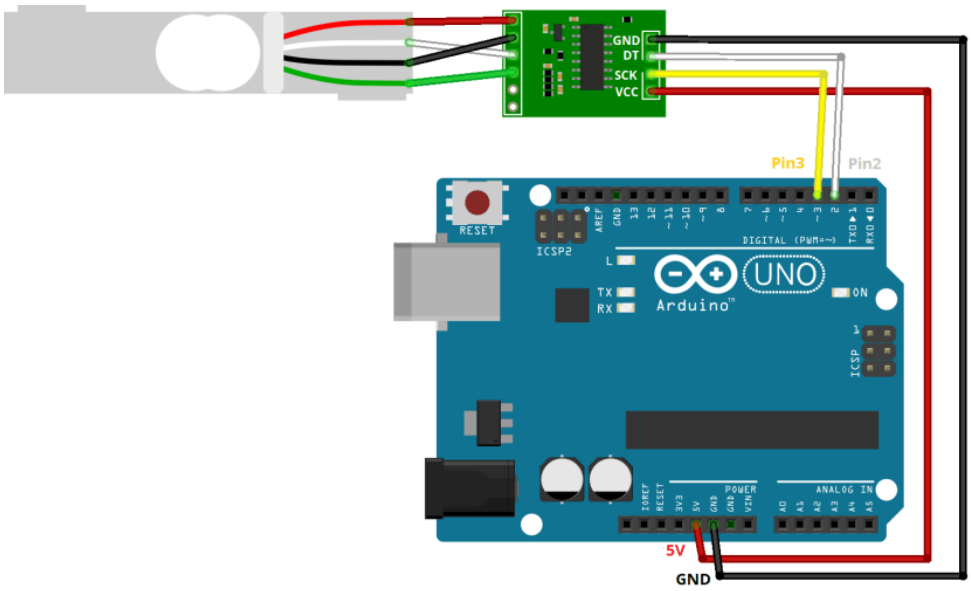
- Crvena: VCC (E+)
- Crna: GND (E-)
- Bijela: Output – (A-)
- Zelena: Output + (A+)

Te žice se spajaju općenito prema ovoj shemi:

Load Cell	HX711	HX711	Arduino
Crvena (E+)	E+	GND	GND
Crna (E-)	E-	DT	Pin 2
Bijela (A-)	A-	SCK	Pin 3
Zelena (A+)	A+	VCC	5V

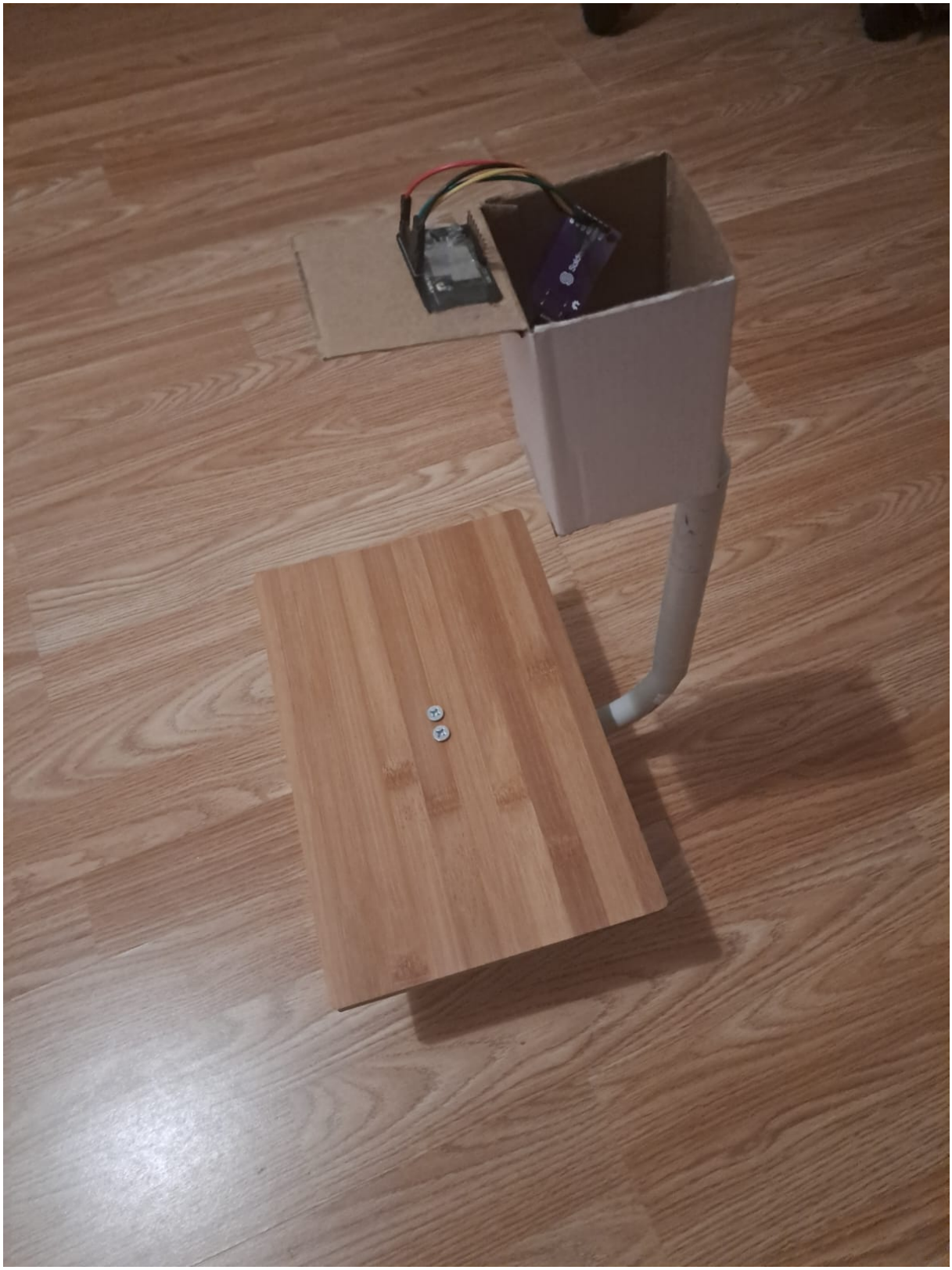
Tablica 1: Spajanje HX711, Load Cell i Arduino (Prema: [26]))

U ovom slučaju nije bilo 5V na ESP32 WROVER modulu kojeg sam koristio, no dovoljan je i 3V3 pin. Bitno je također pogledati shemu mikro kontrolera zbog pinova 2 i 3 koji su spojeni na Data Pin (DT) i Clock Pin (CLK). Ti pinovi su bitni jer HX711 komunicira dvostranim sučeljem i moraju biti spojeni na GPIO 16 i GPIO 4. Na ESP32 WROVER, to su pinovi 4 i 14 prema službenoj shemi.

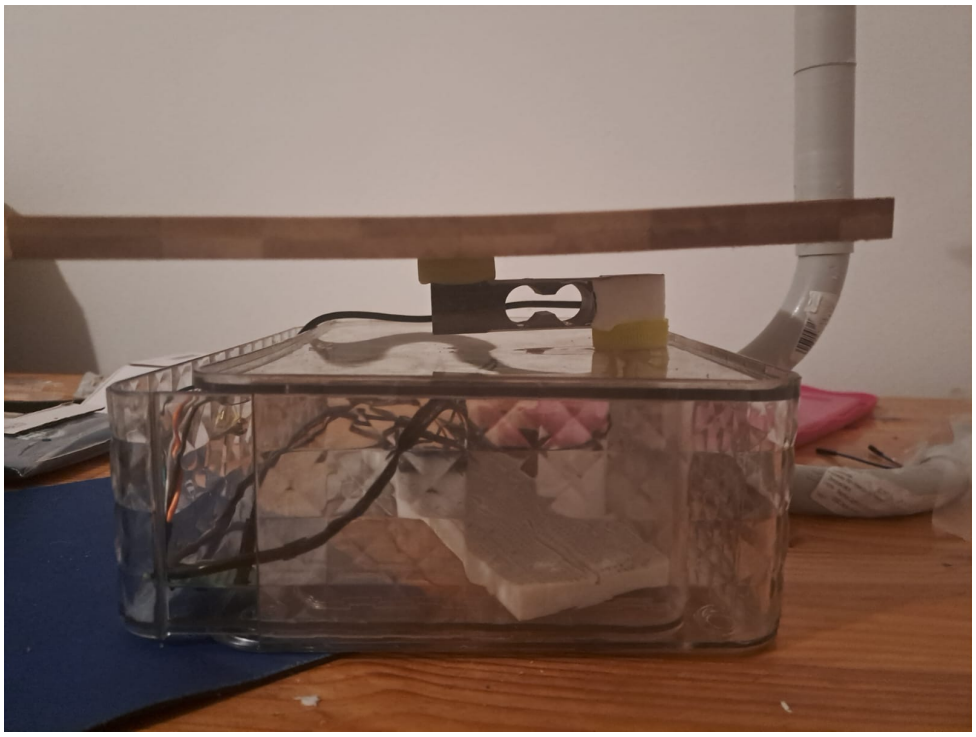


Slika 6: Shema spajanja Load Cell i HX711 uz mikro kontroler (Prema: [26]))

Što se tiče dizajna vage, bilo je potrebno sastaviti snažnu podlogu te nosivi element koji će stajati labilno iznad mjerne ćelije. Za podlogu odabrana je plastična kutija sa otvorom u kojeg mogu staviti sve komponente sustava te kasnije napraviti otvor za napajanje. Nosivi element je obična drvena daska za rezanje nešto šira od same kutije. Između baze i nosivog elementa moraju biti odstupnici, a za taj zadatak bila su dovoljna dva obična plastična čepa istog promjera. Idealniji bi bili odstojni vijci, no to nije bilo pri ruci. Na slici 7 može se vidjeti kompletan uređaj zajedno sa kamerom koja je provučena kroz plastičnu cijev koja proizlazi iz baze vage, dok na slici 7 je vidljiv i bočni pregled same vage gdje se vide čepovi koji služe kao odstojni vijci.



Slika 7: Dizajn uređaja



Slika 8: Bočni pregled uređaja

3.2.1.1. Softver vage

Programski kod je dizajniran s fokusom na fleksibilnost i lakoću uporabe. Omogućava konfiguraciju Wi-Fi vjerodajnica na dva načina: kroz web sučelje kada ESP32 radi u načinu pristupne točke kroz startAP(), ili daljinski putem MQTT poruka. Ova dvostruka metoda konfiguracije osigurava da se uređaj može lako podesiti i u situacijama kada izravan fizički pristup nije moguć.

Komunikacija s vanjskim svijetom odvija se prvenstveno putem MQTT protokola, koristeći sigurnu TLS/SSL vezu. Uređaj se pretplaćuje na nekoliko MQTT tema koje mu omogućavaju primanje naredbi za kalibraciju i zahtjeve za mjerenje težine (handleWeightRequest). Rezultati mjerenja i razni statusi objavljuju se natrag na odgovarajuće MQTT teme, omogućavajući daljinski nadzor i kontrolu sustava.

Kalibracija vage unutar handleCalibration() implementirana je na način da dopušta korisniku da inicira proces kalibracije, postavi poznatu težinu, i potvrdi kalibraciju - sve putem MQTT naredbi. Ovo omogućava precizno podešavanje sustava bez potrebe za fizičkim pristupom uređaju.

```
#include <WiFi.h>
#include <PubSubClient.h>
#include "HX711.h"
#include <Preferences.h>
#include <ESPmDNS.h>
#include <WiFiClientSecure.h>
#include <WebServer.h>

```

```
WebServer server(80);


Preferences preferences;
String ssid;
String password;

const int LOADCELL_DOUT_PIN = 14;
const int LOADCELL_SCK_PIN = 4;

HX711 scale;
const char* root_ca PROGMEM = R"EOF(
-----BEGIN CERTIFICATE-----
<YOUR CERTIFICATE HERE>)EOF";

bool wifiConfigured = false;
const char* mqtt_server = "<MQTT_URL>";
const int mqtt_port = 8883;
const char* mqtt_username = "<YOUR_MQTT_USERNAME>";
const char* mqtt_password = "<YOUR_MQTT_PASSWORD>";

WiFiClientSecure espClient;

PubSubClient client(espClient);

void startAP() {
    WiFi.mode(WIFI_AP);
    WiFi.softAP("ESP32_AP", "12345678");

    server.on("/", HTTP_GET, handleRoot);
    server.on("/configure", HTTP_POST, handleConfigure);
    server.begin();

    Serial.print("Access Point IP address: ");
    Serial.println(WiFi.softAPIP());
}

void setup() {
    Serial.begin(115200);
    espClient.setCACert(root_ca);
    String savedSSID;
    String savedPassword;

    delay(2000);
    Serial.println("Starting up...");

    if (loadWiFiCredentials(savedSSID, savedPassword)) {
        ssid = savedSSID;
        password = savedPassword;
        wifiConfigured = true;
        connectToWiFi();
    }

```

```
    if (!wifiConfigured) {
        startAP();
    }


    scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN);
    scale.set_scale(590);
    scale.tare();

    client.setServer(mqtt_server, mqtt_port);
    client.setCallback(callback);
}

void loop() {
    if (!client.connected() && WiFi.status() == WL_CONNECTED) {
        reconnect();
    }
    client.loop();
    server.handleClient();
}

void connectToWiFi() {
    if (wifiConfigured) {
        Serial.print("Connecting to ");
        Serial.println(ssid);
        WiFi.mode(WIFI_STA);
        WiFi.setSleep(false);
        WiFi.begin(ssid.c_str(), password.c_str());

        unsigned long startTime = millis();
        while (WiFi.status() != WL_CONNECTED && millis() - startTime < 10000) {  // Timeout set to 10 seconds
            delay(500);
            Serial.print(".");
        }

        if (WiFi.status() == WL_CONNECTED) {
            Serial.println("");
            Serial.println("WiFi connected.");
        } else {
            Serial.println("");
            Serial.println("WiFi connection failed. Starting AP mode.");
            startAP();  // Start access point mode
        }
    } else {
        Serial.println("WiFi not configured");
    }
}

void saveWiFiCredentials(const String& ssid, const String& password) {
  preferences.begin("wifi-config", false);
  preferences.putString("ssid", ssid);
  preferences.putString("password", password);
```

```
  preferences.end();
}

bool loadWiFiCredentials(String& ssid, String& password) {
  preferences.begin("wifi-config", true);
  bool hasCredentials = preferences.isKey("ssid") && preferences.isKey("password");

  if (hasCredentials) {
    ssid = preferences.getString("ssid");
    password = preferences.getString("password");
  }

  preferences.end();
  return hasCredentials;
}

void reconnect() {
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        if (client.connect("ESP32Client", mqtt_username, mqtt_password,
          NULL, 0, true, NULL, true)) {
            Serial.println("connected");
            client.subscribe("esp32/calibrate");
            client.subscribe("esp32/weight_request");
            client.subscribe("esp32/wifi_config");
            client.subscribe("android/ping");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            delay(5000);
        }
    }
}

void handleWiFiConfig(String message) {
    int commaIndex = message.indexOf(',');
    if (commaIndex != -1) {
        String newSsid = message.substring(0, commaIndex);
        String newPassword = message.substring(commaIndex + 1);

        saveWiFiCredentials(newSsid, newPassword);

        client.publish("esp32/wifi_config_result", "WiFi credentials updated.
         Restarting...");
        delay(1000);
        ESP.restart();
    } else {
        client.publish("esp32/wifi_config_result", "Invalid WiFi configuration
         format");
    }
}

```

```
165
166 void handleCalibration(String message) {
167     if (message == "calibrate") {
168         Serial.println("Calibration requested");
169         client.publish("esp32/calibration_status", "Place the calibration weight on
          the scale and send 'confirm' to esp32/calibrate");
170     } else if (message == "confirm") {
171         tare();
172         client.publish("esp32/calibration_result", "Calibration completed");
173     }
174 }
175
176 void tare() {
177     Serial.println("Taring scale...");
178     scale.tare();  // Reset the scale to zero
179     Serial.println("Scale tared");
180 }
181 void handlePing() {
182     client.publish("esp32/ping", "pong");
183 }
184
185 void handleWeightRequest() {
186     Serial.println("Weight request received");
187     float weight = scale.get_units(10);
188     String weightStr = String(weight, 2);  // 2 decimal places
189     Serial.println("Weight: " + weightStr);
190
191     uint8_t* payload = (uint8_t*)weightStr.c_str();
192     unsigned int length = weightStr.length();
193
194     if (client.publish("esp32/weight", payload, length, false)) {
195         Serial.println("Weight published successfully");
196     } else {
197         Serial.println("Failed to publish weight");
198     }
199 }
200
201 void callback(char* topic, byte* payload, unsigned int length) {
202     String message = "";
203     for (int i = 0; i < length; i++) {
204         message += (char)payload[i];
205     }
206
207     if (String(topic) == "esp32/calibrate") {
208         handleCalibration(message);
209     } else if (String(topic) == "esp32/weight_request") {
210         handleWeightRequest();
211     } else if (String(topic) == "android/ping") {
212         handlePing();
213     } else if (String(topic) == "esp32/wifi_config") {
214     handleWiFiConfig(message);
215     }
216
```

```
217
218 }
219
220
221 void handleRoot() {
222     String html = "<html><body>";
223     html += "<h1>ESP32 Configuration</h1>";
224     html += "<form method='post' action='/configure'>";
225     html += "SSID: <input type='text' name='ssid'><br>";
226     html += "Password: <input type='password' name='password'><br>";
227     html += "<input type='submit' value='Configure'>";
228     html += "</form></body></html>";
229     server.send(200, "text/html", html);
230 }
231
232 void handleConfigure() {
233     String newSsid = server.arg("ssid");
234     String newPassword = server.arg("password");
235     saveWiFiCredentials(newSsid, newPassword);
236     server.send(200, "text/plain", "Configuration saved. ESP32 will restart.");
237     delay(1000);
238     ESP.restart();
239 }
```

3.2.2. Kamera za prepoznavanje hrane

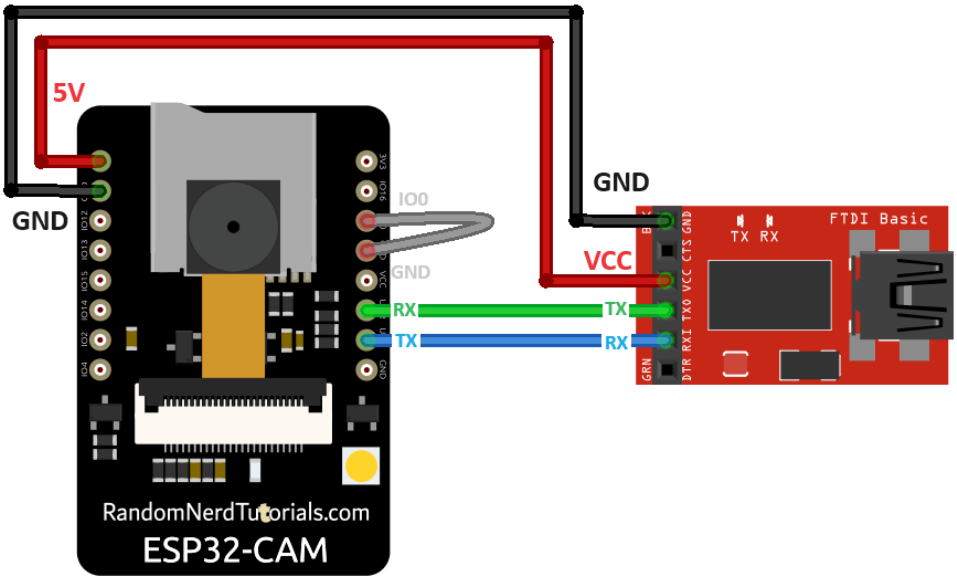
Još jedna komponenta koja je sastavni dio ovog uređaja jest ESP32-CAM kamera koja služi za fotografiranje hrane kako bi se mogla poslati dalje na obradu i prepoznavanje. Kamera je učvršćena za bazu vage i usmjerena prema mjestu gdje je potrebno staviti hranu.

