SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1111

Ugradbeni računalni sustav za upravljanje uređajima na mreži

Karlo Vučilovski

Zagreb, srpanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1111

Ugradbeni računalni sustav za upravljanje uređajima na mreži

Karlo Vučilovski

Zagreb, srpanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 3. ožujka 2025.

DIPLOMSKI ZADATAK br. 1111

Pristupnik: **Karlo Vučilovski (0036524825)**
Studij: Računarstvo
Profil: Računalno inženjerstvo
Mentor: prof. dr. sc. Hrvoje Mlinarić

Zadatak: **Ugradbeni računalni sustav za upravljanje uređajima na mreži**

Opis zadatka:

Projektirati ugradbeni računalni sustav interneta stvari (eng. Ineternet of Things, IoT) zasnovan na procesoru ESP32 koji će omogućiti upravljanje uređajima koji imaju neki oblik upravljačkog sučelja zasnovanog na mrežnoj komunikaciji. Računalni sustav mora se sastojati od procesora ESP32, rotacijskih enkodera I prikaznika. S pomoću rotacijskih enkodera i prikaznika ostvariti sustav izbornika putem kojeg će korisnik moći odabrati pojedinu operaciju i izvršiti upravljanje uređajem putem mrežne komunikacije. Osim sustava izbornika potrebno je ostvariti i osnovne grafičke komponente kao što su klizač i gumb. Cijeli sustav izbornika mora se moći jednostavno konfigurirati korištenjem konfiguracijske datoteke.

Rok za predaju rada: 4. srpnja 2025.

Sadržaj

1. Uvod....7

2. Opis razvijenog proizvoda....8

3. Tehničke značajke....9
- 3.1 O ESP32....10
- 3.2 Vanjske biblioteke....12
  - 3.2.1 LVGL....12
  - 3.2.2 esp_lcd_ili9341....13
  - 3.2.3 espressif/button....13
  - 3.2.4 esp_littlefs....13
  - 3.2.5 cJSON....14
- 3.3 Spremljena konfiguracija u memoriji....14
  - 3.3.1 Spremanje konfiguracije....14
  - 3.3.2 Particije....15
- 3.4 Registracija na poslužitelj i čuvanje podataka....16
- 3.5 Senzori, aktuatori, poruke, prikaz (obrada) odgovora....16
- 3.6 Proširenja grafičkog sučelja....16
- 3.7 Rad sa zahtjevima....17
- 3.8 Vanjsko sklopovlje i upravljački programi....22
  - 3.8.1 Ekran ST7789....22
  - 3.8.2 Grafička biblioteka LVGL....22
  - 3.8.3 Brojač impulsa....24
  - 3.8.4 Datotečni sustav LittleFS....25
  - 3.8.5 Programska biblioteka cJSON....25
- 3.9 Obrada HTTP odgovora....26
- 3.10 Aplikacijski poslužitelj....28
- 3.11 Opis rada....30
- 3.12 Ograničenja....32
- 3.13 Prednosti....32
- 3.14 Opis programskog koda....33
  - 3.14.1 REST API poslužitelj....33
  - 3.14.2 Krajnji čvor....38
- 3.15 Zakjučak....50

4. Literatura....51

5. Naslov....53

6. Sažetak....53

7. Ključne riječi....53

8. Prilozi.. 54

1. Uvod

Upravljanje raspodijeljenim sustavima putem mreže se može brzo zakomplicirati zbog raširenosti dostupnih tehnologija. Razviti mobilnu aplikaciju je relativno brzo i uobičajeno, ali uređaji na kojima ta aplikacija radi nisu jednostavni niti cjenovno efikasni. Također troše mnogo električne energije za jednostavne zadatke zbog svojih dodatnih funkcija. S druge strane, postoje jednostavni uređaji s mnogo ugrađenih funkcija, ali su također skupi jer je skup njihov razvoj. Za operacije koje se izvode povremeno je praktičnije koristiti jedan uređaj koji služi samo za to i ima dobro definirana sučelja i uvijek je dostupan. Bitno je da uređaj može razmjenjivati podatke koristeći što manje resursa. Uređaj treba imati mogućnost podešavanja pomoću konfiguracijske datoteke da se izbjegnu nepotrebne i učestale promjene programskog koda.

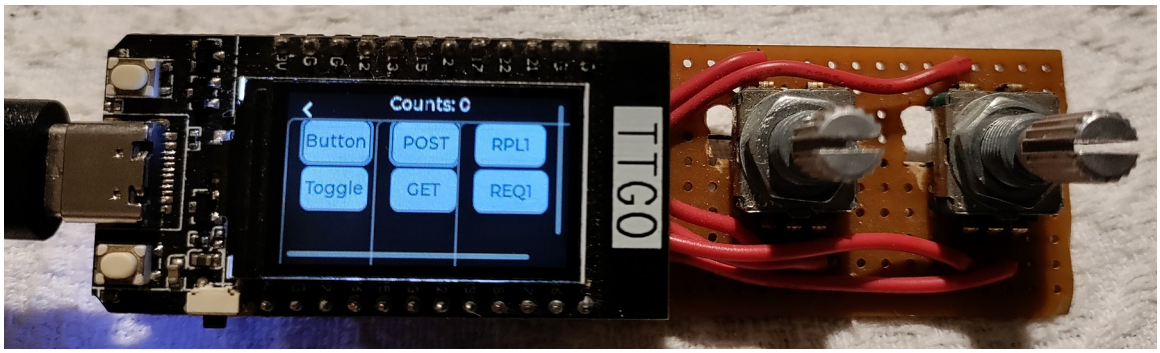
2. Opis razvijenog proizvoda

Razvijeni proizvod se sastoji od 2 osnovna dijela: REST API poslužitelja i klijenta pomoću kojeg korisnik vrši interakciju sa sustavom. REST API poslužitelj služi kao dio sustava koji sinkronizira rad cijelog sustava. On prima zahtjeve s krajnjeg čvora, obrađuje ih te ih prosljeđuje čvorovima kojima su potrebni.

Krajnji uređaj je mikrokontroler ESP32 na koji je spojen mali ekran i 2 enkodera koji služe kao sučelje prema korisniku. On od korisnika prima podatke i zahtjeve te ih prosljeđuje na obradu.

Također može s poslužitelja primiti podatke.

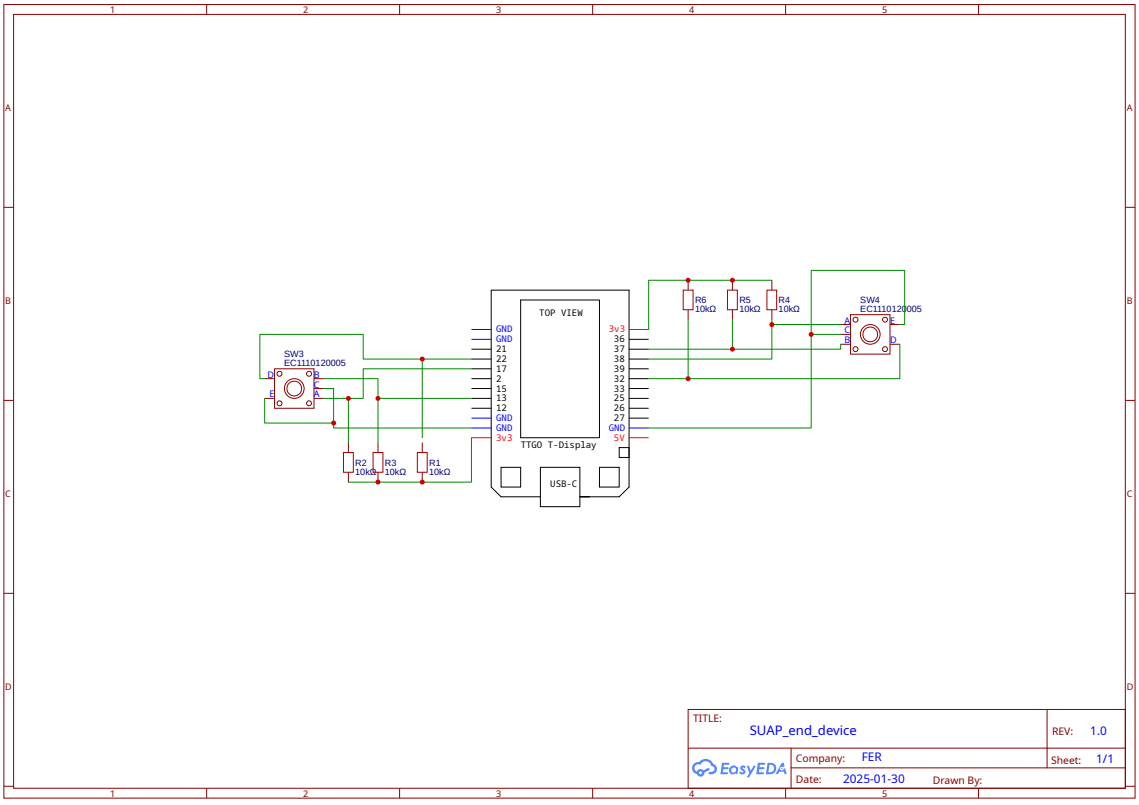
Sustav se u najjednostavnijoj varijanti sastoji od jednog poslužitelja i jednog krajnjeg čvora što je dovoljno za demonstraciju funkcionalnosti.



Ilustracija 1: Razvojna pločica TTGO T display sa spojenim enkoderima

3. Tehničke značajke

Krajnji čvor je uređaj TTGO T display na kojega su dodatno spojeni rotacijski enkoderi. TTGO T display se sastoji od mikrokontrolera ESP32, ugrađene memorije i malog LCD ekrana. Nema spojene iste GPIO izvode kao i ESP-WROOM_32. Služi kao točka interakcije korisnika sa sustavom od kojega preuzima podatke i zadatke te mu prikazuje povratnu informaciju.



Ilustracija 2: Shema spajanja enkodera na TTGO T Display

3.1 O ESP32

ESP32 je kineska serija mikrokontrolera koju je razvila i proizvodi kineska tvrtka Espressif Systems. Popularni su zbog relativno niske cijene i velikih mogućnosti.

Najstariji model iz serije, istoimeni ESP32, ima ugrađena dva 32-bitna procesora Xtensa LX6 koji koriste Harvardsku arhitekturu pristupanja memoriji da bi se poboljšale performanse. (Espressif Systems Co., 2023)

Pored moćnih procesora posjeduju dodatno sklopovlje za obradu signala u hardveru. Za ovaj projekt je značajan brojač impulsa PCNT, a postoje i upravljački sklopovi za različitu periferiju kao što su: UART, I2C, SPI, I2S, TOUCH (kontroler za panel), CAN bus, ETHERNET PHY, Bluetooth i WiFI radio

WiFI radio se koristi za komunikaciju između mikrokontrolera i REST API poslužitelja

Zbog radijskog primopredajnika, ESP32 nije pogodan za dugotrajno napajanje baterijama, već je poželjno baterije koristiti kao pomoćni izvor napajanja u slučaju nestanka električne energije iz mreže.