Shema spajanja je sljedeća:

ESP32-CAM	FTDI Programer
GND	GND
5V	5V
RX	TX
TX	RX
GPIO 0 GND	

Tablica 2: Spajanje ESP32-CAM i programera (Prema: [28])

Za postavljanje programskog koda na kameru potreban je FTDI programer jer kamera nema USB->TTL utor. Opet kao i kod vage, nije bilo moguće spojiti na 5V u mojem slučaju i uredu je spojiti sa 3V3. Pri učitavanju koda, potrebno je imati GPIO0 i GND spojene na kameri. Nakon učitavanja potrebno je od spojiti ta dva pina jer inače ostaje u flash modu. Shema spajanja može se vidjeti na slici 9.



Slika 9: Shema spajanja ESP32-CAM i programera (Prema: [28])

3.2.2.1. Softver kamere

Programski kod je veoma sličan onome za vagu kada gledamo konekcijske sposobnosti, no ostale funkcionalnosti su različite.

Kamera započinje s inicijalizacijom detaljnih postavki specifičnih za AI Thinker ESP32-CAM modul.

Značajna funkcionalnost je sposobnost prilagodbe postavki kamere ovisno o uvjetima osvjetljenja. Funkcija checkLightingConditions() analizira prosječnu svjetlinu slike, a adjustCameraSettings() prilagođava postavke kamere prema detektiranim uvjetima osvjetljenja, optimizirajući kvalitetu slike u različitim okruženjima.

Kamera se pretplaćuje na nekoliko MQTT tema za primanje naredbi, uključujući zahtjeve za snimanje slike, Wi-Fi konfiguraciju i provjeru statusa povezanosti. Kada primi naredbu za snimanje, sustav snima sliku, prilagođava postavke prema uvjetima osvjetljenja i šalje sliku putem MQTT-a.

Dodatne funkcije, kao i kod vage, uključuju mogućnost daljinskog ažuriranja Wi-Fi postavki i provjeru statusa povezanosti uređaja.

```
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
#include "esp_camera.h"
#include <Preferences.h>
#include <ESPmDNS.h>
#include <WebServer.h>
#include "base64.h"


WebServer server(80);
```

```
Preferences preferences;
String ssid;
String password;

const char* mqtt_server = "<YOUR_MQTT_SERVER>";
const int mqtt_port = 8883;
const char* mqtt_username = "<MQTT_USERNAME>";
const char* mqtt_password = "<MQTT_PASSWORD>";

const char* root_ca PROGMEM = R"EOF(
-----BEGIN CERTIFICATE-----
<YOUR_CERTIFICATE_HERE>
-----END CERTIFICATE-----
)EOF";

#define PWDN_GPIO_NUM     32
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM      0
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27
#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22

WiFiClientSecure espClient;
PubSubClient client(espClient);

bool wifiConfigured = false;

void startAP() {
    WiFi.mode(WIFI_AP);
    WiFi.softAP("ESP32CAM_AP", "12345678");

    server.on("/", HTTP_GET, handleRoot);
    server.on("/configure", HTTP_POST, handleConfigure);
    server.begin();

    Serial.print("Access Point IP address: ");
    Serial.println(WiFi.softAPIP());
}

void setup() {
    Serial.begin(115200);
    espClient.setCACert(root_ca);
    String savedSSID;
```

```
    String savedPassword;

    delay(2000);
    Serial.println("Starting up...");

    if (loadWiFiCredentials(savedSSID, savedPassword)) {
        ssid = savedSSID;
        password = savedPassword;
        wifiConfigured = true;
        connectToWiFi();
    }

    if (!wifiConfigured) {
        startAP();
    }

    initCamera();

    client.setServer(mqtt_server, mqtt_port);
    client.setCallback(callback);
}

void loop() {
    if (!client.connected() && WiFi.status() == WL_CONNECTED) {
        reconnect();
    }
    client.loop();
    server.handleClient();
}

void initCamera() {
    camera_config_t config;
    config.ledc_channel = LEDC_CHANNEL_0;
    config.ledc_timer = LEDC_TIMER_0;
    config.pin_d0 = Y2_GPIO_NUM;
    config.pin_d1 = Y3_GPIO_NUM;
    config.pin_d2 = Y4_GPIO_NUM;
    config.pin_d3 = Y5_GPIO_NUM;
    config.pin_d4 = Y6_GPIO_NUM;
    config.pin_d5 = Y7_GPIO_NUM;
    config.pin_d6 = Y8_GPIO_NUM;
    config.pin_d7 = Y9_GPIO_NUM;
    config.pin_xclk = XCLK_GPIO_NUM;
    config.pin_pclk = PCLK_GPIO_NUM;
    config.pin_vsync = VSYNC_GPIO_NUM;
    config.pin_href = HREF_GPIO_NUM;
    config.pin_sscb_sda = SIOD_GPIO_NUM;
    config.pin_sscb_scl = SIOC_GPIO_NUM;
    config.pin_pwdn = PWDN_GPIO_NUM;
    config.pin_reset = RESET_GPIO_NUM;
    config.xclk_freq_hz = 20000000;
    config.pixel_format = PIXFORMAT_JPEG;
    config.frame_size = FRAMESIZE_VGA;
```

```
    config.jpeg_quality = 10;
    config.fb_count = 2;

    esp_err_t err = esp_camera_init(&config);
    if (err != ESP_OK) {
        Serial.printf("Camera init failed with error 0x%x", err);
        return;
    }
}

int checkLightingConditions() {
    camera_fb_t * fb = NULL;

    sensor_t * s = esp_camera_sensor_get();
    framesize_t original_resolution = s->status.framesize;
    s->set_framesize(s, FRAMESIZE_QVGA);

    fb = esp_camera_fb_get();
    if (!fb) {
        Serial.println("Camera capture failed");
        return -1;  // Return error code
    }

    long sum = 0;
    int pixelCount = fb->width * fb->height;
    uint8_t* pixels = fb->buf;

    for (int i = 0; i < pixelCount; i++) {
        sum += pixels[i];
    }

    float averageBrightness = sum / (float)pixelCount;

    esp_camera_fb_return(fb);

    s->set_framesize(s, (framesize_t)original_resolution);

    int lightLevel = map(averageBrightness, 0, 255, 0, 100);

    Serial.printf("Average brightness: %.2f, Light level: %d\n", averageBrightness,
     lightLevel);

    return lightLevel;
}

void adjustCameraSettings(int lightLevel) {
    sensor_t * s = esp_camera_sensor_get();

    if (lightLevel < 30) {  // Low light
        s->set_gain_ctrl(s, 1);  // Auto gain on
        s->set_exposure_ctrl(s, 1);  // Auto exposure on
        s->set_awb_gain(s, 1);  // Auto white balance gain on
    } else if (lightLevel > 70) {  // Bright light
```

```
        s->set_gain_ctrl(s, 0);  // Auto gain off
        s->set_exposure_ctrl(s, 0);  // Auto exposure off
        s->set_awb_gain(s, 0);  // Auto white balance gain off
        s->set_exposure_ctrl(s, 1);
        s->set_aec_value(s, 300);  // Set a fixed exposure time
    } else {  // Moderate light
        s->set_gain_ctrl(s, 1);  // Auto gain on
        s->set_exposure_ctrl(s, 1);  // Auto exposure on
        s->set_awb_gain(s, 1);  // Auto white balance gain on
    }

}

void connectToWiFi() {
    if (wifiConfigured) {
        Serial.print("Connecting to ");
        Serial.println(ssid);
        WiFi.mode(WIFI_STA);
        WiFi.begin(ssid.c_str(), password.c_str());

        unsigned long startTime = millis();
        while (WiFi.status() != WL_CONNECTED && millis() - startTime < 10000) {
            delay(500);
            Serial.print(".");
        }

        if (WiFi.status() == WL_CONNECTED) {
            Serial.println("");
            Serial.println("WiFi connected.");
        } else {
            Serial.println("");
            Serial.println("WiFi connection failed. Starting AP mode.");
            startAP();
        }
    } else {
        Serial.println("WiFi not configured");
    }
}

void saveWiFiCredentials(const String& ssid, const String& password) {
    preferences.begin("wifi-config", false);
    preferences.putString("ssid", ssid);
    preferences.putString("password", password);
    preferences.end();
}

bool loadWiFiCredentials(String& ssid, String& password) {
    preferences.begin("wifi-config", true);
    bool hasCredentials = preferences.isKey("ssid") && preferences.isKey("password")
     ;

    if (hasCredentials) {
        ssid = preferences.getString("ssid");
```

```
        password = preferences.getString("password");
    }

    preferences.end();
    return hasCredentials;
}

void reconnect() {
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        if (client.connect("ESP32CAMClient", mqtt_username, mqtt_password)) {
            Serial.println("connected");
            client.subscribe("esp32cam/capture");
            client.subscribe("esp32cam/wifi_config");
            client.subscribe("esp32cam/connect");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            delay(5000);
        }
    }
}

void handleWiFiConfig(String message) {
    int commaIndex = message.indexOf(',');
    if (commaIndex != -1) {
        String newSsid = message.substring(0, commaIndex);
        String newPassword = message.substring(commaIndex + 1);

        saveWiFiCredentials(newSsid, newPassword);

        client.publish("esp32cam/wifi_config_result", "WiFi credentials updated.
         Restarting...");
        delay(1000);
        ESP.restart();
    } else {
        client.publish("esp32cam/wifi_config_result", "Invalid WiFi configuration
         format");
    }
}

void handleConnectionRequest() {
    if (client.connected()) {
        client.publish("esp32cam/status", "connected");
    } else {
        client.publish("esp32cam/status", "disconnected");
    }
}

void callback(char* topic, byte* payload, unsigned int length) {
    String message = "";
    for (int i = 0; i < length; i++) {
```

```
        message += (char)payload[i];
    }

    if (String(topic) == "esp32cam/capture") {
        captureAndSendImage();
    } else if (String(topic) == "esp32cam/wifi_config") {
        handleWiFiConfig(message);
    } else if (String(topic) == "esp32cam/connect") {
        handleConnectionRequest();
    }
}

void captureAndSendImage() {
    int lightLevel = checkLightingConditions();

    adjustCameraSettings(lightLevel);

    camera_fb_t * fb = esp_camera_fb_get();
    if (!fb) {
        Serial.println("Camera capture failed");
        return;
    }

    if (client.connected()) {
        client.beginPublish("esp32cam/image", fb->len, false);
        client.write(fb->buf, fb->len);
        client.endPublish();
        Serial.println("Image sent to MQTT");
    }

    esp_camera_fb_return(fb);
}

void handleRoot() {
    String html = "<html><body>";
    html += "<h1>ESP32-CAM Configuration</h1>";
    html += "<form method='post' action='/configure'>";
    html += "SSID: <input type='text' name='ssid'><br>";
    html += "Password: <input type='password' name='password'><br>";
    html += "<input type='submit' value='Configure'>";
    html += "</form></body></html>";
    server.send(200, "text/html", html);
}

void handleConfigure() {
    String newSsid = server.arg("ssid");
    String newPassword = server.arg("password");
    saveWiFiCredentials(newSsid, newPassword);
    server.send(200, "text/plain", "Configuration saved. ESP32-CAM will restart.");
    delay(1000);
    ESP.restart();
}
```

3.2.3. AI model za prepoznavanje hrane

Model za prepoznavanje hrane je treniran na skupu podataka zvanom Food-101 [29] koji u sebi sadrži 101 kategoriju hrane sa 101000 slika. Slike za treniranje nisu namjerno očišćene te sadrže određenu količinu šuma kao što su jake boje ili ponekad i pogrešne labele.

Treniranje je izvršeno uz pomoć sljedeće Python skripte:

```
import os
import tarfile
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications.inception_v3 import InceptionV3, preprocess_input
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img,
 img_to_array
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
from urllib.request import urlretrieve
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

dataset_url = 'http://data.vision.ee.ethz.ch/cvl/food-101.tar.gz'
dataset_path = 'food-101.tar.gz'

if not os.path.exists(dataset_path):
    print('Downloading the Food-101 dataset...')
    urlretrieve(dataset_url, dataset_path)
    print('Dataset downloaded.')
else:
    print('Dataset already downloaded.')

if not os.path.exists('food-101'):
    print('Extracting the dataset...')
    with tarfile.open(dataset_path, 'r:gz') as tar_ref:
        tar_ref.extractall('.')
    print('Dataset extracted.')
else:
    print('Dataset already extracted.')

data_dir = os.path.join('food-101', 'images')
meta_dir = os.path.join('food-101', 'meta')

with open(os.path.join(meta_dir, 'classes.txt'), 'r') as f:
    classes = [line.strip() for line in f]

label_to_id = {class_label: i for i, class_label in enumerate(classes)}

with open(os.path.join(meta_dir, 'train.txt'), 'r') as f:
    train_data = [line.strip() for line in f]

```

```
with open(os.path.join(meta_dir, 'test.txt'), 'r') as f:
    test_data = [line.strip() for line in f]

train_images = [os.path.join(data_dir, line + '.jpg') for line in train_data]
train_labels = [label_to_id[line.split('/')[0]] for line in train_data]

test_images = [os.path.join(data_dir, line + '.jpg') for line in test_data]
test_labels = [label_to_id[line.split('/')[0]] for line in test_data]

train_images, val_images, train_labels, val_labels = train_test_split(train_images,
 train_labels, test_size=0.25, random_state=42)

def load_and_preprocess_image(img_path, target_size=(299, 299)):
    img = tf.io.read_file(img_path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, target_size)
    img = tf.cast(img, tf.float32)
    img = preprocess_input(img)
    return img

def custom_generator(images, labels, datagen, target_size=(299, 299), batch_size=32,
  shuffle=True):
    num_samples = len(images)
    while True:
        if shuffle:
            indices = np.random.permutation(num_samples)
        else:
            indices = np.arange(num_samples)

        for i in range(0, num_samples, batch_size):
            batch_indices = indices[i:i+batch_size]
            batch_images = []
            batch_labels = []

            for j in batch_indices:
                img_path = images[j]
                try:
                    img = load_and_preprocess_image(img_path, target_size)
                    img = datagen.random_transform(img.numpy())  # Apply random
                     transformations
                    batch_images.append(img)
                    batch_labels.append(labels[j])
                except Exception as e:
                    print(f"Error loading image {img_path}: {e}")

            if batch_images:
                yield tf.convert_to_tensor(batch_images), to_categorical(
                 batch_labels, num_classes=101)

def count_valid_images(image_list):
    count = 0
    for img_path in image_list:
        try:
```

```
            _ = load_and_preprocess_image(img_path)
            count += 1
        except:
            pass
    return count

train_datagen = ImageDataGenerator(rotation_range=20,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)

val_datagen = ImageDataGenerator()
test_datagen = ImageDataGenerator()

train_generator = custom_generator(train_images, train_labels, train_datagen,
 batch_size=32)
val_generator = custom_generator(val_images, val_labels, val_datagen, batch_size=32,
  shuffle=False)
test_generator = custom_generator(test_images, test_labels, test_datagen, batch_size
 =32, shuffle=False)

base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(299,
 299, 3))
for layer in base_model.layers:
    layer.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
predictions = Dense(101, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)

optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['
 accuracy'])

valid_train_images = count_valid_images(train_images)
valid_val_images = count_valid_images(val_images)
valid_test_images = count_valid_images(test_images)

train_steps = valid_train_images // 32 + (1 if valid_train_images % 32 != 0 else 0)
val_steps = valid_val_images // 32 + (1 if valid_val_images % 32 != 0 else 0)
test_steps = valid_test_images // 32 + (1 if valid_test_images % 32 != 0 else 0)

early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1,
 restore_best_weights=True)
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5,
 patience=5, min_lr=0.00001)

print(f"Training on {valid_train_images} valid images")
print(f"Validating on {valid_val_images} valid images")
print(f"Testing on {valid_test_images} valid images")
```