Radio se može isključiti zbog uštede energije.

Budući da je ESP32 serija mikrokontrolera, na različitim modelima su dostupni različiti procesori i dodatno sklopovlje.

Dodatno, neki modeli, kao što je najstariji model ESP8266, se više ne proizvode što može biti nezgodno zbog dostupnosti rezervnih dijelova za uređaje u kojima se koriste, a direktna zamjena za iste ne postoji.

S druge strane, Atmel ATmega328P, koji se koristi na razvojnom sustavu Arduino UNO čiji je razvoj započet 2009. godine (Arduino Team, 2021) i još uvijek je popularan razvojni sustav za kojim postoji potražnja, se još uvijek proizvodi. Atmel AtMega328P je dio serije mikrokontrolera ATmega8 koja je nastala tijekom 90-ih godina 20. stoljeća. (Wikipedia, 2025)

ESP32 može biti nezgodan za korištenje s vanjskom periferijom jer neki GPIO priključci moraju za vrijeme podizanja sustava biti u predviđenom stanju, a promjena njihovog stanja je dozvoljena nakon podizanja sustava. Moguće ih je koristiti za dijagnostiku pokretanja prije učitavanja programa, a njihovo nepravilno korištenje onemogućava neke funkcije mikrokontrolera kao što su: pokretanje sustava, UART, korištenje vanjske SPI flash memorije, snimanje programskog koda na uređaj. Njihova stanja nakon pokretanja ostaju nepromijenjena ukoliko se isto ne napravi kroz korisnički program

ESP32 has 5 takvih priključaka:

GPIO 0 – pritegnut na napajanje – koristi se za odabir izvora za podizanje sustava

GPIO 2 – pritegnut na masu – koristi se s GPIO 0 za odabir preuzimanja firmware-a s vanjskog izvora

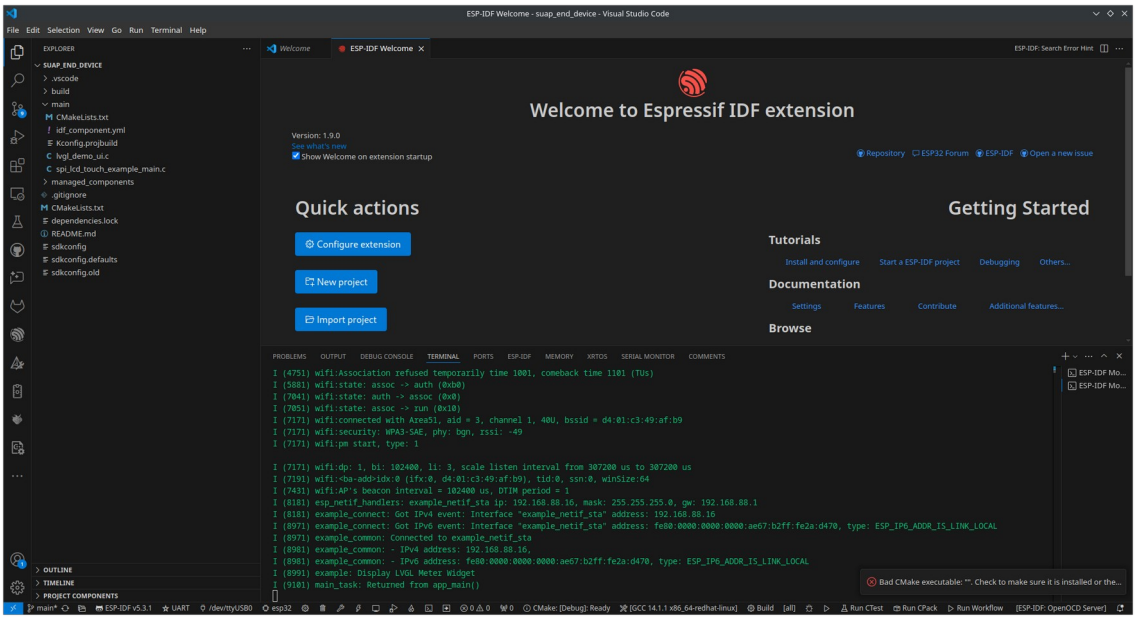
GPIO 5 – pritegnut na napajanje – povezano s odabirom načina rada SDIO (kontroler za SD karticu)

GPIO 12 (MTDI) – pritegnut na masu – odabir radnog napona za SDIO

GPIO 15 (MTDO) – pritegnut na napajanje – način rada SDIO i ispis dijagnostičkih poruka pri pokretanju

Postavke ugrađenih otpornika za pritezanje izvoda ostaju nepromijenjene dok ih ne primijeni korisnički program. (Espressif Systems Co., 2023.)

Korišteni model mikrokontrolera ESP32 nije preporučen za upotrebu u novodizajniranim uređajima.



Ilustracija 3: Primjer dijagnostičkog ispisa na ESP32

3.2 Vanjske biblioteke

3.2.1 LVGL

LVGL je biblioteka za stvaranje grafičkog sučelja napisana u programskom jeziku C. Koristi se u kombinaciji s upravljačkim programima za ekran i ulazne uređaje. U inicijalizacijskom dijelu programa se u strukturu podataka *disp_drv* tipa *lv_disp_drv_t* unose podaci o ekranu, a zatim se u zasebne strukture podataka unose podaci o ulaznim uređajima. Nakon toga se u glavnom potprogramu pokreće grafičko sučelje definirano nizom naredbi za oblikovanje prikazanih elemenata i interakcije s korisnikom. Interakcija s korisnikom se modelira unutar callback funkcija (funkcije koje se izvršavaju kad se nad nekim grafičkim elementom dovrši neki događaj) povezanih s grafičkim elementima prikazanim na ekranu. Za zaustavljanje izvođenja nakon instanciranja do kraja inicijalizacije koristi semafor. (LVGL, 2024.)