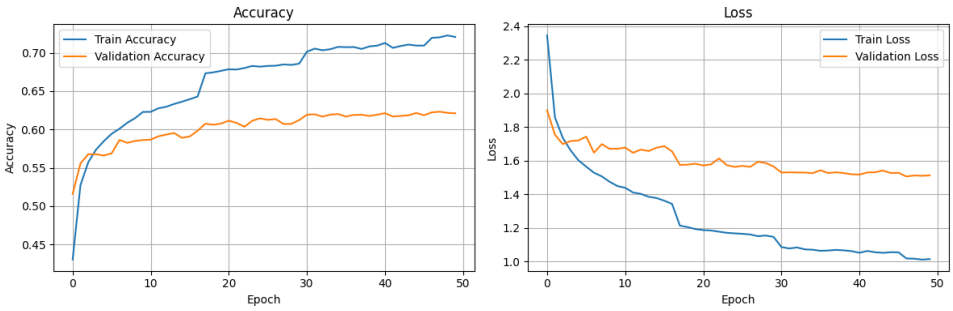
```
140
141 history = model.fit(train_generator,
142           validation_data=val_generator,
143           epochs=50,
144           steps_per_epoch=train_steps,
145           validation_steps=val_steps,
146           callbacks=[early_stopping, reduce_lr])
147
148 print(f"Model was trained for {len(history.history['loss'])} epochs")
149
150 plt.figure(figsize=(12, 4))
151
152 plt.subplot(1, 2, 1)
153 plt.plot(history.history['accuracy'], label='Train Accuracy')
154 plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
155 plt.title('Accuracy')
156 plt.xlabel('Epoch')
157 plt.ylabel('Accuracy')
158 plt.legend()
159 plt.grid()
160
161 plt.subplot(1, 2, 2)
162 plt.plot(history.history['loss'], label='Train Loss')
163 plt.plot(history.history['val_loss'], label='Validation Loss')
164 plt.title('Loss')
165 plt.xlabel('Epoch')
166 plt.ylabel('Loss')
167 plt.legend()
168 plt.grid()
169
170 plt.tight_layout()
171 plt.savefig('training_history.png')
172
173 test_loss, test_accuracy = model.evaluate(test_generator, steps=test_steps)
174 print(f'Test Loss: {test_loss}, Test Accuracy: {test_accuracy}')
175
176 model.save('retrained_inception_v3.h5')
177
178 print("Class Names:")
179 for i, class_name in enumerate(classes):
180     print(f"{i}: {class_name}")
```

Ova skripta služi za treniranje modela strojnog učenja za prepoznavanje različitih vrsta hrane na slikama. Započinje preuzimanjem skupa podataka. Sljedeći korak je priprema podataka za treniranje. Skripta učitava informacije o klasama hrane i stvara liste putanja do slika zajedno s njihovim oznakama. Podaci za treniranje se zatim dijele na skup za trening i skup za validaciju. Za efikasno rukovanje slikama, skripta definira funkcije za učitavanje i predobradu slika koristeći TensorFlow. Implementira se i generator podataka koji učitava slike u serijama i primjenjuje tehnike augmentacije podataka, što pomaže u poboljšanju generalizacije modela.

Srž skripte je priprema i treniranje modela. Koristi se transfer learning pristup, gdje se

predtrenirani Inception-v3 model prilagođava za specifičan zadatak prepoznavanja hrane. Na zadnji sloj ovog modela dodaju se novi slojevi za klasifikaciju 101 vrste hrane. Konfiguracija treninga uključuje postavljanje optimizatora, funkcije gubitka i metrika za praćenje performansi. Definiraju se i posebni mehanizmi poput ranog zaustavljanja i prilagodljive stope učenja, koji pomažu u optimizaciji procesa treniranja.

Tijekom treninga, model uči na pripremljenim podacima, a napredak se prati i bilježi. Nakon završetka treninga, skripta vizualizira rezultate crtajući grafove točnosti i gubitka kroz epohe treninga. Konačno, istrenirani model se evaluira na testnom skupu podataka kako bi se procijenila njegova stvarna učinkovitost. Model se zatim sprema na disk za buduću uporabu, a ispisuju se i imena klasa hrane koje model može prepoznati.



Slika 10: Točnost i gubitak tijekom epoha treniranja

Trenirani model je dosegao 72 posto točnosti unutar 50 epoha što je prihvatljivo za ovakve modele. Detaljnije se može vidjeti na slici 10.

Nakon treniranja, bilo je potrebno izraditi Python aplikaciju koja će koristiti trenirani model za prepoznavanje hrane sa slika koje kamera pošalje na MQTT. Slike koje su dohvaćene sa određenog kanala na MQTT brokeru prolaze predobradu te ih model pokušava prepoznati. Nakon toga se vraća rezultat na brokera za daljnju obradu.

```python
import base64
import tensorflow as tf
import numpy as np
from PIL import Image
import io
import json
import paho.mqtt.client as mqtt
import logging
from dotenv import load_dotenv

load_dotenv()

logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)

model = tf.keras.models.load_model('retrained_inception_v3.h5')

with open('labels.txt', 'r') as f:
    labels = [line.strip() for line in f.readlines()]

```

```
mqtt_client = mqtt.Client()
mqtt_client.username_pw_set(os.getenv('MQTT_USERNAME'), os.getenv('MQTT_PASSWORD'))
mqtt_client.tls_set()  # Enable TLS
mqtt_client.connect(os.getenv('MQTT_BROKER'), int(os.getenv('MQTT_PORT')), 60)

def process_image(image_data):
    try:
        logger.debug(f"Received image data of length: {len(image_data)} bytes")
        image = Image.open(io.BytesIO(image_data))

        logger.debug(f"Image format: {image.format}, Size: {image.size}, Mode: {
         image.mode}")

        image = image.resize((299, 299))  # Adjust size as needed
        image = np.array(image) / 255.0
        image = np.expand_dims(image, axis=0)

        predictions = model.predict(image)
        top_prediction = np.argmax(predictions[0])

        buffered = io.BytesIO()
        Image.fromarray((image[0] * 255).astype(np.uint8)).save(buffered, format="
         JPEG")
        img_str = base64.b64encode(buffered.getvalue()).decode()

        result = {
            'predicted_class': labels[top_prediction],
            'confidence': float(predictions[0][top_prediction]),
            'image_data': img_str
        }

        logger.debug(f"Prediction result: {result['predicted_class']}, Confidence: {
         result['confidence']}")

        mqtt_client.publish("model/prediction", json.dumps(result))
    except Exception as e:
        logger.error(f"Error processing image: {str(e)}", exc_info=True)
        mqtt_client.publish("model/error", json.dumps({"error": str(e)}))

def on_message(client, userdata, message):
    if message.topic == "esp32cam/image":
        logger.debug(f"Received message on topic {message.topic} with payload of
         length {len(message.payload)} bytes")
        process_image(message.payload)

mqtt_client.on_message = on_message
mqtt_client.subscribe("esp32cam/image")

mqtt_client.loop_forever()
```

3.2.4. Android aplikacija

Android aplikacija je komponenta koja pokreće sve komunikacije između drugih agenata. Korisnici se mogu prijaviti ili registrirati u aplikaciju, a nakon toga ih aplikacija traži da unesu svoje osnovne podatke kako bi se mogao odrediti optimalni unos kalorija i nutritivnih vrijednosti za tu osobu. Na prvom dijelu aplikacije korisnik može vidjeti svoj dnevni unos nutritivnih vrijednosti i kalorija, te može preći na unašanje jela uz pomoć uređaja, ili na povijest svih unešenih jela, ili na svoj profil na kojemu može mijenjati vrijednosti nekolicine varijabli.

Na povijesti unešenih jela, korisnik može vidjeti svako jelo koje je bilo uneseno uz pomoć uređaja te kalorije i nutritivne vrijednosti tog jela.

Ako korisnik odabere aktivnost za unašanje hrane, prvo slijedi provjera moguće li je povezati se s vagom i kamerom, a ako ne, slijedi konfiguracija WiFi podataka koje se trebaju proslijediti prema vagi i kameri. Nakon te konfiguracije i uspješnog pokušaja spajanja, korisnik može unesti hranu u bazu podataka. Uz to, moguća je i manualna kalibracija vage.

Svaki zahtjev i odgovor se šalje preko MQTT brokera. Prvo se šalje zahtjev za težinu prema vagi, i kada se dobije odgovor šalje se zahtjev za sliku prema kameri. Kamera nakon što uspješno fotografira hranu, šalje tu sliku prema Python aplikaciji koja uz pomoć AI modela prepoznaje hranu i vraća sliku te prepoznatu klasu hrane nazad aplikaciji.

Aplikacija onda prikazuje tu hranu korisniku te izračunate kalorije i nutritivne vrijednosti. Nutritivne vrijednosti su kalkulirane prema podacima sa sljedeće stranice: https://fdc.nal.usda.gov/api-guide.htmlbkmk-3. Korisnik može dodavati više hrana koje onda se spremaju u jedan objekt jela nakon što je korisnik gotov sa unašanjem.

3.2.4.1. Programski kod

Struktura programskog koda, uz koju postoje još i layout resursi za prikaz je sljedeća:

```
client
   MqttManager
controller
   AddMealActivity
   CalorieHistoryActivity
   CaptureFoodItemActivity
   InitialQuestionnaireActivity
   LoginActivity
   MainActivity
   RegisterActivity
   UserProfileActivity
   WebViewActivity
db
   FirebaseManager
model
```

```
    Meal
    MealItem
    NutritiveValues
    User
 util
     NutritionCalculator
```

Glavni entiteti su Meal i User, a unutar Meal se nalaze i MealItem te NutritiveValues koji sadrže više detalja o samom jelu. Korisnik može imati spojeno na sebe više jela.

```
data class Meal(
   val id: String = "",
   val userId: String = "",
   val name: String = "",
   val calories: Float = 0f,
   val timestamp: Long = System.currentTimeMillis(),
   val items: List<MealItem> = emptyList(),
   val totalNutritiveValues: NutritiveValues = NutritiveValues()
)
data class MealItem(
   val name: String = "",
   val weight: Float = 0f,
   val calories: Float = 0f,
   val nutritiveValues: NutritiveValues = NutritiveValues(),
   val imageData: String = ""
) {

   constructor() : this("", 0f, 0f, NutritiveValues(), "")
}
data class NutritiveValues(
   var protein: Float = 0f,
   var calcium: Float = 0f,
   var fat: Float = 0f,
   var carbohydrates: Float = 0f,
   var vitamins: Float = 0f
)
data class User(
   val uid: String = "",
   val email: String = "",
   val username: String = "",
   val isEmailVerified: Boolean = false,
   val height: Double = 0.0,
   val weight: Double = 0.0,
   val age: Int = 0,
   val desiredWeight: Double = 0.0,
   val activityLevel: Int = 0,
   val dailyCalories: Float = 0f,
   val dailyProtein: Float = 0f,
   val dailyFat: Float = 0f,
   val dailyCarbs: Float = 0f,
   val dailyVitamins: Float = 0f
```

Upravitelj baze jest FirebaseManager unutar koje se nalaze sve metode koje upravljaju Firebase bazom. Sve funkcije unutar klase su veoma jednostavne. Jedna zanimljivost koja je vrijedna spomena jest kalkulacija dnevno potrebnih nutritivnih vrijednosti CalculateDailyNeeds koja je bazirana na Mifflin-St Jeor funkciji koja kalkulira korištenje kalorija u mirujućem stanju.

```
object FirebaseManager {
    private val auth = FirebaseAuth.getInstance()
    private val db = FirebaseFirestore.getInstance()

    fun registerUser(email: String, password: String, username: String, callback: (Boolean, String) -> Unit) {
        auth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    val user = User(auth.uid!!, email, username, false)
                    db.collection("users").document(auth.uid!!).set(user)
                        .addOnSuccessListener {
                            auth.currentUser?.sendEmailVerification()
                            callback(true, "Verification email sent")
                        }
                        .addOnFailureListener { callback(false, it.message ?: "Error saving user") }
                } else {
                    callback(false, task.exception?.message ?: "Registration failed")
                }
            }
    }

    fun loginUser(email: String, password: String, callback: (Boolean, String) -> Unit) {
        auth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    if (auth.currentUser?.isEmailVerified == true) {
                        callback(true, "Login successful")
                    } else {
                        callback(false, "Please verify your email first")
                    }
                } else {
                    callback(false, task.exception?.message ?: "Login failed")
                }
            }
    }

    fun getCurrentUser(callback: (User?) -> Unit) {
        val uid = auth.uid
        if (uid != null) {
            db.collection("users").document(uid).get()
                .addOnSuccessListener { doc ->
                    val user = doc.toObject(User::class.java)
                    callback(user)
                }
                .addOnFailureListener { callback(null) }
```

```
        } else {
            callback(null)
        }
    }

    fun addMeal(meal: Meal, callback: (Boolean, String) -> Unit) {
        val db = FirebaseFirestore.getInstance()
        db.collection("meals")
            .add(meal)
            .addOnSuccessListener {
                callback(true, "Meal added successfully")
            }
            .addOnFailureListener { e ->
                callback(false, e.message ?: "Failed to add meal")
            }
    }

    fun getDailyNutrition(userId: String, callback: (DailyNutrition) -> Unit) {
        val startOfDay = Calendar.getInstance().apply {
            set(Calendar.HOUR_OF_DAY, 0)
            set(Calendar.MINUTE, 0)
            set(Calendar.SECOND, 0)
            set(Calendar.MILLISECOND, 0)
        }.timeInMillis

        db.collection("meals")
            .whereEqualTo("userId", userId)
            .whereGreaterThanOrEqualTo("timestamp", startOfDay)
            .get()
            .addOnSuccessListener { documents ->
                var totalCalories = 0f
                val totalNutritiveValues = NutritiveValues()

                for (document in documents) {
                    val meal = document.toObject(Meal::class.java)
                    totalCalories += meal.calories
                    totalNutritiveValues.protein += meal.totalNutritiveValues.protein
                    totalNutritiveValues.calcium += meal.totalNutritiveValues.calcium
                    totalNutritiveValues.fat += meal.totalNutritiveValues.fat
                    totalNutritiveValues.carbohydrates += meal.totalNutritiveValues.
                     carbohydrates
                    totalNutritiveValues.vitamins += meal.totalNutritiveValues.
                     vitamins
                }

                val dailyNutrition = DailyNutrition(totalCalories,
                 totalNutritiveValues)
                callback(dailyNutrition)
            }
            .addOnFailureListener { exception ->
                println("Error getting daily nutrition: ${exception.message}")
                callback(DailyNutrition(0f, NutritiveValues()))
            }
```

```
    }

    fun getMeals(userId: String, callback: (List<Meal>) -> Unit) {
        db.collection("meals")
            .whereEqualTo("userId", userId)
            .orderBy("timestamp", Query.Direction.DESCENDING)
            .get()
            .addOnSuccessListener { documents ->
                val meals = documents.mapNotNull { it.toObject(Meal::class.java) }
                callback(meals)
            }
            .addOnFailureListener { exception ->
                println("Error getting meals: ${exception.message}")
                callback(emptyList())
            }
    }

    fun resetPassword(email: String, callback: (Boolean, String) -> Unit) {
        auth.sendPasswordResetEmail(email)
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    callback(true, "Password reset email sent to $email")
                } else {
                    callback(false, task.exception?.message ?: "Failed to send reset
                     email")
                }
            }
    }
    fun updateUser(user: User, callback: (Boolean) -> Unit) {
        val updatedUser = calculateDailyNeeds(user)
        db.collection("users").document(user.uid).set(updatedUser)
            .addOnSuccessListener {
                callback(true)
            }
            .addOnFailureListener { callback(false) }
    }


    private fun calculateDailyNeeds(user: User) {
        val bmr = when {
            user.weight > 0 && user.height > 0 && user.age > 0 -> {
                // Mifflin-St Jeor Equation
                (10 * user.weight) + (6.25 * user.height) - (5 * user.age) + 5
            }
            else -> 2000.0 // Default value if data is missing
        }

        val activityFactor = when (user.activityLevel) {
            1 -> 1.2
            2 -> 1.375
            3 -> 1.55
            4 -> 1.725
            5 -> 1.9
```

```
            else -> 1.2
        }

        val dailyCalories = (bmr * activityFactor).toFloat()
        val dailyProtein = (dailyCalories * 0.3 / 4).toFloat()
        val dailyFat = (dailyCalories * 0.3 / 9).toFloat()
        val dailyCarbs = (dailyCalories * 0.4 / 4).toFloat()

        val updatedUser = user.copy(
            dailyCalories = dailyCalories,
            dailyProtein = dailyProtein,
            dailyFat = dailyFat,
            dailyCarbs = dailyCarbs,
        )

        db.collection("users").document(user.uid).set(updatedUser)
    }

    data class DailyNutrition(
        val calories: Float,
        val totalNutritiveValues: NutritiveValues
    )

}
```