3.2.2 esp_lcd_ili9341

Ovo je univerzalni upravljački program za LCD ekran ili9341 i slične ekrane koji za komunikaciju koriste sučelje SPI. Prvo je potrebno inicijalizirati strukture podataka koje opisuju ekran (njegove priključke, razlučivost, orijentaciju, brzinu osvježavanja ekrana itd.), a zatim se u pokazivač na upravljački program upisuje adresa strukture podataka koja nastaje instanciranjem upravljačkog programa. Nakon toga se izvršavaju dodatna podešavanja upravljačkog programa. Strukture podataka stvorene instanciranjem upravljačkog programa se predaju biblioteci LVGL koja se preko njih povezuje s ekranom. (Espressif Systems Co.)

3.2.3 espressif/button

Zaglavlje ove biblioteka se zove *iot_button.h* i služi za lakšu integraciju tipkala u sustav. U inicijalizacijskoj strukturi podataka je potrebno predati vrstu tipkala, trajanje dugog i kratkog pritiska u milisekundama, broj GPIO izvoda na koji je tipkalo spojeno te logičku razinu ulaznog signala za koju se smatra da je tipkalo pritisnuto.

Umjesto jednostavne provjere je li tipkalo pritisnuto ili nije, moguće je generirati različite statusne signale koji se mogu koristiti u korisničkom programu za upravljanje programskim funkcijama. (Espressif Systems Co.)

3.2.4 esp_littlefs

Upravljački program za rad s datotečnim sustavom LittleFS. Služi za pronalaženje tražene LittleFS particije i omogućuje čitanje datoteka s iste. (joltwallet, 2025)

3.2.5 cJSON

Biblioteka za rad s JSON objektima. U ovom projektu se primarno koristi za deserijalizaciju JSON stringova, ali se kože koristiti i za stvaranje istih. Za rad s podacima koristi strukture i pokazivače.

Funkcionalnost vanjskih biblioteka je detaljnije opisana u poglavlju 3.8.

3.3 Spremljena konfiguracija u memoriji

3.3.1 Spremanje konfiguracije

Za spremanje korisničke konfiguracije koristi se zasebna LittleFS particija veličine 960kB što je i više nego dovoljno za spremiti potrebne podatke. Konfiguracija sučelja se nalazi u datoteci *interface_config.json* i sastoji se od popisa korisnički definiranih grafičkih elemenata (gumbi) i s njima povezanih zahtjeva. Spremanje u json datoteku umjesto direktne implementacije unutar programskog koda korisniku omogućava laku doradu starih i stvaranje novih gumba

Konfiguracija treba biti lako dostupna za izmjene i dopunjavanje. Zahtjeve je lako moguće pogrešno definirati pa je njihova laka korekcija nužna.

Svaki unos u konfiguraciji predstavlja jedan gumb u krajnjem desnom stupcu na ekranu.

Primjer unosa konfiguracije:

{

"widget_type": "button", -> vrsta elementa na ekranu; implementiran samo gumb

"label": "RPL1", -> oznaka na gumbu

"group": 1, -> grupa u koju se dodaje gumb, treba za odabir enkodera

"slider": { -> izbornik za odabir vrijednosti, ako se ne prikazuje treba biti *null*

"min": 0, -> minimalna dozvoljena vrijednost odabira

"max": 5, -> maksimalna vrijednost odabira

"step": 1 -> rezervirano za buduću upotrebu jer LVGL trenutno ne podržava

},

"method": "POST", -> HTTP metoda, za slanje zahtjeva je uvijek POST

"request":{ -> zahtjev koji se šalje kad se pritisne gumb; detaljnije opisan kasnije

"id": "predef_request2",

"network": "WiFi1",

"network_type": 0,

"interface": "WIFI",

"sourceID": 0,

"targetID": 28,

"body": {

"request_type": 1,

"logical_clock": 2,

"device_id": 2,

"data": {

"type": 1,

"old_state": 0,

"new_state": 1

}

}

}

}

3.3.2 Particije

Budući da se korisnička konfiguracija sprema u datoteku na zasebnoj particiji, potrebno je ručno definirati particijsku tablicu imajući na umu potrebe i ograničenja. Particijska tablica se u ovom slučaju nalazi u datoteci *suap_end_device_littlefs.csv*.
U slučaju rada s jednom aplikacijom bez ažuriranja preko bežičnog medija, normalno se koriste tri particije: *NVS*, *PHY_init* i *factory*. Prve dvije sadrže inicijalizacijske podatke, a particija *factory* sadrži prevedeni oblik korisničkog programa.
Uz to je potrebno definirati dodatnu particiju za korisničke podatke tipa LittleFS koja se nalazi iza particije *factory* i služi za spremanje konfiguracije.

3.4 Registracija na poslužitelj i čuvanje podataka

Sustav predviđa rad s identifikatorima, a ne MAC ili IP adresama. Stoga je svaki spojeni uređaj koji sudjeluje u komunikaciji potrebno registrirati. Registracija se vrši na zasebnoj pristupnoj točki na REST API poslužitelju. Da bi registracija bila uspješna, na poslužitelj je potrebno poslati identifikator (ignorira se pri registraciji, potreban zbog strukture podataka pa se može poslati bilo što, npr. -1), IP adresu, MAC adresu i listu perifernih uređaja. Budući da je IP adresa podložna promjenama, a MAC adresa stalna, identifikatori se dodjeljuju na temelju MAC adrese. Registrirani čvorovi se stavljaju u listu susjeda koju je moguće dohvatiti s bilo kojeg uređaja koji ima

3.5 Senzori, aktuatori, poruke, prikaz (obrada) odgovora

Svaki senzor spojen na neki od uređaja zahtijeva upravljački program prilagođen čitanju jednog podatka i pripadne mjerne jedinice. U slučaju da senzor može mjeriti više veličina, svaki podatak se očitava i pakira zasebno i to se onda tretira kao više zasebnih senzora. Slično se tretira i aktuator: ako uređaj ima više funkcija, svaka se tretira kao zasebni uređaj. Poruka za korisnika se prikazuje direktno na ekranu, kao i

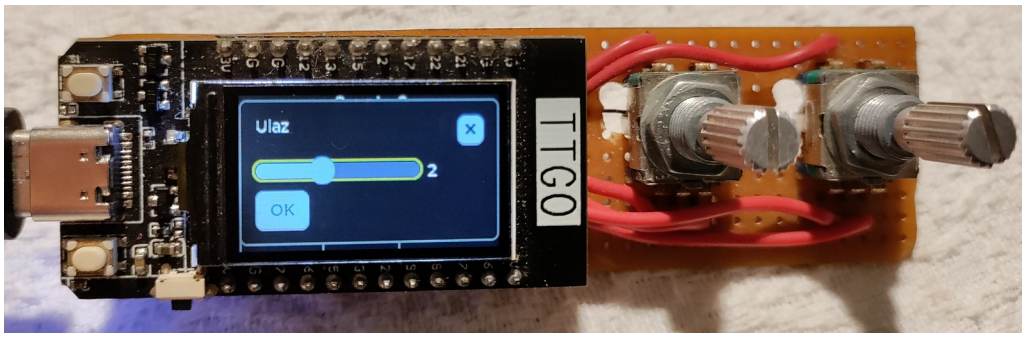
odgovori s očitanjima.

3.6 Proširenja grafičkog sučelja

Grafičko sučelje, osim gumba za generiranje podataka i njihovu razmjenu s poslužiteljem ima i dodatne gumbe koje u konfiguraciji definira korisnik. Uz te gumbe, postoje dodatne poruke koje su vezane da funkcionalnost povezanu s korisnički definiranim gumbima. Neki gumbi imaju, nakon što ih korisnik pritisne, mogućnost biranja vrijednosti pomoću klizača. Taj klizač se prikazuje u okviru za poruku koji se briše nakon korisnikovog potvrđivanja željene vrijednosti. (LVGL, 2024)

Uz to, grafičko sučelje može korisniku prikazati poruku s drugog uređaja i od njega preko te poruke tražiti odgovor. Taj odgovor korisnik može izabrati kroz matricu gumbi prikazanu u okviru za poruku.(LVGL, 2024)

Zadnje proširenje je isto okvir za poruku, ali s funkcijom prikaza očitanja iz odgovora pristiglog s drugog uređaja.



Ilustracija 4: Odabir željenog stanja aktuatora pomoću klizača

3.7 Rad sa zahtjevima

Svi zahtjevi u sustavu pristižu na centralni REST API poslužitelj koji ih stavlja u zasebne redove čekanja za svaki uređaj. Time se osigurava premještanje opterećenja na uređaj s više resursa i pohrana zahtjeva ako je ciljani uređaj trenutno isključen. Poslužitelj zahtjeve čuva u svojoj radnoj memoriji.

Zahtjevi se na poslužitelju spremanju u redove čekanja koji su pohranjeni u rječnik. Ako u rječniku ne postoji red čekanja za neki uređaj, on se stvara automatski. Za svaki

red čekanja se može provjeriti koliko zahtjeva ima u njemu prije čitanja istih. Tako, na primjer, krajnji uređaj može prije obrade serije zahtjeva u nekom početnom trenutku provjeriti trenutni broj pristiglih zahtjeva, a zatim započeti s preuzimanjem i obradom jednog po jednog.

Time se izbjegava istovremeno povlačenje velikog broja zahtjeva i prepunjavanje memorije na mikrokontroleru. Svaki zahtjev ima ograničen broj znakova koji može sadržavati (ograničenje je vezano za alociranje memorije na krajnjem čvoru) zbog čega je korištenje memorije moguće predvidjeti.

Zbog uštede resursa na krajnjem čvoru, on ne čeka obavijest s poslužitelja, nego sam provjerava ima li za njega na poslužitelju dostupnih novih zahtjeva te ih sam povlači za obradu. Na kraju obrade sam šalje odgovor izvorišnom uređaju na poslužitelj s kojega će biti preuzet.

Osim uštede memorije, korištenje centralnog poslužitelja olakšava programiranje jer nije potrebno voditi brigu o usklađivanju rada poslužiteljskog procesa, korisničkog sučelja i pozadinskih servisa (povlačenje zahtjeva i odgovaranje). Poslužitelj na krajnjem uređaju bi omogućio direktnu komunikaciju između uređaja, ali bi značajno otežao održavanje liste susjeda. Jedan čvor bi i dalje morao biti vodeći i stalno aktivan zbog dodavanja novih čvorova u mrežu što bi značilo da on uvijek mora imati točnu listu susjeda koju će slati novom čvoru. Jednostavnije je tu listu održavati na zasebnom poslužitelju.

Zahtjev se sastoji od 3 glavna dijela: polja s podacima, okvira s metapodacima o zahtjevu te okvira s metapodacima o uređajima i mreži.