Krenuvši od početka korisničkog iskustva, prva klasa u kodu bila bi LoginActivity koja služi za autorizaciju korisnika i ulaz u samu aplikaciju. Postoji glavna metoda login() koja provjerava korisnikov email i lozinku, te showForgotPasswordDialog() za mijenjanje lozinke.

```
class LoginActivity : AppCompatActivity() {
    private lateinit var etEmail: EditText
    private lateinit var etPassword: EditText
    private lateinit var btnLogin: Button
    private lateinit var tvRegister: TextView
    private lateinit var tvForgotPassword: TextView

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_login)

        etEmail = findViewById(R.id.etEmail)
        etPassword = findViewById(R.id.etPassword)
        btnLogin = findViewById(R.id.btnLogin)
        tvRegister = findViewById(R.id.tvRegister)
        tvForgotPassword = findViewById(R.id.tvForgotPassword)


        btnLogin.setOnClickListener { login() }
        tvRegister.setOnClickListener { startActivity(Intent(this, RegisterActivity::
         class.java)) }
        tvForgotPassword.setOnClickListener { showForgotPasswordDialog() }

        FirebaseManager.getCurrentUser { user ->
```

```
            if (user != null && user.isEmailVerified) {
                startActivity(Intent(this, MainActivity::class.java))
                finish()
            }
        }
    }

    private fun showForgotPasswordDialog() {
        val builder = AlertDialog.Builder(this)
        val inflater = layoutInflater
        val dialogLayout = inflater.inflate(R.layout.dialog_forgot_password, null)
        val etEmail = dialogLayout.findViewById<EditText>(R.id.etEmailForgotPassword)

        with(builder) {
            setTitle("Forgot Password")
            setPositiveButton("Reset") { dialog, which ->
                val email = etEmail.text.toString()
                if (email.isNotEmpty()) {
                    FirebaseManager.resetPassword(email) { success, message ->
                        Toast.makeText(this@LoginActivity, message, Toast.LENGTH_LONG
                         ).show()
                    }
                } else {
                    Toast.makeText(this@LoginActivity, "Please enter your email",
                     Toast.LENGTH_SHORT).show()
                }
            }
            setNegativeButton("Cancel") { dialog, which ->
                dialog.dismiss()
            }
            setView(dialogLayout)
            show()
        }
    }

    private fun login() {
        val email = etEmail.text.toString()
        val password = etPassword.text.toString()
        if (email.isNotEmpty() && password.isNotEmpty()) {
            FirebaseManager.loginUser(email, password) { success, message ->
                Toast.makeText(this, message, Toast.LENGTH_SHORT).show()
                if (success) {
                    FirebaseManager.getCurrentUser { user ->
                        if (user != null) {
                            if (user.height == 0.0 || user.weight == 0.0) {
                                startActivity(Intent(this,
                                 InitialQuestionnaireActivity::class.java))
                            } else {
                                startActivity(Intent(this, MainActivity::class.java))
                            }
                            finish()
                        }
                    }
```

```
                }
            }
        } else {
            Toast.makeText(this, "Please fill all fields", Toast.LENGTH_SHORT).show()
        }
    }
}
```

Uz login, postoji naravno i registracija ako korisnik nije u sustavu. Jedina bitna metoda ovdje jest register() koja postavlja novog korisnika u bazu.

```
class RegisterActivity : AppCompatActivity() {
    private lateinit var etUsername: EditText
    private lateinit var etEmail: EditText
    private lateinit var etPassword: EditText
    private lateinit var btnRegister: Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_register)

        etUsername = findViewById(R.id.etUsername)
        etEmail = findViewById(R.id.etEmail)
        etPassword = findViewById(R.id.etPassword)
        btnRegister = findViewById(R.id.btnRegister)

        btnRegister.setOnClickListener { register() }
    }

    private fun register() {
        val username = etUsername.text.toString()
        val email = etEmail.text.toString()
        val password = etPassword.text.toString()
        if (username.isNotEmpty() && email.isNotEmpty() && password.isNotEmpty()) {
            FirebaseManager.registerUser(email, password, username) { success,
             message ->
                Toast.makeText(this, message, Toast.LENGTH_SHORT).show()
                if (success) {
                    finish()
                }
            }
        } else {
            Toast.makeText(this, "Please fill all fields", Toast.LENGTH_SHORT).show()
        }
    }
}
```

Nakon toga, pri prvom ulazu u aplikaciju, korisniku se otvara upitnik koji ga pita za njegove navike, visinu, težinu, starost i željenu težinu. Ti se podaci spremaju za daljnju obradu.

```
        class InitialQuestionnaireActivity : AppCompatActivity() {
    private lateinit var etHeight: EditText
    private lateinit var etWeight: EditText
```

```
    private lateinit var etAge: EditText
    private lateinit var etDesiredWeight: EditText
    private lateinit var rgActivityLevel: RadioGroup
    private lateinit var btnSubmit: Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_initial_questionnaire)

        initializeViews()

        btnSubmit.setOnClickListener { submitQuestionnaire() }
    }

    private fun initializeViews() {
        etHeight = findViewById(R.id.etHeight)
        etWeight = findViewById(R.id.etWeight)
        etAge = findViewById(R.id.etAge)
        etDesiredWeight = findViewById(R.id.etDesiredWeight)
        rgActivityLevel = findViewById(R.id.rgActivityLevel)
        btnSubmit = findViewById(R.id.btnSubmit)
    }

    private fun submitQuestionnaire() {
        val height = etHeight.text.toString().toDoubleOrNull() ?: 0.0
        val weight = etWeight.text.toString().toDoubleOrNull() ?: 0.0
        val age = etAge.text.toString().toIntOrNull() ?: 0
        val desiredWeight = etDesiredWeight.text.toString().toDoubleOrNull() ?: 0.0
        val activityLevel = when (rgActivityLevel.checkedRadioButtonId) {
            R.id.rbSedentary -> 1
            R.id.rbLightlyActive -> 2
            R.id.rbModeratelyActive -> 3
            R.id.rbVeryActive -> 4
            R.id.rbExtremelyActive -> 5
            else -> 1
        }

        FirebaseManager.getCurrentUser { user ->
            if (user != null) {
                val updatedUser = user.copy(
                    height = height,
                    weight = weight,
                    age = age,
                    desiredWeight = desiredWeight,
                    activityLevel = activityLevel
                )
                FirebaseManager.updateUser(updatedUser) { success ->
                    if (success) {
                        startActivity(Intent(this, MainActivity::class.java))
                        finish()
                    }
                }
            }
```

```
        }
    }
}
```

Nastavljajući dalje, korisnik dolazi na glavnu stranicu na kojoj se nalaze i kružni grafikoni koji prikazuju dnevni unos kalorija i ostalih nutritivnih vrijednosti. Uz to se nalaze i gumbi za prijelaz u daljnje aktivnosti kao što su povijest, profil ili unos jela.

```
        class MainActivity : AppCompatActivity() {
    private lateinit var tvWelcome: TextView
    private lateinit var caloriesProgressIndicator: CircularProgressIndicator
    private lateinit var proteinProgressIndicator: CircularProgressIndicator
    private lateinit var fatProgressIndicator: CircularProgressIndicator
    private lateinit var carbsProgressIndicator: CircularProgressIndicator
    private lateinit var tvCaloriesValue: TextView
    private lateinit var tvProteinValue: TextView
    private lateinit var tvFatValue: TextView
    private lateinit var tvCarbsValue: TextView
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        initializeViews()

        FirebaseManager.getCurrentUser { user ->
            if (user != null) {
                tvWelcome.text = "Welcome, ${user.username}"
                updateNutritionProgress(user)
            } else {
                startActivity(Intent(this, LoginActivity::class.java))
                finish()
            }
        }
    }

    private fun initializeViews() {
        tvWelcome = findViewById(R.id.tvWelcome)

        val caloriesLayout = findViewById<View>(R.id.caloriesLayout)
        val proteinLayout = findViewById<View>(R.id.proteinLayout)
        val fatLayout = findViewById<View>(R.id.fatLayout)
        val carbsLayout = findViewById<View>(R.id.carbsLayout)

        caloriesProgressIndicator = caloriesLayout.findViewById(R.id.
         progressIndicator)
        proteinProgressIndicator = proteinLayout.findViewById(R.id.progressIndicator)
        fatProgressIndicator = fatLayout.findViewById(R.id.progressIndicator)
        carbsProgressIndicator = carbsLayout.findViewById(R.id.progressIndicator)

        tvCaloriesValue = caloriesLayout.findViewById(R.id.tvValue)
        tvProteinValue = proteinLayout.findViewById(R.id.tvValue)
        tvFatValue = fatLayout.findViewById(R.id.tvValue)
        tvCarbsValue = carbsLayout.findViewById(R.id.tvValue)
```

```
45
46          caloriesLayout.findViewById<TextView>(R.id.tvLabel).text = "Calories"
47          proteinLayout.findViewById<TextView>(R.id.tvLabel).text = "Protein"
48          fatLayout.findViewById<TextView>(R.id.tvLabel).text = "Fat"
49          carbsLayout.findViewById<TextView>(R.id.tvLabel).text = "Carbs"
50
51          caloriesProgressIndicator.progress = 0
52          proteinProgressIndicator.progress = 0
53          fatProgressIndicator.progress = 0
54          carbsProgressIndicator.progress = 0
55      }
56
57      private fun updateNutritionProgress(user: User) {
58          FirebaseManager.getDailyNutrition(user.uid) { dailyNutrition ->
59              updateProgressIndicator(caloriesProgressIndicator, tvCaloriesValue,
                 dailyNutrition.calories, user.dailyCalories, "kcal", R.color.
                 colorCalories)
60              updateProgressIndicator(proteinProgressIndicator, tvProteinValue,
                 dailyNutrition.totalNutritiveValues.protein, user.dailyProtein, "g", R.
                 color.colorProtein)
61              updateProgressIndicator(fatProgressIndicator, tvFatValue, dailyNutrition.
                 totalNutritiveValues.fat, user.dailyFat, "g", R.color.colorFat)
62              updateProgressIndicator(carbsProgressIndicator, tvCarbsValue,
                 dailyNutrition.totalNutritiveValues.carbohydrates, user.dailyCarbs, "g",
                  R.color.colorCarbs)
63          }
64      }
65
66      private fun updateProgressIndicator(indicator: CircularProgressIndicator,
         textView: TextView, value: Float, maxValue: Float, unit: String, colorResId: Int
         ) {
67          val progress = ((value / maxValue) * 100).toInt().coerceIn(0, 100)
68          ObjectAnimator.ofInt(indicator, "progress", 0, progress).apply {
69              duration = 1000
70              interpolator = DecelerateInterpolator()
71              start()
72          }
73          indicator.setIndicatorColor(ContextCompat.getColor(this, colorResId))
74          textView.text = "${value.toInt()} / ${maxValue.toInt()} $unit"
75      }
76
77      fun openAddMealActivity(view: View) {
78          startActivity(Intent(this, AddMealActivity::class.java))
79      }
80
81      fun openCalorieHistoryActivity(view: View) {
82          startActivity(Intent(this, CalorieHistoryActivity::class.java))
83      }
84      fun openUserProfileActivity(view: View) {
85          startActivity(Intent(this, UserProfileActivity::class.java))
86      }
87  }
```

Klasa UserProfileActivity unutar sebe sadrži samo vrijednosti koje su bile postavljene pri početnom upitniku te se one mogu i mijenjati. Nakon promjena se preračunavaju vrijednosti dnevnog unosa nutritivnih vrijednosti.

```
      class UserProfileActivity : AppCompatActivity() {
    private lateinit var etHeight: TextInputEditText
    private lateinit var etWeight: TextInputEditText
    private lateinit var etAge: TextInputEditText
    private lateinit var etDesiredWeight: TextInputEditText
    private lateinit var rgActivityLevel: RadioGroup
    private lateinit var btnSave: Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_user_profile)

        initializeViews()
        loadUserProfile()

        btnSave.setOnClickListener {
            saveUserProfile()
        }
    }

    private fun initializeViews() {
        etHeight = findViewById(R.id.etHeight)
        etWeight = findViewById(R.id.etWeight)
        etAge = findViewById(R.id.etAge)
        etDesiredWeight = findViewById(R.id.etDesiredWeight)
        rgActivityLevel = findViewById(R.id.rgActivityLevel)
        btnSave = findViewById(R.id.btnSave)
    }

    private fun loadUserProfile() {
        FirebaseManager.getCurrentUser { user ->
            if (user != null) {
                etHeight.setText(user.height.toString())
                etWeight.setText(user.weight.toString())
                etAge.setText(user.age.toString())
                etDesiredWeight.setText(user.desiredWeight.toString())
                when (user.activityLevel) {
                    1 -> rgActivityLevel.check(R.id.rbSedentary)
                    2 -> rgActivityLevel.check(R.id.rbLightlyActive)
                    3 -> rgActivityLevel.check(R.id.rbModeratelyActive)
                    4 -> rgActivityLevel.check(R.id.rbVeryActive)
                    5 -> rgActivityLevel.check(R.id.rbExtremelyActive)
                }
            }
        }
    }

    private fun saveUserProfile() {
        val height = etHeight.text.toString().toDoubleOrNull() ?: 0.0
```

```
        val weight = etWeight.text.toString().toDoubleOrNull() ?: 0.0
        val age = etAge.text.toString().toIntOrNull() ?: 0
        val desiredWeight = etDesiredWeight.text.toString().toDoubleOrNull() ?: 0.0
        val activityLevel = when (rgActivityLevel.checkedRadioButtonId) {
            R.id.rbSedentary -> 1
            R.id.rbLightlyActive -> 2
            R.id.rbModeratelyActive -> 3
            R.id.rbVeryActive -> 4
            R.id.rbExtremelyActive -> 5
            else -> 1
        }

        FirebaseManager.getCurrentUser { currentUser ->
            if (currentUser != null) {
                val updatedUser = currentUser.copy(
                    height = height,
                    weight = weight,
                    age = age,
                    desiredWeight = desiredWeight,
                    activityLevel = activityLevel
                )
                FirebaseManager.updateUser(updatedUser) { success ->
                    if (success) {
                        Toast.makeText(this, "Profile updated successfully", Toast.
                         LENGTH_SHORT).show()
                        finish()
                    } else {
                        Toast.makeText(this, "Failed to update profile", Toast.
                         LENGTH_SHORT).show()
                    }
                }
            }
        }
    }
}
```