Polje s podacima je najvažniji dio zahtjeva i može imati 3 vrste:

- podaci s očitanjem senzora
- podaci sa stanjem aktuatora
- podaci za direktnu interakciju s korisnikom.

Podaci s očitanjem senzora su najjednostavniji tip polja s podacima koje se može sastojati od očitanja i mjerne jedinice ili je prazno unutar okvira zahtjeva. Svaki

odgovor na zahtjev može prenijeti najviše jedno očitanje jedne vrijednosti s jednog senzora. Kad se šalje zahtjev, onda je polje s podacima prazno, a u odgovoru je popunjeno.

Stanje aktuatora se opisuje s 2 podatka: staro stanje (old_state; stanje aktuatora prije promjene) i novo stanje (new_state; stanje aktuatora nakon promjene). U zahtjevu za čitanje podaci o stanju aktuatora ostaju prazni, a u odgovor se upisuje staro stanje. U zahtjev za postavljanje aktuatora se upisuje traženo novo stanje, a u odgovoru se vraćaju i stanje aktuatora prije zahtjeva za promjenom i stanje aktuatora nakon promjene.

Staro stanje aktuatora se u odgovoru na zahtjev može iskoristiti za vođenje evidencije da se vidi je li ono primijenjeno ako je bilo drugačije od traženog.

Interakcija s korisnikom se sastoji od kratke poruke i eventulanog slanja odgovora po potrebi. Kao odgovor se može izabrati jedan od ponuđenih brojeva, a njihova značenja je potrebno navesti u poruci.

Struktura zahtjeva:

- metapodaci vezani za mrežu

{
  "id": "string", -> identifikator zahtjeva; jedinstveni identifikatori nisu implementirani jer nisu bitni za testiranje rada
  "network": "string", -> identifikator mreže, korisno ukoliko se sustav proširi za rad s više mreža (REST API poslužitelj ima više mrežnih kratica)
  "network_type": 0, -> vrsta mreže; u ovom slučaju je implementiran samo WiFi
  "interface": "string", -> mrežna kartica na uređaju koji šalje zahtjev, korisno kod unaprijed spremljenih zahtjeva
  "sourceID": broj, ->identifikator uređaja koji šalje zahtjev
  "targetID": broj, -> identifikator uređaja kojemu je zahtjev namijenjen
  "body": {} -> tijelo zahtjeva
}

- body": {

"request_type": broj, ->vrsta „zahtjeva” (zahtjev ili odgovor)

"logical_clock": broj, -> globalni logički sat

"device_id": broj, -> identifikator perifernog uređaja

"data": {} -> korisni podaci

}

- data": {

"type": 2, -> vrsta perifernog uređaja, daje kontekst podacima; rad s korisnikom

"message": "message", ->poruka namijenjena za korisnika

"input_required": false, -> treba li korisnik odgovoriti?

"user_input": 0 -> korisnikov odgovor (podatak seignorira ako nije potreban)

}

- "data": {

"type": 1, -> vrsta perifernog uređaja, daje kontekst podacima; rad s aktuatorom

"old_state": 0, -> staro stanje aktuatora

"new_state": 1 -> novo stanje aktuatora

}

- "data":{"type": 0, -> vrsta perifernog uređaja, daje kontekst podacima; rad sa senzorom

"measurement":-58, -> vrijednost očitanja

"unit":"dBi" -> mjerna jedinica

}

Zahtjev se obrađuje u više koraka:

1) preuzimanje i parsiranje zahtjeva

Za preuzimanje zahtjeva je zadužena beskonačna petlja koja svakih nekoliko sekundi provjerava ima li dostupnih zahtjeva. To se radi povlačenjem podatka s krajnje točke *MessageCount* na REST API poslužitelju. Nakon toga, krajnji uređaj,

jednu po jednu, povlači svaki zahtjev (poruku) te ga obrađuje.

Prvi korak obrade zahtjeva je njegovo parsiranje iz JSON formata u oblik razumljiv programskom jeziku C korištenjem biblioteke cJSON. Iz zahtjeva se izvlače svi podaci koji su potrebni za njegovu obradu i naknadno stvaranje odgovora.

Nakon izvlačenja podataka iste je potrebno obraditi. Ako je u pitanju zahtjev, izvodi se dio programa vezan za obradu istog. Ako je u pitanju odgovor, on se prikazuje na ekranu krajnjeg uređaja u obliku prozora s porukom za korisnika. Ako se šalje zahtjev za očitavanjem stanja, novo stanje mora biti prazno, a u staro se upisuje trenutno stanje aktuatora. Ako je potrebno postaviti aktuator, njegovo traženo novo stanje je upisuje u varijablu new_state.

2) rad s podacima

   Svaka vrsta periferije zahtijeva drugačiju obradu podataka.

   Podaci za senzor se obrađuju samo u sklopu odgovora tako što se čitaju iz zahtjeva/poruke te se prikazuju na ekranu uređaja.

   Podaci s aktuatora se čitaju u ovisnosti što s njima treba raditi. Ako se samo očitava stanje, ono se upisuje u varijablu *old_state*. U slušaju postavljanja novog stanja, prvo se očitava staro stanje i upisuje u varijablu *old_state*, a zatim se postavlja novo stanje. Nakon toga se opet očitava stanje aktuatora i rezultat upisuje u varijablu *new_state*.

   Kad se radi o interakciji s korisnikom, poruka se obavezno prikazuje na ekranu, a odgovor od korisnika se može tražiti po potrebi. U tom slučaju korisnik može odabrati jedan od 4 broja, a moguće odgovore koje ti brojevi pretpostavljaju je potrebno dostaviti u poruci.