AddMealActivity je nešto kompleksnija klasa. Kada korisnik klikne na gumb za spajanje sa uređajem, zove se mqttManager klasa koja služi za spajanje sa MQTT brokerom. Prvo se spaja na brokera, te pretplaćuje na temu za provjeru konekcije s vagom. Ako vaga pošalje preko MQTT odgovor da je komunikacija ok, program ide dalje, no ako nije korisniku se izbacuje prozor za postavljanje WiFi konfiguracije na vagu. Nakon toga se događa ista stvar sa kamerom, te ako je sve na kraju uredu omogućava se korisniku ili kalibracija vage ili pokretanje aktivnosti za unos jela. Vaga se također kalibrira uz pomoć MQTT teme esp32/calibrate. Uz te dvije opcije omogućeno je i ponovno postavljanje WiFi postavki na oba uređaja uz pomoć MQTT. Ako su postavke dobro poslane, uređaji se sami ponovno spajaju i korisnik ne mora ništa drugo postavljati.

```
class AddMealActivity : AppCompatActivity() {
    private lateinit var statusBar: TextView
    private lateinit var btnConnectDevices: Button
    private lateinit var btnChangeWifi: Button
```

```
    private lateinit var btnCalibrate: Button
    private lateinit var btnCaptureFoodItem: Button
    private lateinit var progressBar: ProgressBar
    private lateinit var tvLoading: TextView
    private lateinit var tvStatus: TextView
    private lateinit var mqttManager: MqttManager

    private var isCameraConnected = false
    private var isScaleConnected = false
    private lateinit var wifiManager: WifiManager

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_add_meal)
        initializeViews()
        setUIEnabled(false)
        setListeners()
        mqttManager = MqttManager(this)
    }

    private fun initializeViews() {
        statusBar = findViewById(R.id.statusBar)
        btnConnectDevices = findViewById(R.id.btnConnectScale)
        btnChangeWifi = findViewById(R.id.btnChangeWifi)
        btnCalibrate = findViewById(R.id.btnCalibrate)
        btnCaptureFoodItem = findViewById(R.id.btnCaptureFoodItem)
        progressBar = findViewById(R.id.progressBar)
        tvLoading = findViewById(R.id.tvLoading)
        tvStatus = findViewById(R.id.tvStatus)
        updateStatusText("Ready to connect")
    }

    private fun updateStatusText(text: String) {
        if (!isFinishing && !isDestroyed) {
            runOnUiThread { tvStatus.text = text }
        }
    }

    private fun setListeners() {
        btnConnectDevices.setOnClickListener { connectToDevices() }
        btnChangeWifi.setOnClickListener { changeWifiCredentials() }
        btnCalibrate.setOnClickListener { calibrateScale() }
        btnCaptureFoodItem.setOnClickListener { startCaptureFoodItemActivity() }
    }

    private fun connectToDevices() {
        isCameraConnected = false
        isScaleConnected = false
        lifecycleScope.launch {
            if (mqttManager.connect()) {
                updateStatusText("Connected to MQTT broker")
                subscribeToEsp32()
            } else {
```

```
                updateStatusText("Failed to connect to MQTT broker")
            }
        }
    }

    private fun subscribeToEsp32() {
        isScaleConnected = false

        mqttManager.subscribe("esp32/ping", 1) { _, message ->
            handleEsp32PingResponse(message.toString())
        }
        mqttManager.publish("esp32/ping", "ping")
        updateStatusText("Checking ESP32 reachability...")

        Handler(Looper.getMainLooper()).postDelayed({
            if (!isScaleConnected) {
                handleDeviceUnreachable("esp32")
            } else {
                isCameraConnected = false
                mqttManager.unsubscribe("esp32/ping")
                subscribeToEsp32Cam()
            }
        }, 5000)
    }

    private fun subscribeToEsp32Cam() {

        mqttManager.subscribe("esp32cam/ping", 1) { _, message ->
            handleEsp32CamPingResponse(message.toString())
        }
        mqttManager.publish("esp32cam/ping", "ping")
        updateStatusText("Checking ESP32Cam reachability...")

        Handler(Looper.getMainLooper()).postDelayed({
            if (!isCameraConnected) {
                handleDeviceUnreachable("esp32cam")
            } else {
                mqttManager.unsubscribe("esp32cam/ping")
                setUIEnabled(true)
            }
        }, 10000)
    }

    private fun handleEsp32PingResponse(message: String) {
        if (message == "pong") {
            isScaleConnected = true
            updateStatusText("Connected to ESP32, Trying to connect to Cam...")
        }
    }

    private fun handleEsp32CamPingResponse(message: String) {
        if (message == "pong") {
            isCameraConnected = true
```

```
            updateStatusText("Connected to ESP32Cam")
            updateConnectionStatus()
        }
    }
    private fun handleDeviceUnreachable(device: String) {
        updateStatusText("$device not reachable. Setting up Wi-Fi...")
        when (device) {
            "esp32cam" -> configureDevice(getProperty("ESP32CAM_AP_SSID"),
             getProperty("ESP32CAM_AP_PASSWORD"), getProperty("ESP32_AP_URL"),
             CAMERA_CONFIG_REQUEST)
            "esp32" -> configureDevice(getProperty("ESP32_AP_SSID"), getProperty("
             ESP32_AP_PASSWORD"), getProperty("ESP32_AP_URL"), ESP32_CONFIG_REQUEST)
        }
    }

    private fun updateConnectionStatus() {
        if (isScaleConnected && isCameraConnected) {
            runOnUiThread {
                statusBar.text = "Connected"
                statusBar.setBackgroundColor(resources.getColor(R.color.green))
                btnConnectDevices.visibility = View.GONE
                btnChangeWifi.visibility = View.VISIBLE
                btnCalibrate.visibility = View.VISIBLE
            }
        } else {
            runOnUiThread {
                statusBar.text = "Disconnected"
                statusBar.setBackgroundColor(resources.getColor(R.color.red))
                btnConnectDevices.visibility = View.VISIBLE
                btnChangeWifi.visibility = View.GONE
                btnCalibrate.visibility = View.GONE
            }
        }
    }
    private fun configureDevice(ssid: String, password: String, url: String,
     configRequestCode: Int) {
        wifiManager = applicationContext.getSystemService(Context.WIFI_SERVICE) as
         WifiManager

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
            promptManualWifiConnection(ssid, password)
        } else {
            connectToAP(ssid, password) {
                openDeviceConfigurationWebsite(url, configRequestCode)
            }
        }
    }

    private fun connectToAP(apSsid: String, apPassword: String, onSuccess: () -> Unit
     ) {
        val conf = WifiConfiguration().apply {
            SSID = "\"$apSsid\""
            preSharedKey = "\"$apPassword\""
```

```
        }
        val networkId = wifiManager.addNetwork(conf)
        wifiManager.disconnect()
        wifiManager.enableNetwork(networkId, true)
        wifiManager.reconnect()

        Handler(Looper.getMainLooper()).postDelayed({
            if (wifiManager.connectionInfo.ssid == "\"$apSsid\"") {
                onSuccess()
            } else {
                promptManualWifiConnection(apSsid, apPassword)
            }
        }, 5000) // Wait 5 seconds for connection
    }

    private fun promptManualWifiConnection(ssid: String, password: String) {
        AlertDialog.Builder(this)
            .setTitle("Manual Wi-Fi Connection Required")
            .setMessage("Please connect to the '$ssid' Wi-Fi network manually. The
             password is '$password'. After connecting, return to this app.")
            .setPositiveButton("Open Wi-Fi Settings") { _, _ ->
                startActivityForResult(Intent(Settings.ACTION_WIFI_SETTINGS),
                 WIFI_SETTINGS_REQUEST_CODE)
            }
            .setCancelable(false)
            .show()
    }

    private fun reconnectToOriginalNetwork() {
        val userSsid = getProperty("USER_SSID")
        val userPassword = getProperty("USER_PASSWORD")

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
            promptManualReconnection(userSsid)
        } else {
            connectToAP(userSsid, userPassword) {
                updateStatusText("Reconnected to original network. Retrying
                 connection...")
                connectToDevices()
            }
        }
    }

    private fun promptManualReconnection(ssid: String) {
        AlertDialog.Builder(this)
            .setTitle("Reconnect to Original Network")
            .setMessage("Please reconnect to your original Wi-Fi network '$ssid'
             manually.")
            .setPositiveButton("Open Wi-Fi Settings") { _, _ ->
                startActivity(Intent(Settings.ACTION_WIFI_SETTINGS))
            }
            .setNegativeButton("I'm Reconnected") { _, _ ->
                updateStatusText("Retrying connection...")
```

```
                connectToDevices()
            }
            .setCancelable(false)
            .show()
    }

    private fun openDeviceConfigurationWebsite(url: String, requestCode: Int) {
        val intent = Intent(this, WebViewActivity::class.java).apply {
            putExtra("url", url)
        }
        startActivityForResult(intent, requestCode)
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        when (requestCode) {
            WIFI_SETTINGS_REQUEST_CODE -> {
                if (isConnectedToDeviceAp()) {
                    openDeviceConfigurationWebsite(getProperty("ESP32_AP_URL"),
                     ESP32_CONFIG_REQUEST)
                } else {
                    promptManualWifiConnection(getProperty("ESP32_AP_SSID"),
                     getProperty("ESP32_AP_PASSWORD"))
                }
            }
            ESP32_CONFIG_REQUEST, CAMERA_CONFIG_REQUEST -> {
                if (resultCode == Activity.RESULT_OK) {
                    reconnectToOriginalNetwork()
                } else {
                    updateStatusText("Configuration failed. Retrying...")
                    Handler(Looper.getMainLooper()).postDelayed({
                        reconnectToOriginalNetwork()
                    }, 2000) // Retry after 2 seconds
                }
            }
        }
    }

    private fun isConnectedToDeviceAp(): Boolean {
        val ssid = wifiManager.connectionInfo.ssid
        return ssid == "\"${getProperty("ESP32_AP_SSID")}\"" || ssid == "\"${
         getProperty("ESP32CAM_AP_SSID")}\""
    }

    private fun startCaptureFoodItemActivity() {
        val intent = Intent(this, CaptureFoodItemActivity::class.java)
        startActivity(intent)
    }

    private fun setUIEnabled(enabled: Boolean) {
        runOnUiThread {
            btnCalibrate.isEnabled = enabled
            btnCaptureFoodItem.isEnabled = enabled
```

```
        }
    }

    private fun calibrateScale() {
        showCalibrationPrompt { shouldCalibrate ->
            if (shouldCalibrate) {
                mqttManager.publish("esp32/calibrate", "calibrate")
                updateStatusText("Sent calibration request to ESP32")
                showConfirmationPrompt { confirmed ->
                    if (confirmed) {
                        mqttManager.publish("esp32/calibrate/confirm", "confirm")
                        updateStatusText("Sent calibration confirmation to ESP32")
                    } else {
                        updateStatusText("Calibration cancelled")
                    }
                }
            }
        }
    }

    private fun showCalibrationPrompt(callback: (Boolean) -> Unit) {
        AlertDialog.Builder(this)
            .setTitle("Calibrate Scale")
            .setMessage("Do you want to calibrate the scale?")
            .setPositiveButton("Yes") { _, _ -> callback(true) }
            .setNegativeButton("No") { _, _ -> callback(false) }
            .show()
    }

    private fun showConfirmationPrompt(callback: (Boolean) -> Unit) {
        AlertDialog.Builder(this)
            .setTitle("Calibration")
            .setMessage("Place the calibration object on the scale and click 'Confirm
             '")
            .setPositiveButton("Confirm") { _, _ -> callback(true) }
            .setNegativeButton("Cancel") { _, _ -> callback(false) }
            .show()
    }

    private fun changeWifiCredentials() {
        val ssidEditText = EditText(this)
        val passwordEditText = EditText(this)

        AlertDialog.Builder(this)
            .setTitle("Change WiFi Credentials")
            .setView(LinearLayout(this).apply {
                orientation = LinearLayout.VERTICAL
                addView(TextView(context).apply { text = "SSID" })
                addView(ssidEditText)
                addView(TextView(context).apply { text = "Password" })
                addView(passwordEditText)
            })
            .setPositiveButton("Submit") { _, _ ->
```

```
                val newSsid = ssidEditText.text.toString()
                val newPassword = passwordEditText.text.toString()
                sendNewWifiCredentials(newSsid, newPassword)
            }
            .setNegativeButton("Cancel", null)
            .show()
    }

    private fun sendNewWifiCredentials(ssid: String, password: String) {
        mqttManager.publish("esp32/wifi_config", "$ssid,$password")
        updateStatusText("Sent new WiFi credentials to ESP32")
        mqttManager.publish("esp32cam/wifi_config", "$ssid,$password")
        updateStatusText("Sent new WiFi credentials to ESP32")
        isScaleConnected = false
        isCameraConnected = false
        mqttManager.subscribe("esp32/wifi_config_result", 0) { topic, message ->
            val result = message.toString()
            updateStatusText(result)
            if (result.contains("Restarting")) {
                Handler(Looper.getMainLooper()).postDelayed({
                    updateConnectionStatus()
                    connectToDevices()
                }, 5000)
            }
        }
    }
    private fun getProperty(propertyName: String): String {
        val properties = Properties().apply {
            val inputStream: InputStream = assets.open("config.properties")
            load(inputStream)
        }
        return properties.getProperty(propertyName)
    }

    companion object {
        const val WIFI_SETTINGS_REQUEST_CODE = 1001
        const val ESP32_CONFIG_REQUEST = 1002
        const val CAMERA_CONFIG_REQUEST = 1003
    }
}
```

Prijašnje spomenuti MqttManager se koristi za upravljanje konekcijom sa MQTT brokerom. Postavlja se konfiguracija i SSL certifikat, a uz to postoje i tri glavne funkcije: publish(), subscribe() i unsubscribe() koje se pretplaćuju, makivaju pretplatu ili šalju podatke na određenu temu.

```
class MqttManager(private val context: Context) {
    private lateinit var mqttClient: MqttAndroidClient
    private lateinit var mqttConfig: JSONObject

    suspend fun connect(): Boolean {
        loadMqttConfig()
        return connectMqtt()
```

```
    }

    private fun loadMqttConfig() {
        try {
            val inputStream = context.assets.open("mqtt_config.json")
            val size = inputStream.available()
            val buffer = ByteArray(size)
            inputStream.read(buffer)
            inputStream.close()
            val json = String(buffer, Charsets.UTF_8)
            mqttConfig = JSONObject(json)
        } catch (e: Exception) {
            e.printStackTrace()
            throw IllegalStateException("Error loading MQTT config")
        }
    }

    private suspend fun connectMqtt(): Boolean = withContext(Dispatchers.IO) {
        try {
            val serverUri = "ssl://${mqttConfig.getString("server")}:${mqttConfig.
             getInt("port")}"
            val clientId = "AndroidApp_${System.currentTimeMillis()}"

            mqttClient = MqttAndroidClient(context, serverUri, clientId)
            val options = MqttConnectOptions().apply {
                isCleanSession = true
                userName = mqttConfig.getString("username")
                password = mqttConfig.getString("password").toCharArray()
                connectionTimeout = 30
                keepAliveInterval = 60
                socketFactory = getSSLSocketFactory()
            }

            val token = mqttClient.connect(options)
            token.waitForCompletion()
            true
        } catch (e: Exception) {
            Log.e("MqttManager", "Error in MQTT connection: ${e.message}")
            false
        }
    }

    private fun getSSLSocketFactory(): SSLSocketFactory? {
        return try {
            val cf = CertificateFactory.getInstance("X.509")
            val caInput: InputStream = context.assets.open("ca.crt")
            val ca = caInput.use { cf.generateCertificate(it) }

            val keyStoreType = KeyStore.getDefaultType()
            val keyStore = KeyStore.getInstance(keyStoreType)
            keyStore.load(null, null)
            keyStore.setCertificateEntry("ca", ca)

```

```
            val tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm()
            val tmf = TrustManagerFactory.getInstance(tmfAlgorithm)
            tmf.init(keyStore)

            val sslContext = SSLContext.getInstance("TLS")
            sslContext.init(null, tmf.trustManagers, null)

            sslContext.socketFactory
        } catch (e: Exception) {
            e.printStackTrace()
            null
        }
    }

    fun subscribe(topic: String, qos: Int, callback: (String, MqttMessage) -> Unit) {
        try {
            mqttClient.subscribe(topic, qos, null, object : IMqttActionListener {
                override fun onSuccess(asyncActionToken: IMqttToken?) {
                    Log.d("MqttManager", "Subscribed to $topic")
                }

                override fun onFailure(asyncActionToken: IMqttToken?, exception:
                 Throwable?) {
                    Log.e("MqttManager", "Failed to subscribe to $topic", exception)
                }
            })

            mqttClient.setCallback(object : MqttCallback {
                override fun connectionLost(cause: Throwable?) {
                    Log.e("MqttManager", "Connection lost", cause)
                }

                override fun messageArrived(topic: String?, message: MqttMessage?) {
                    if (topic != null && message != null) {
                        callback(topic, message)
                    }
                }

                override fun deliveryComplete(token: IMqttDeliveryToken?) {}
            })
        } catch (e: MqttException) {
            e.printStackTrace()
        }
    }

    fun unsubscribe(topic: String) {
        try {
            val token = mqttClient.unsubscribe(topic)
            token.setActionCallback(object : IMqttActionListener {
                override fun onSuccess(asyncActionToken: IMqttToken?) {
                    Log.d("MqttManager", "Unsubscribed from $topic")
                }

```

```
                override fun onFailure(asyncActionToken: IMqttToken?, exception:
                 Throwable?) {
                    Log.e("MqttManager", "Failed to unsubscribe from $topic",
                     exception)
                }
            })
            token.actionCallback = token.actionCallback
        } catch (e: MqttException) {
            Log.e("MqttManager", "Failed to unsubscribe from $topic", e)
        }
    }

    fun publish(topic: String, message: String) {
        try {
            val mqttMessage = MqttMessage(message.toByteArray())
            mqttClient.publish(topic, mqttMessage)
        } catch (e: MqttException) {
            e.printStackTrace()
        }
    }

    fun disconnect() {
        try {
            if (mqttClient.isConnected) {
                mqttClient.disconnect()
                mqttClient.unregisterResources()
                mqttClient.close()
            }
        } catch (e: MqttException) {
            e.printStackTrace()
        }
    }
}
```

Pri pokušaju konekcije sa uređajem, ako postoje problemi korisnik je prebačen na aktivnost za postavljanje WiFi postavki na uređaj kada se korisnik poveže sa pristupnom točkom uređaja. To je definirano u WebViewActivity koja je jednostavni klijent koji služi kao preglednik:

```
class WebViewActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_web_view)

        val webView: WebView = findViewById(R.id.webView)
        webView.settings.javaScriptEnabled = true

        val url = intent.getStringExtra("url")
        if (url != null) {
            webView.loadUrl(url)
        }

        webView.webViewClient = object : WebViewClient() {
            override fun onPageFinished(view: WebView?, url: String?) {
```

```
                super.onPageFinished(view, url)
                if (url?.contains("success") == true || view?.title?.contains("
                 Configuration Complete") == true) {
                    setResult(Activity.RESULT_OK)
                    finish()
                }
            }

            override fun onReceivedError(view: WebView?, errorCode: Int, description:
              String?, failingUrl: String?) {
                super.onReceivedError(view, errorCode, description, failingUrl)
                setResult(Activity.RESULT_CANCELED)
                finish()
            }
        }
    }

    override fun onBackPressed() {
        super.onBackPressed()
        setResult(Activity.RESULT_OK)
        Handler(Looper.getMainLooper()).postDelayed({
            finish()
        }, 500) // Delay to ensure the back press is properly handled
    }
}
```

Zadnja i najbitnija aktivnost jest ona u kojoj se hrana zapravo unosi u bazu nakon procesiranja. Prvo se ponovo provjerava povezanost sa MQTT. Nakon toga se aplikacija pretplaćuje na temu za težinu i šalje poruku da želi vidjeti trenutnu težinu na vagi. Vaga, pošto je već pretplaćena na tu temu, prima poruku i šalje nazad na različitoj temi trenutnu težinu. Nakon što dobije težinu, pretplaćuje se na temu od Python poslužitelja za prepoznavanje hrane i šalje poruku na drugu temu na koju je pretplaćena kamera. Kamera dobiva zahtjev, provjerava da li treba konfigurirati svjetlinu i slične karakteristike slike te šalje sliku Python poslužitelj dobiva zahtjev za predikciju i pokušava prepoznati hranu koja je na slici nakon početne obrade. Slika se nakon toga vraća aplikaciji uz prepoznatu klasu hrane. Kalkuliraju se kalorije i nutritivne vrijednosti te se to prikazuje korisniku uz sliku. Korisnik nakon toga odlučuje hoće li nastaviti dodavati stvari u jelo ili završiti ga.