3) stvaranje i slanje odgovora

   Odgovor za sve vrste zahtjeva se stvara po istom principu: u JSON objekt s metapodacima se ugrađuju podaci. Za te podatke je potrebno znati kontekst, a on ovisi o vrsti uređaja koji je vezan za zahtjev. Metapodaci su skoro isti, ali su zamijenjeni izvorišni i ciljni uređaj te je vrsta poruke ODGOVOR, a ne ZAHTJEV.

Kao odgovor na zahtjev za očitanjem senzora pakiraju se podaci preuzeti sa senzora i to se šalje na REST API poslužitelj.

Ista stvar se radi i s podacima vezanim za aktuator. Ovisno o odrađenoj akciji, potrebni podaci se ugrađuju u JSON objekt namijenjen za slanje na poslužitelj.

Za korisnički odgovor se, osim samog odgovora, natrag šalje i izvorna poruka s upitom zbog lakšeg snalaženja u slučaju greške.

Ilustracija 5: Primjer uspješno pročitane poruke za korisnika

3.8 Vanjsko sklopovlje i upravljački programi

3.8.1 Ekran ST7789

Ekran ST7789 je na ESP32 spojen preko sučelja SPI Postoji više verzija tog ekrana, a za potrebe projekta se koristi ekran razlučivosti 240x135 piksela s pomakom početne točke u točku (52,40).

Kao upravljački program koristi esp_lcd_ili9341 koji ima podršku za različite ekrane. Upravljačkom programu je potrebno predati podatke o kontaktima na koje je ekran spojen s mikrokontrolerom te osnovne podatke o ekranu (razlučivost i pomak). Problem s upravljačkim programom je taj da u dokumentaciji niti u primjerima nije opisano podešavanje pomaka slike pa je kako se to radi potrebno tražiti u programskom kodu upravljačkog programa.

3.8.2 Grafička biblioteka LVGL

Za interakciju s korisnikom se koristi grafička biblioteka LVGL napisana u programskom jeziku C. Zbog toga je pogodna za korištenje na mikrokontrolerima. Iako posjeduje opširnu dokumentaciju, naprednije funkcije ipak nisu detaljno opisane.

Za povezivanje s ulaznim uređajima koristi vlastite strukture podataka tipa *lvgl_indev_drv_*t koje nisu upravljački programi, nego služe sa spremanje podataka o vrsti ulaznog uređaja i referencu na funkciju za izvlačenje podataka iz istog. Instanciranje virtualnog uređaja se vrši preko strukture *lvgl_indev_t*. Postupak povezivanja ulaznih uređaja s grafičkim sučeljem je loše dokumentiran pa je potrebno proučiti programski kod i koristiti dodatne izvore informacija, npr. forume. Kao ulazne uređaje moguće je koristiti ekrane osjetljive na dodir, tipkala, tipkovnice, enkodere i druge uređaje namijenjene za interakciju s korisnikom.

LVGL ima gotove funkcije za stvaranje različitih grafičkih elemenata pa je rad s njima vrlo jednostavan, ali zahtijeva pisanje relativno velike količine programskog koda za njihovo usklađivanje. Da bi se olakšalo upravljanje grafičkim sučeljem, elementi se mogu grupirati što je korisno za ulazne uređaje kao što su enkoderi i tipkovnice.

Postoje razni primjeri programskog koda koji implementira različite funkcionalnosti, ali su njihove napredne funkcije često nedokumentirane. Pozadinska funkcionalnost povezana s grafičkim elementima se poziva preko callback funkcija koje se mogu pridružiti svakom elementu zasebno ili ista funkcija može posluživati više grafičkih elemenata. (LVGL, 2024)

3.8.3 Brojač impulsa

Događaje na enkoderima obrađuje ugrađeni sklop PCNT (brojač impulsa, eng. pulse counter) koji ima više načina rada. Može brojiti rastuće i padajuće bridove ulaznog signala ili predviđene naponske razine istog. U mikrokontrolerima ESP32 postoji nekoliko višekanalnih PCNT sklopova, ali postoje i verzije bez njih. Najčešće primjene su im određivanje frekvencije ulaznog signala brojenjem impulsa u vremenskom signalu i dekodiranje para kvadratnih signala u brzinu i smjer što se koristi za rad s enkoderima. Moguće je definirati kritične točke kada sklop može generirati događaje (npr. enkoder je napravio X koraka) u kojima upozorava korisnički program da je došlo do istih. Iako u kontekstu ovog rada nije kritično uhvatiti baš sve korake enkodera, PCNT nudi kompenzaciju gubitaka zbog preljeva.

Upravljački program za PCNT se nalazi u biblioteki *driver/pulse_cnt*. U višedretvenim sustavima postoji opasnost od gubitka konteksta izvođenja dretve. Zbog toga u upravljačkom programu za PCNT postoje funkcije koke su pisane na način da se u višedretvenom sustavu mogu koristiti bez dodatnog spremanja konteksta i korištenja mehanizama za sinkronizaciju. To su funkcije *pcnt_new_unit()* i *pcnt_new_channel()* i za njih autor garantira pravilno „istovremeno” izvođenje u višedretvenom programu. Za korištenje u prekidnim potprogramima su pogodne funkcije *pcnt_unit_start()*, *pcnt_unit_stop()*, *pcnt_unit_clear_count()* i *pcnt_unit_get_count()* -> poziva se iz LVGL-ove callback funkcije za enkoder. Funkcije koje kao prvi argument primaju podatak tipa *pcnt_unit_handle_t* ili *pcnt_channel_handle_t* se ne smatraju sigurnima za istovremeno izvođenje u višedretvenim sustavima pa njihovo korištenje treba izbjegavati u zadacima koji se izvode odvojeno od glavne dretve ili koristiti semafore.