Uz to, ova klasa razmišlja o dobivenoj razini sigurnosti (da li je predviđena hrana točna) te daje povratne informacije korisniku vezano uz to.

```
class CaptureFoodItemActivity : AppCompatActivity() {
    private lateinit var btnCapture: Button
    private lateinit var tvWeight: TextView
    private lateinit var tvFoodName: TextView
    private lateinit var tvCalories: TextView
    private lateinit var tvNutrition: TextView
    private lateinit var btnAddAnotherItem: Button
    private lateinit var btnFinishMeal: Button
    private lateinit var mqttManager: MqttManager
    private val mealItems = mutableListOf<MealItem>()
```

```
    private var weightReceived = false
    private var currentWeight: Float = 0.0f
    private lateinit var ivFoodImage: ImageView
    private var capturedImageData: String = ""
    private lateinit var etMealName: EditText
    private lateinit var btnConfirmItem: Button
    private var currentMealItem: MealItem? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_capture_food_item)

        initializeViews()
        setListeners()
        NutritionCalculator.initialize(applicationContext)
        mqttManager = MqttManager(this)
        connectMqtt()
    }

    private fun initializeViews() {
        btnCapture = findViewById(R.id.btnCapture)
        tvWeight = findViewById(R.id.tvWeight)
        tvFoodName = findViewById(R.id.tvFoodName)
        tvNutrition = findViewById(R.id.tvNutrition)
        tvCalories = findViewById(R.id.tvCalories)
        ivFoodImage = findViewById(R.id.ivFoodImage)
        btnAddAnotherItem = findViewById(R.id.btnAddAnotherItem)
        btnFinishMeal = findViewById(R.id.btnFinishMeal)
        etMealName = findViewById(R.id.etMealName)
        btnConfirmItem = findViewById(R.id.btnConfirmItem)
    }

    private fun setListeners() {
        btnCapture.setOnClickListener { captureFood() }
        btnAddAnotherItem.setOnClickListener { addAnotherItem() }
        btnFinishMeal.setOnClickListener { finishMeal() }
        btnConfirmItem.setOnClickListener { confirmItem() }
    }

    private fun connectMqtt() {
        lifecycleScope.launch {
            val connected = mqttManager.connect()
            if (connected) {
                Toast.makeText(this@CaptureFoodItemActivity, "Connected to MQTT
                 broker", Toast.LENGTH_SHORT).show()
            } else {
                Toast.makeText(this@CaptureFoodItemActivity, "Failed to connect to
                 MQTT broker", Toast.LENGTH_SHORT).show()
            }
        }
    }


```

```
62      private fun captureFood() {
63          weightReceived = false
64          currentWeight = 0.0f
65          currentMealItem = null
66          Log.d("CaptureFoodItemActivity", "Sending weight request")
67          mqttManager.subscribe("esp32/weight", 0) { _, message ->
68              handleWeightMessage(message.toString())
69          }
70          mqttManager.publish("esp32/weight_request", "request")
71
72          lifecycleScope.launch {
73              for (i in 1..10) { // Try for 10 seconds
74                  delay(1000)
75                  if (weightReceived) {
76                      Log.d("CaptureFoodItemActivity", "Weight received: $currentWeight
                         ")
77                      break
78                  }
79              }
80              if (!weightReceived) {
81                  Log.d("CaptureFoodItemActivity", "Weight not received after 10
                     seconds, proceeding with capture anyway")
82              }
83              Log.d("CaptureFoodItemActivity", "Sending capture request")
84              mqttManager.subscribe("model/prediction", 0) { _, message ->
85                  handlePredictionMessage(message.toString())
86              }
87              mqttManager.publish("esp32cam/capture", "capture")
88          }
89      }
90
91      private fun handleWeightMessage(weightStr: String?) {
92              if (weightStr != null) {
93                  currentWeight = weightStr.toFloat()
94              };
95              weightReceived = true
96      }
97
98      private fun handlePredictionMessage(predictionStr: String) {
99          runOnUiThread {
100             try {
101                 val json = JSONObject(predictionStr)
102                 val predictedClass = json.getString("predicted_class")
103                 val confidence = json.getDouble("confidence")
104                 capturedImageData = json.getString("image_data")
105
106                 tvFoodName.text = "Food: $predictedClass"
107                 tvWeight.text = "Weight: ${currentWeight}g"
108
109                 val (calories, nutritiveValues) = NutritionCalculator.
                     calculateNutritiveValues(predictedClass, currentWeight)
110
111                 val imageBytes = Base64.decode(capturedImageData, Base64.DEFAULT)
```

```
                Glide.with(this).load(imageBytes).apply(RequestOptions().
                 diskCacheStrategy(
                    DiskCacheStrategy.NONE)).into(ivFoodImage)


                val mealItem = MealItem(
                    name = predictedClass,
                    weight = currentWeight,
                    calories = calories,
                    nutritiveValues = nutritiveValues,
                    imageData = capturedImageData
                )

                mealItems.add(mealItem)
                updateNutritionInfo(nutritiveValues, calories)
                handleConfidence(confidence)

            } catch (e: Exception) {
                Log.e("CaptureFoodItemActivity", "Error processing prediction", e)
            }
        }
    }

    private fun handleConfidence(confidence: Double) {
        when {
            confidence < 0.5 -> {
                Toast.makeText(this, "The device probably cannot recognize this food.
                  Please try again.", Toast.LENGTH_LONG).show()
                btnConfirmItem.isEnabled = false
            }
            confidence < 0.75 -> {
                Toast.makeText(this, "Please shift the food around for a clearer view
                  and try again.", Toast.LENGTH_LONG).show()
                btnConfirmItem.isEnabled = true
            }
            else -> {
                Toast.makeText(this, "Food recognized with high confidence. Is this
                 correct?", Toast.LENGTH_LONG).show()
                btnConfirmItem.isEnabled = true
            }
        }
    }

    private fun confirmItem() {
        AlertDialog.Builder(this)
            .setTitle("Confirm Food Item")
            .setMessage("Is the predicted food item correct?")
            .setPositiveButton("Yes") { _, _ ->
                currentMealItem?.let { mealItems.add(it) }
                updateUI()
                resetCapture()
            }
            .setNegativeButton("No") { _, _ ->
```

```
                Toast.makeText(this, "Please capture the food item again", Toast.
                 LENGTH_SHORT).show()
                resetCapture()
            }
            .show()
    }

    private fun resetCapture() {
        currentMealItem = null
        tvWeight.text = ""
        tvFoodName.text = ""
        tvCalories.text = ""
        tvNutrition.text = ""
        ivFoodImage.setImageDrawable(null)
        btnConfirmItem.isEnabled = false
    }


    private fun finishMeal() {
        FirebaseManager.getCurrentUser { user ->
            if (user == null) {
                Toast.makeText(this, "Error: User not logged in", Toast.LENGTH_SHORT)
                 .show()
                return@getCurrentUser
            }
            val userInputMealName = etMealName.text.toString().trim()
            val mealName = if (userInputMealName.isNotEmpty()) {
                "$userInputMealName - ${getCurrentDateTime()}"
            } else {
                "Meal ${getCurrentDateTime()}"
            }

            val totalCalories = mealItems.sumOf { it.calories.toDouble() }.toFloat()
            val totalNutritiveValues = calculateTotalNutritiveValues()

            val meal = Meal(
                userId = user.uid,
                name = mealName,
                calories = totalCalories,
                items = mealItems.toList(),
                totalNutritiveValues = totalNutritiveValues
            )

            FirebaseManager.addMeal(meal) { success, message ->
                runOnUiThread {
                    if (success) {
                        Toast.makeText(this, "Meal added successfully", Toast.
                         LENGTH_SHORT).show()
                        finish()
                    } else {
                        Toast.makeText(this, "Failed to add meal: $message", Toast.
                         LENGTH_SHORT).show()
                    }
```

```
                }
            }
        }
    }
    private fun getCurrentDateTime(): String {
        val sdf = SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.getDefault())
        return sdf.format(Date())
    }

    private fun updateNutritionInfo(nutritiveValues: NutritiveValues, calories: Float
     ) {
        tvNutrition.text = """
            Calories: ${String.format("%.2f", calories)}
            Protein: ${String.format("%.2f", nutritiveValues.protein)}g
            Calcium: ${String.format("%.2f", nutritiveValues.calcium)}g
            Fat: ${String.format("%.2f", nutritiveValues.fat)}g
            Carbohydrates: ${String.format("%.2f", nutritiveValues.carbohydrates)}g
            Vitamins: ${String.format("%.2f", nutritiveValues.vitamins)}g
        """.trimIndent()
    }
    private fun calculateTotalNutritiveValues(): NutritiveValues {
        return mealItems.fold(NutritiveValues()) { acc, item ->
            NutritiveValues(
                protein = acc.protein + item.nutritiveValues.protein,
                calcium = acc.calcium + item.nutritiveValues.calcium,
                fat = acc.fat + item.nutritiveValues.fat,
                carbohydrates = acc.carbohydrates + item.nutritiveValues.
                 carbohydrates,
                vitamins = acc.vitamins + item.nutritiveValues.vitamins
            )
        }
    }

    private fun addAnotherItem() {
        resetCapture()
    }

    private fun updateUI() {
        btnAddAnotherItem.isEnabled = mealItems.isNotEmpty()
        btnFinishMeal.isEnabled = mealItems.isNotEmpty()
        btnConfirmItem.isEnabled = currentMealItem != null
    }

    override fun onDestroy() {
        super.onDestroy()
        mqttManager.disconnect()
    }
}
```

3.2.5. MQTT broker

MQTT broker korišten u ovom slučaju je Serverless EMQX. EMQX je visoko skalabilni open-source MQTT broker koji podržava serverless arhitekturu. Posebno je pogodan za IoT aplikacije, povezane uređaje i scenarije gdje je potrebna obrada velikog broja poruka u realnom vremenu. Njegova serverless priroda omogućava korisnicima da se fokusiraju na razvoj aplikacija bez brige o infrastrukturi. Ključne karakteristike su: [30]

- Skalabilnost: Može podržati milijune istovremenih MQTT konekcija i obrađivati velike količine poruka u realnom vremenu.
- Bezserverska arhitektura: Ne zahtijeva upravljanje serverima, već se automatski skalira prema potrebama.
- Visoka dostupnost: Dizajniran za distribuirano okruženje, pružajući visoku dostupnost i otpornost na greške.
- Cloud-native: Može se lako implementirati u cloud okruženjima poput Kubernetesa.
- Podržava MQTT 5.0: Kompatibilan je s najnovijim MQTT protokolom, uz podršku za starije verzije.
- Proširivost: Nudi brojne dodatke i integracije s drugim sistemima i protokolima.
- Siurnost: Uključuje napredne sigurnosne funkcije poput TLS/SSL enkripcije i autentifikacije.
- Performanse: Optimiziran za brzu obradu poruka i nisku latenciju.
- Upravljanje i monitoring: Pruža detaljne metrike i alate za praćenje performansi.
- Fleksibilnost implementacije: Može se koristiti on-premise, u cloudu ili u hibridnim okruženjima.

Obično dolazi s robusnom kontrolnom pločom koja omogućava administratorima i developerima da nadziru, upravljaju i optimiziraju svoj MQTT broker. Ključni elementi koji su na toj ploči jesu:

Metrike i statistike:

- Broj aktivnih konekcija
- Stopa protoka poruka (poruke po sekundi)
- Latencija
- Korištenje resursa (CPU, memorija, mreža)

Upravljanje klijentima:

- Pregled aktivnih klijenata
- Detalji o pojedinačnim klijentima (ID, IP adresa, vrijeme konekcije)
- Mogućnost isključivanja problematičnih klijenata

Teme i pretplate:

- Pregled aktivnih tema
- Statistike o broju pretplatnika po temi
- Mogućnost filtriranja i pretraživanja tema

Sigurnost i autentifikacija:

- Upravljanje SSL/TLS certifikatima
- Konfiguracija autentifikacijskih metoda
- Pregled i upravljanje API ključevima

Pravila i rutiranje:

- Konfiguracija pravila za obradu poruka
- Definiranje akcija za specifične događaje
- Integracija s vanjskim servisima

Skaliranje i performanse:

- Automatsko ili ručno skaliranje resursa
- Podešavanje postavki za optimizaciju performansi

Logovi i dijagnostika:

- Pregled sistemskih logova
- Alati za dijagnostiku problema
- Postavke za debugging

Integracije:

- Konfiguracija integracija s drugim servisima (npr. baze podataka, analitički alati)
- Upravljanje dodatcima i proširenjima

Alerting:

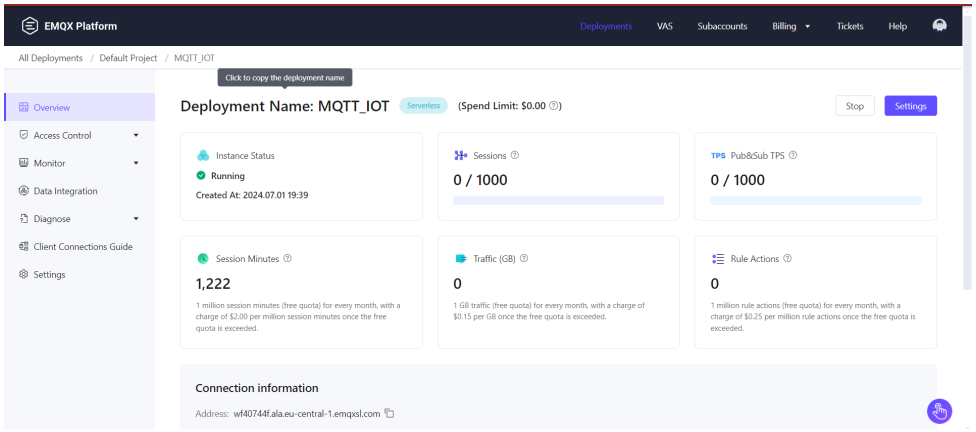
- Postavljanje pragova za alarme
- Konfiguracija notifikacija (email, SMS, integrirani sistemi)

Vizualizacije:

- Grafovi i dijagrami koji prikazuju ključne metrike tijekom vremena
- Toplinske mape aktivnosti klijenata ili tema

Upravljanje korisnicima kontrolne ploče:

- Dodavanje i uklanjanje administratora
- Postavljanje različitih nivoa pristupa

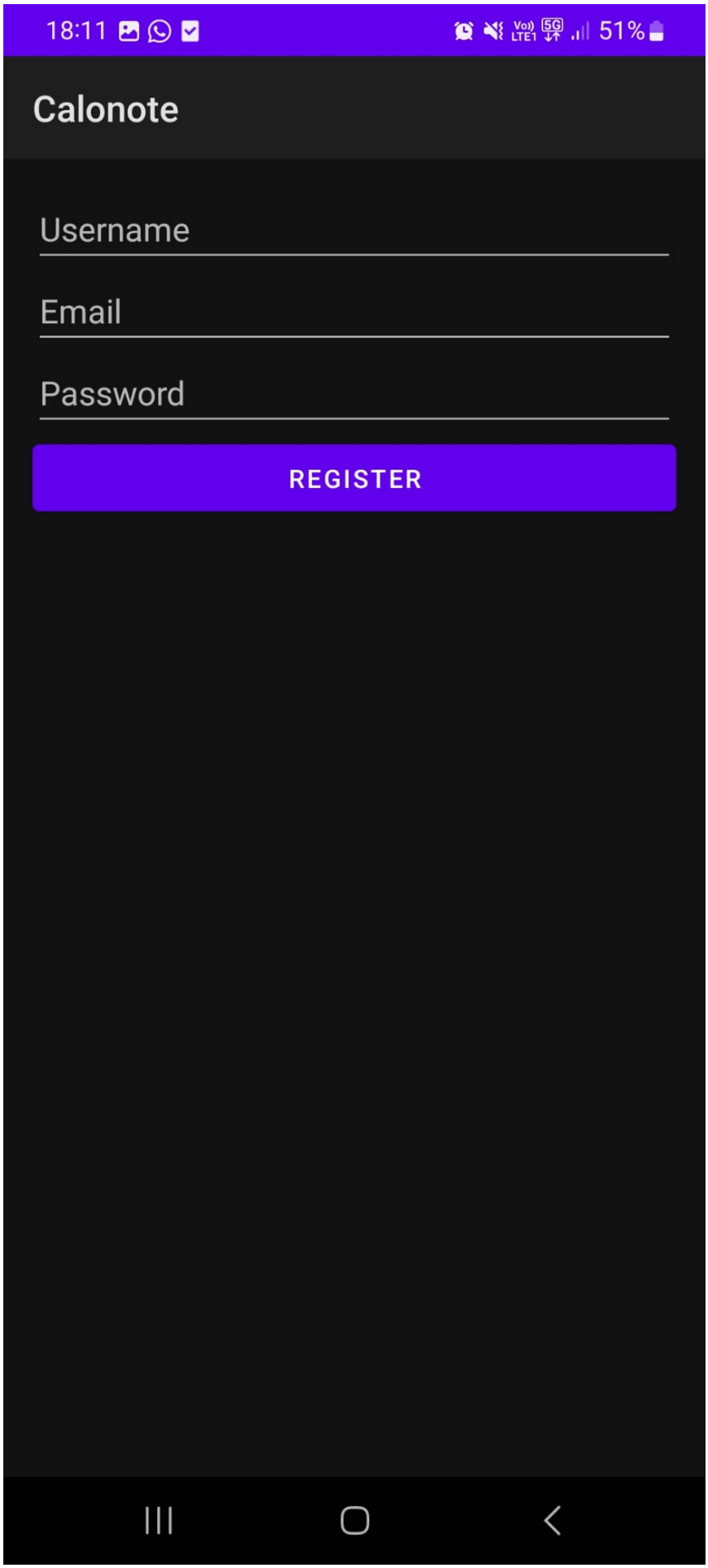


Slika 11: EMQX administratorska ploča

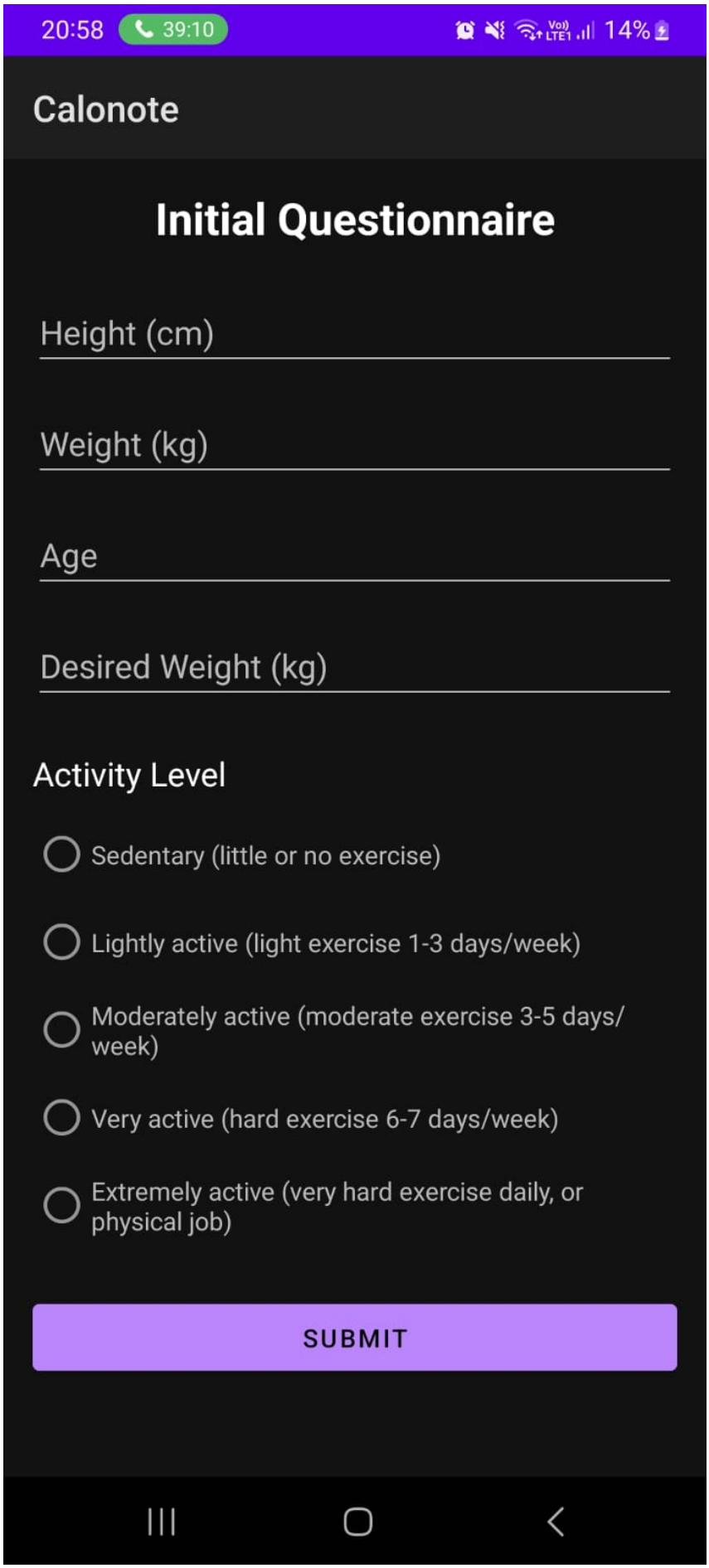
3.3. Prikaz rada sustava

Kao prvo, korisnik ulazi u aktivnost za prijavljivanje u sustav, što je prikazano na slici 12. Ako korisnik prvi puta ulazi u sustav, može se registrirati kao što je i prikazano na slici 13. Slika 14 također prikazuje inicijalni upitnik kojeg se pita svakog korisnika tijekom prvog ulaska u aplikaciju. Nakon toga na slici 15 može se vidjeti početna stranica aplikacije sa kružnim trakama napretka koje prate kalorije i ostale nutritivne vrijednosti unesene tokom dana. Maksimalni željeni unos svake vrijednosti je određen prema vrijednostima koje je naveo korisnik pri inicijalnom upitniku. Nadalje, na slici 16 može se vidjeti profil za mijenjanje tih vrijednosti, a na slici 17 primjer povijesti unosa hrane. Klikom na „Add Meal“, korisnik dolazi u aktivnost gdje se aplikacija pokušava spojiti sa uređajem (slika 18). Nakon spajanja ili potrebne konfiguracije, korisnik može ili kalibrirati vagu ili otići na daljnju aktivnost a to je zapravo prepoznavanje i dodavanje hrane (slika 20). Na slikama 19 i 20 možemo također vidjeti konkretni primjer prepoznavanja pizze (prije toga je kalibrirana vaga sa kartonskom kutijom od pizze). Na kraju svega pri povratku u početnu aktivnost korisnik može vidjeti da su danas unešene neke vrijednosti i koliko odstupaju od željenih dnevnih vrijednosti (slika 21).

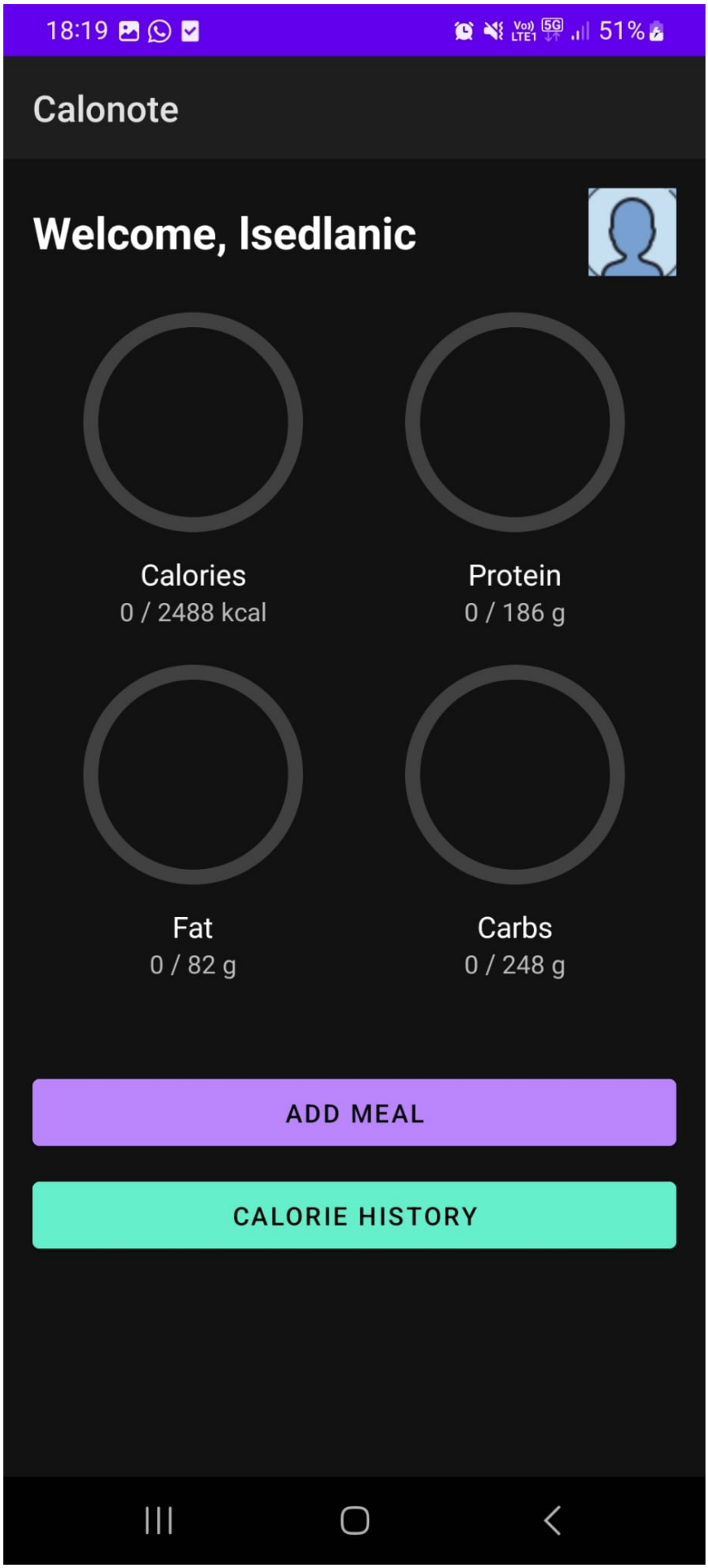
Slika 12: Login



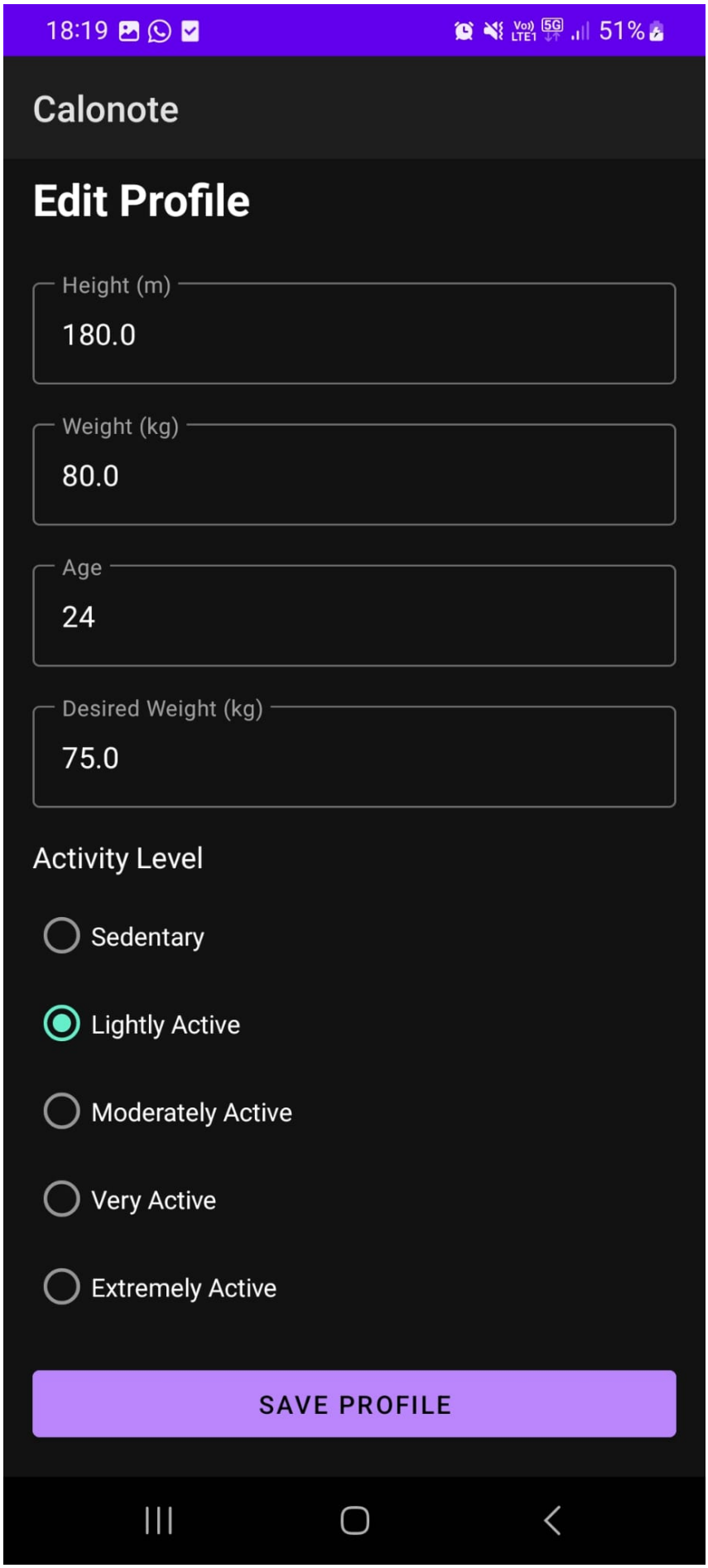
Slika 13: Registracija



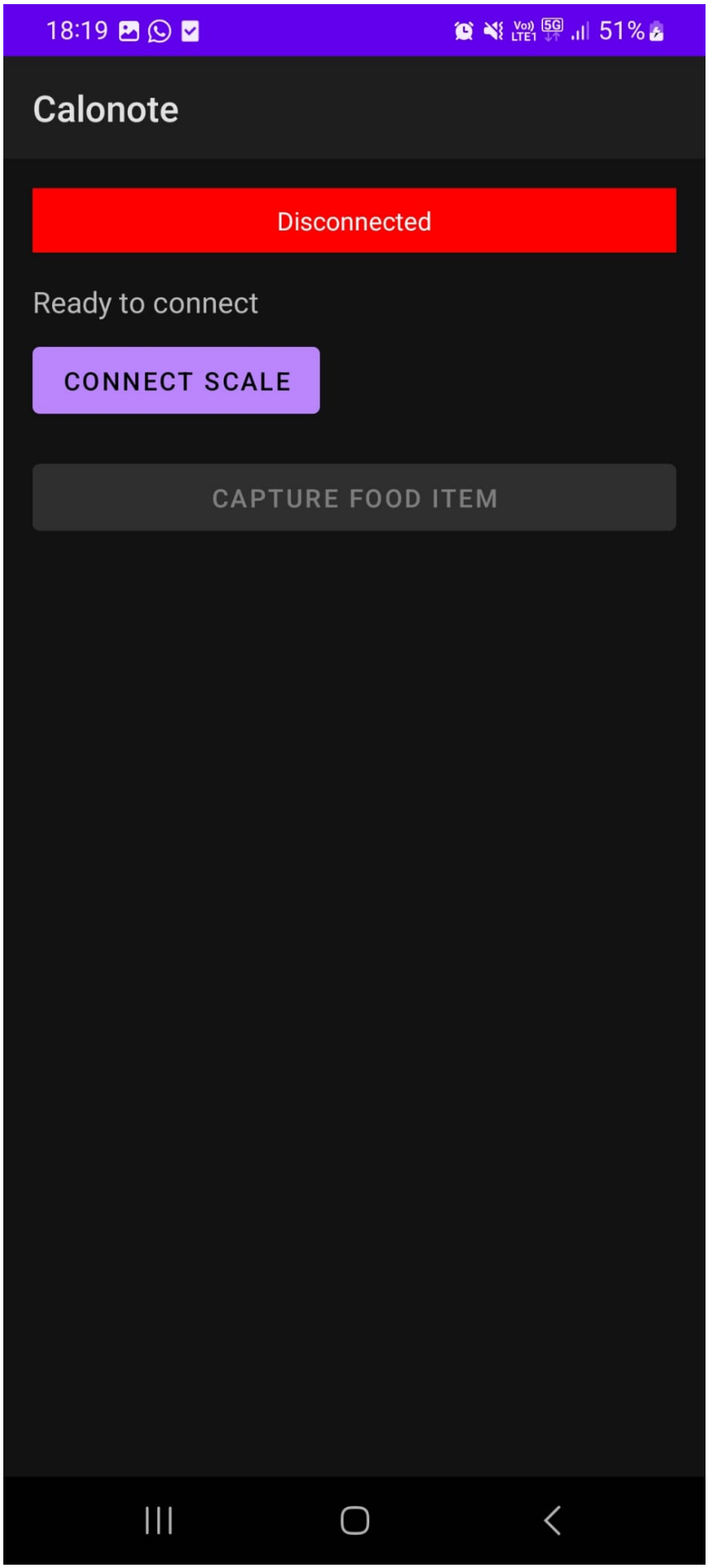
Slika 14: Inicijalni upitnik



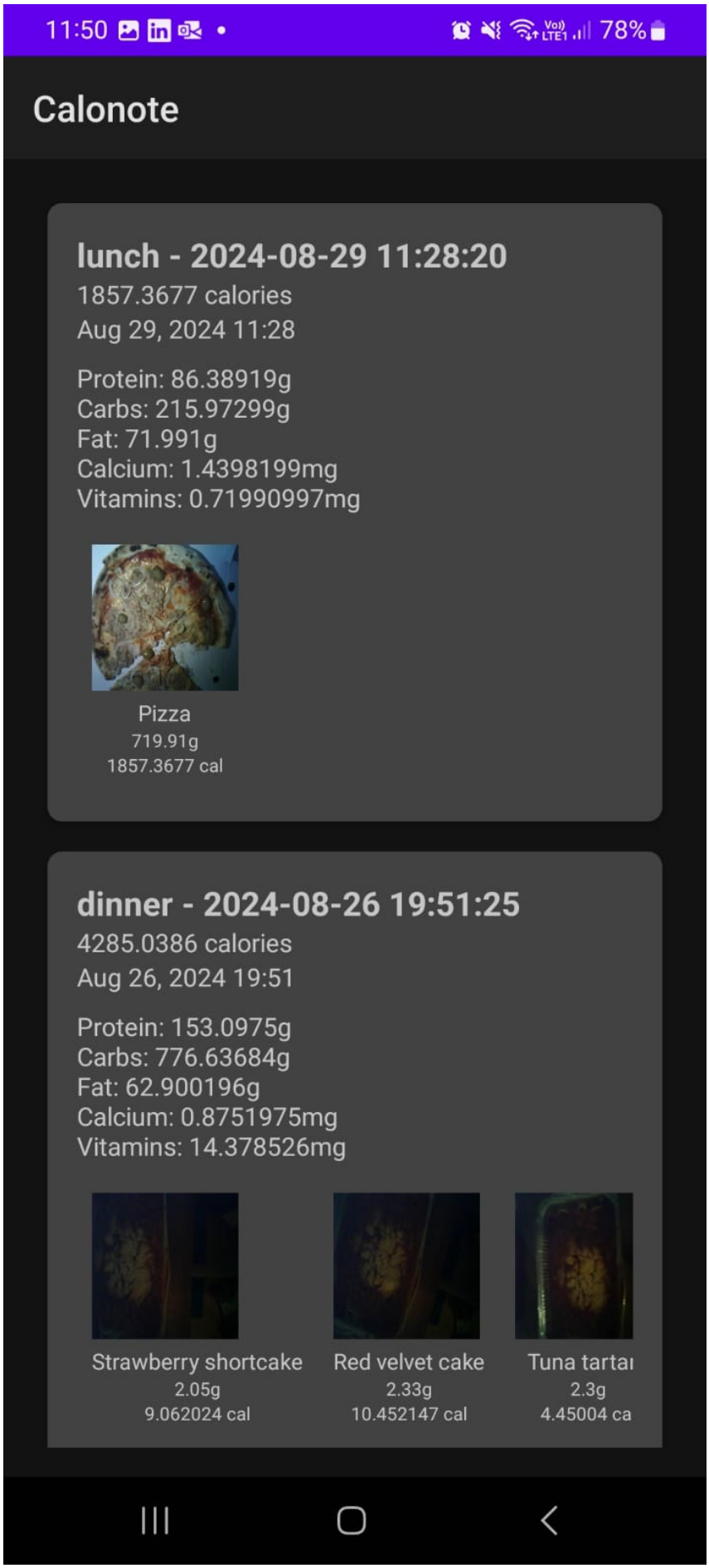
Slika 15: Početna stranica



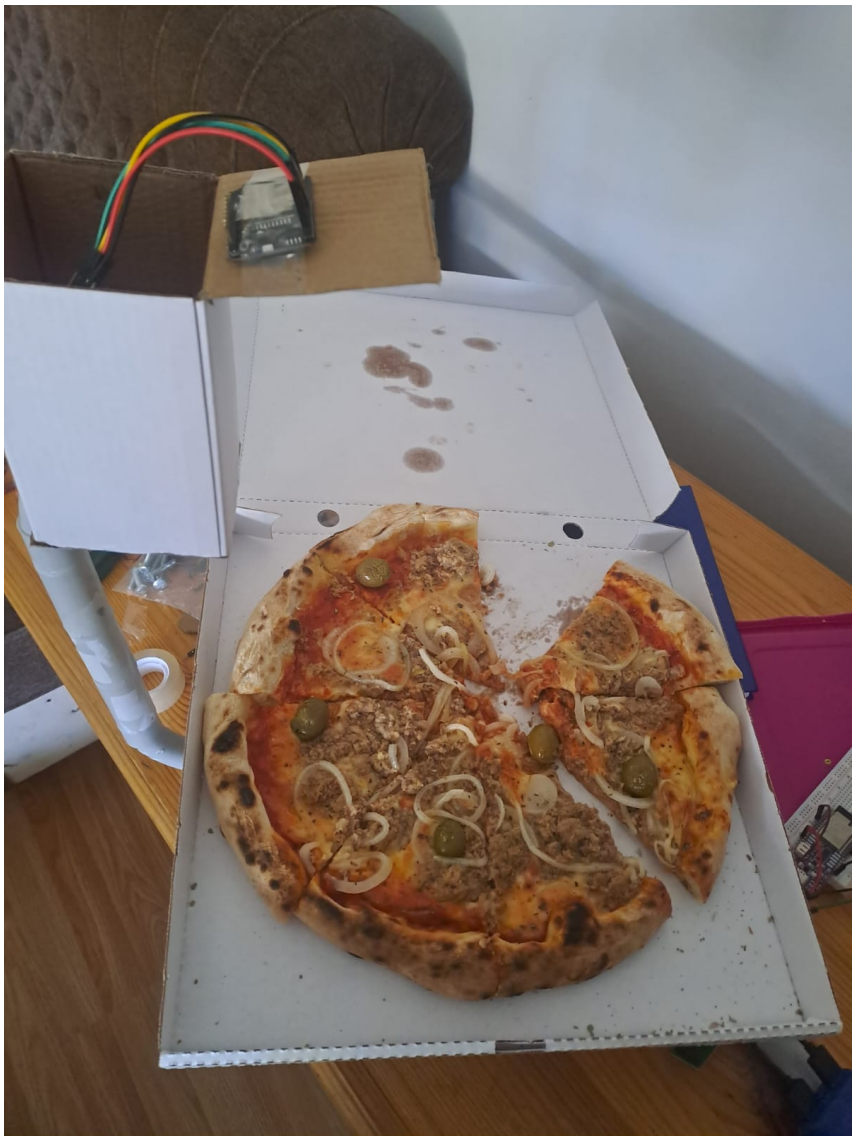
Slika 16: Profil



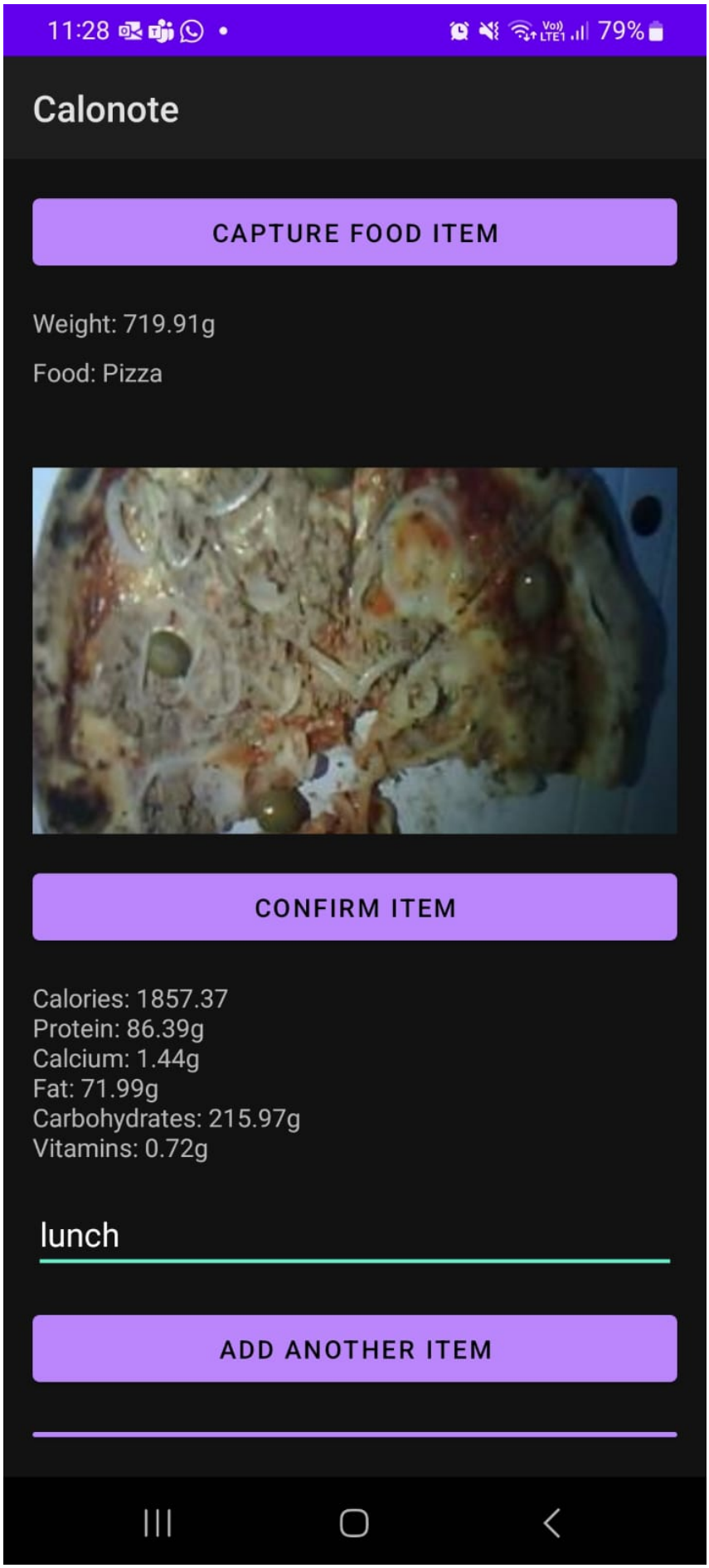
Slika 17: Spajanje s uređajem



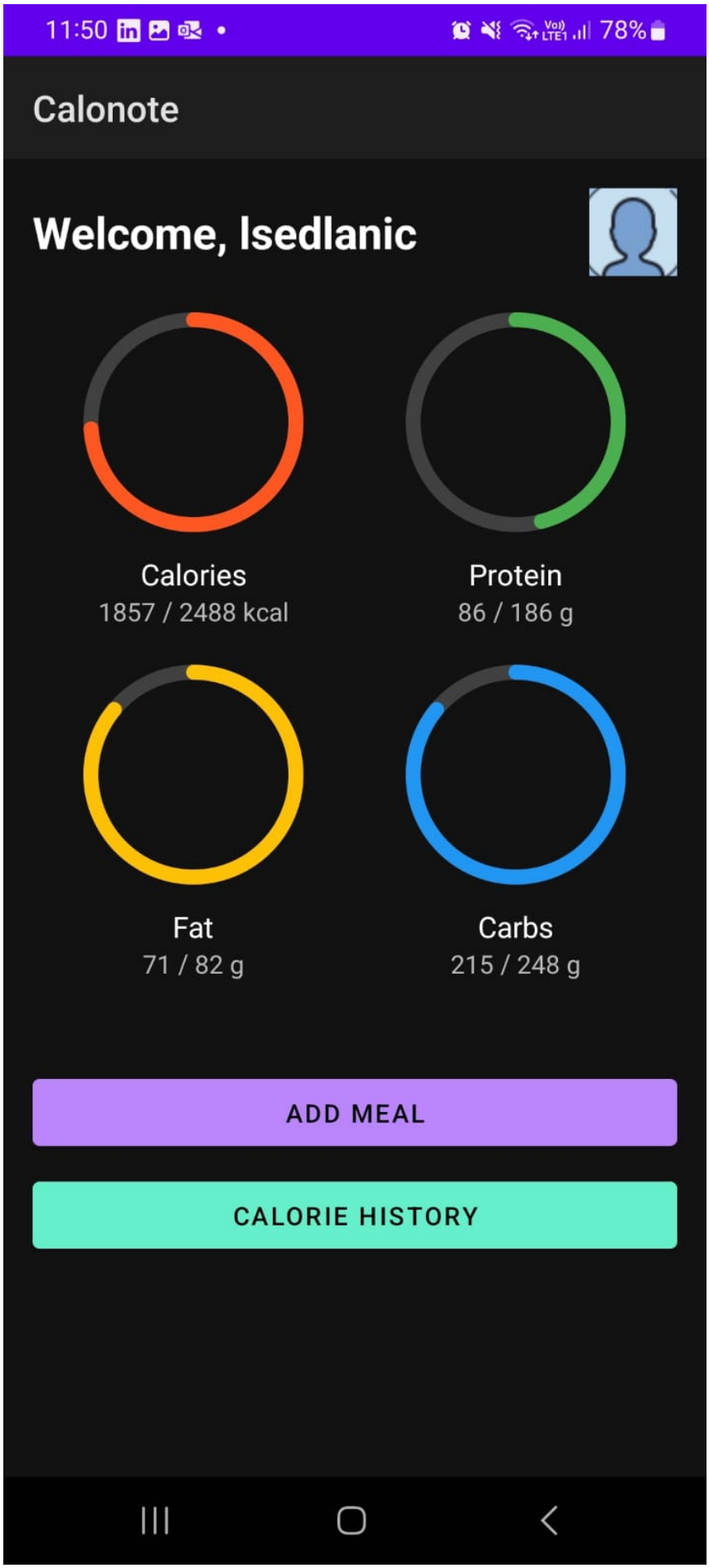
Slika 18: Povijest unosa hrane



Slika 19: Prikaz hrane (pizza) na uređaju



Slika 20: Prikaz hrane (pizza) unutar aplikacije



Slika 21: Ažurirane nutritivne vrijednosti nakon unosa hrane

4. Zaključak

Razvoj pametnog uređaja za praćenje kalorija predstavlja značajan iskorak u spajanju tehnologije i zdravlja, pružajući korisnicima moćan alat za praćenje i upravljanje prehrambenim navikama. Kroz ovaj diplomski rad prikazano je kako integracija IoT tehnologija, senzora i algoritama strojnog učenja može rezultirati sustavom koji automatski prepoznaje hranu i izračunava njezinu nutritivnu vrijednost, omogućujući korisnicima precizno praćenje unosa kalorija.

Rezultati istraživanja analiziranih u ovome radu pokazuju da ovakav uređaj napravljen u sklopu rada može značajno olakšati održavanje uravnotežene prehrane i poboljšanje općeg zdravlja. Korištenjem naprednih senzora i tehnologija strojnog učenja, uspješno je stvoren sustav koji je intuitivan za korištenje i koji pruža točne podatke o prehrani korisnika.

Budući rad na ovom području može se usmjeriti na daljnju optimizaciju algoritama za prepoznavanje hrane, povećanje preciznosti senzora te proširenje funkcionalnosti uređaja kako bi pokrivao i druge aspekte zdravlja, poput praćenja unosa tekućine, fizičke aktivnosti i spavanja. Skup podataka koji je korišten se isto tako može proširiti sa više klasa hrane. Također, integracija ovakvih sustava s drugim zdravstvenim aplikacijama i platformama može dodatno unaprijediti korisničko iskustvo i pružiti sveobuhvatniji pristup upravljanju zdravljem.

Zaključno, razvoj pametnog uređaja za praćenje kalorija otvara nove mogućnosti u području personalizirane zdravstvene skrbi, omogućujući korisnicima aktivnu ulogu u održavanju vlastitog zdravlja putem jednostavnih i učinkovitih tehnoloških rješenja. Kroz daljnja istraživanja i unapređenja, ovakvi sustavi mogu postati ključni alati u prevenciji bolesti i promicanju zdravog načina života.

4.1. Popis Literature

1. Amazon (2024.). What is the Internet of Things (IoT)?. Preuzeto 29.7.2024. s https://aws.amazon.com/what-is/iot/

2. IBM (2024.). What is artificial intelligence (AI)?. Preuzeto 29.7.2024. s https://www.ibm.com/topics/artificial-intelligence

3. Yashodha G., Pameela R., Lavanya A. i Sathyavathy V. (2021.). Role of Artificial Intelligence in the Internet of Things – A Review. Preuzeto 25.7.2024. s https://iopscience.iop.org/article/10.1088/1757-899X/1055/1/012090/pdf

4. Ghosh A., Chakraborty D., Law A., (2014.). Artificial intelligence in Internet of things. Preuzeto 25.7.2024. s https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/trit.2018.1008

5. Dong B, Shi, Q., Yang Y., Weng F., Zhang Z., Lee C. Technology evolution from self-powered sensors to AIoT enabled smart homes. Preuzeto 25.7.2024. s https://www.ece.nus.edu.sg/stfpage/elelc/Publication/2021/NanoEnergy_21V79_105414_Technology%20evolution%20from%20self-powered%20sensors%20to%20AIoT%20enabled%20smart%20homes_Rev_BoweiDong.pdf

6. Hou K. M., Diao X., Shi H., Ding H., Zhou H., i Vaulx C. Trends and Challenges in AIoT/IIoT/IoT Implementation. Preuzeto 25.7.2024. s https://www.mdpi.com/1424-8220/23/11/5074

7. Madakam S., Uchiya T., Mark S., i Lurie Y. Artificial Intelligence, Machine Learning and Deep Learning. Preuzeto 25.7.2024. s https://journals.sagepub.com/doi/epub/10.1177/2319510X221136682

8. Giuseppe Carleo et al. Machine learning and the physical sciences. Preuzeto 25.7.2024. s https://link.aps.org/accepted/10.1103/RevModPhys.91.045002

9. Javatpoint (2024.). Applications of Machine learning. Preuzeto 30.7.2024. s https://www.javatpoint.com/applications-of-machine-learning

10. Adi E., Anwar A., Baig Z., Zeadally S. (2020.) Artificial Intelligence, Machine Learning and Deep Learning. Preuzeto 25.7.2024. s https://arxiv.org/pdf/2007.04093

11. Sundaravadivel P, Kesavan K, Kesavan L, Mohanty SP, Kougianos E (2018). A deep-learning based automated nutrition monitoring system in the iot. (IEEE Transactions on Consumer Electronics 64(3):390–398, DOI 10.1109/TCE.2018.2867802)

12. Iqbal H. Sarke (2021.) Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. Preuzeto 25.7.2024. s https://link.springer.com/article/10.1007/s42979-021-00815-1

13. Truong Thanh Tai, Dang Ngoc Hoang Thanh i Nguyen Quoc Hung (2021.) A Dish Recognition Framework Using Transfer Learning. Preuzeto 25.7.2024. s https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9681805

14. Gupta S. (2020.) Inception V3 Model. Preuzeto 26.7.2024. s https://www.kaggle.com/datasets/imsparsh/inception-v3-model

15. Vishwanath C. Burkapalli i Priyadarshini C. Patil (2020.) TRANSFER LEARNING: INCEPTION-V3 BASED CUSTOM CLASSIFICATION APPROACH FOR FOOD IMAGES. Preuzeto 25.7.2024. s https://ictactjournals.in/paper/IJIVP_Vol_10_Iss_4_Paper_6_2261_2267.pdf

16. Julian V. i Botti V. (2019.) Multi-Agent Systems. Preuzeto 25.7.2024. s https://www.mdpi.com/2076-3417/9/7/1402

17. Munindar P. Singh i Amit K. Chopra (2017.) The Internet of Things and Multiagent Systems: Decentralized Intelligence in Distributed Computing. Preuzeto 25.7.2024. s https://ieeexplore.ieee.org/document/7980111

18. Jung T., Shah P. i Weyrich M (2018.) Dynamic Co-Simulation of Internet-of-Things-Components using a Multi-Agent-System. Preuzeto 25.7.2024. s https://www.sciencedirect.com/science/article/pii/S2212827118301884

19. Agostino Forestiero (2017.) Multi-Agent Recommendation System in Internet of Things. Preuzeto 25.7.2024. s https://ieeexplore.ieee.org/abstract/document/7973778

20. Syed Umar Amin i M. Shamim Hossain (2020.) Edge Intelligence and Internet of Things in Healthcare: A Survey. Preuzeto 25.7.2024. s https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9294145

21. Buyya R. i Dastjerdi A. (2016.) Internet of Things: Principles and Paradigms. Preuzeto 25.7.2024. s https://dhoto.lecturer.pens.ac.id/lecture_notes/internet_of_things/Internet%20of%20Things%20Principles%20and%20Paradigms.pdf

22. S. Hrushikesava Raju et al. (2022.) An IoT Vision for Dietary Monitoring System and for Health Recommendations. Preuzeto 25.7.2024. s https://link.springer.com/chapter/10.1007/978-981-16-5529-6_65

23. Tobias Goebel (2023.) What is MQTT?. Preuzeto 25.7.2024. s https://www.twilio.com/en-us/blog/what-is-mqtt

24. Chiradeep BasuMallick (2022.) What Is MQTT (MQ Telemetry Transport)? Working, Types, Importance, and Applications. Preuzeto 25.7.2024. s https://www.spiceworks.com/tech/iot/articles/what-is-mqtt/

25. Soni D. i Makwana A. (2017.) A SURVEY ON MQTT: A PROTOCOL OF INTERNET OF THINGS(IOT). Preuzeto 25.7.2024. s https://www.researchgate.net/

publication/316018571_A_SURVEY_ON_MQTT_A_PROTOCOL_OF_INTERNET_OF_THINGSIOT

26. randomnerdtutorials (2024.) Arduino with Load Cell and HX711 Amplifier (Digital Scale). Preuzeto 25.7.2024. s https://randomnerdtutorials.com/arduino-load-cell
27. Flintec (2024.) What is a load cell and how does it work?. Preuzeto 25.7.2024. s https://www.flintec.com/weight-sensors/load-cells/what-is-a-load-cell
28. randomnerdtutorials (2024.) How to Program / Upload Code to ESP32-CAM AI-Thinker (Arduino IDE). Preuzeto 25.7.2024. s https://randomnerdtutorials.com/program-upload-code-esp32-cam/
29. Bossard L., Guillaumin M. i Van Gool L. (2014.) Food-101 – Mining Discriminative Components with Random Forests. Preuzeto 26.7.2024. s https://data.vision.ee.ethz.ch/cvl/datasets_extra/food-101/
30. EMQX (2024.) Serverless MQTT Broker. Preuzeto 25.7.2024. s https://www.emqx.com/en/cloud/serverless-mqtt

Popis slika

Popis tablica