Za mikrokontrolere koji nemaju ugrađen PCNT (npr. ESP32-C serija) postoji biblioteka koja tu funkcionalnost obavlja softverski. (Espressif Systems Co.)

Sa PCNT sklopovljem su povezana 2 rotacijska enkodera koji su namijenjeni sa interakciju s korisnikom. Enkoder se sastoji od dva dijela: gumba koji je obično tipkalo i dvoizlaznog rotirajućeg generatora kvadratnog signala. Faze izlaznih signala su pomaknute za 90°. Izlazi su označeni slovima A i B, a često se koriste i oznake CLK i DT. Kada se enkoder okreće u smjeru kazaljke na satu, izlaz A ima ulogu generatora takta (CLK), a izlaz B je „podatkovni" izlaz, odnosno pomaknut je za 90° u odnosu za CLK i služi za određivanje smjera za okretanja. U slučaju okretanja enkodera u smjeru suprotnom od kazaljke na satu, uloga izlaza A i B su zamijenjene. Zato se za sklop PCNT koji je zadužen za rad s enkoderom radi inicijalizacija za rad u oba smjera.

Signali koji dolaze s tipkala ugrađenog u enkoder se obrađuju pomoću upravljačkog programa *iot_button* koji je ESP-ov univerzalni upravljački program za tipkala. Njegova inicijalizacija se vrši u istoj funkciji u kojoj se inicijalizira i PCNT sklop. (Espressif Systems Co.)

3.8.4 Datotečni sustav LittleFS

Littlefs je datotečni sustav dizajniran za mikrokontrolere. Otporan je na greške u slučaju problema s napajanjem, podržava automatsko upravljanje trošenjem ugrađene FLASH memorije, a njegova upotreba treba samo ograničene količine radne memorije. Ugrađene funkcije su slične POSIX-ovim, a operacije nad podacima su atomičke zbog povećanja pouzdanosti rada s podacima. Upravljački program za LittleFS je prije korištenja potrebno ručno podesiti, a nakon toga se rad s datotekama odvija kao na normalnom računalu. Zbog što šire kompatibilnosti je napisan u programskom jeziku C. Podaci se prenose u blokovima po principu *copy-on-write* s dodatnim kratkim zapisnicima. Struktura datotečnog sustava je stablasta, odnosno stablo blokova. Svaki blok se može zasebno alocirati. (littlefs-project, 2025)

3.8.5 Programska biblioteka cJSON

cJSON je standardna programska biblioteka za programski jezik C koja pomaže u obradi JSON stringova. Može raditi serijalizaciju, deserijalizaciju, stvaranje, izmjenu i izvlačenje podataka iz objekata. Podatkovna struktura se temelji na strukturama podataka i pokazivačima. Za gotovo svaku operaciju postoji ugrađena funkcija koja olakšava rad, a JSON objekt je moguće obraditi do najsitnijeg detalja. (Gamble, 2024) Zbog loše dokumentacije je za upotrebu velikog dijela funkcionalnosti treba proučavati programska zaglavlja. Nazivi funkcija su oblikovani vrlo intuitivno što olakšava potragu potrebne funkcije. Sama biblioteka je vrlo jednostavna za upotrebu i posao odrađuje savršeno.

Da bi se dupliciralo neki dio JSON objekta nije dobro kopirati memoriju korištenjem funkcije *memcpy(),* već je potrebno koristiti ugrađenu funkciju. Također, nije dobro pokušavati brisati dijelove objekta pozivanjem operacije brisanja nad pokazivačem na podobjekt jer je ona rekurzivna i briše cijeli objekt. Brisanje treba obaviti tek nakon kraja rada s objektom. cJSON je moćan alat koji uz pravilnu i pažljivu upotrebu

uvelike olakšava rad s JSON objektima.

3.9 Obrada HTTP odgovora

ESP-ov HTTP klijent u standardnog konfiguraciji može primati samo nefragmentirane odgovore na zahtjev. Zbog predviđenog načina rada sustava ovo nije poželjno svojstvo te ga je potrebno promijeniti. To se radi u funkciji koja služi za obradu HTTP odgovora. To je funkcija koju poziva HTTP klijent i mora raditi bez obzira na odgovor koji dobije. Ta funkcija može imati različitu složenost ovisno o potrebnoj funkcionalnost, ali niti primjer njezinog najsloženijeg oblika ne može bez promjena raditi s fragmentiranim paketima. U komentaru kod dijela funkcije u kojem se provjerava fragmentiranost piše da je moguće raditi s fragmentiranim odgovorima, ali niti u komentaru niti u dokumentaciji ne piše kako. Za to je potrebno umjesto provjere uvjeta (je li odgivir fragmentiran) napisati *true* jer brisanje provjere uvjeta rezultira programom koji odbacuje fragmentirane podatke, vjerojatno zbog pregaženja istoimenih varijabli. To bi moglo izazivati probleme za veće količine primljenih podataka.

Nakon što se podaci uspješno prime, njihova obrada ovisi o tome s koje pristupne točke na poslužitelju dolaze. Ako dolaze s pristupne točke *SuapApi*, iz njih se izvlače korisne informacije te se prikazuju korisniku u poruci na ekranu. Za podatke s pristupne točke *Registration*, oni se koriste za ažuriranje liste susjeda na krajnjem čvoru. Pristigli JSON string se obrađuje i podaci se prepisuju i niz struktura s podacima o susjedima.

Ilustracija 6: Primjer obrađenog odgovora s poslužitelja

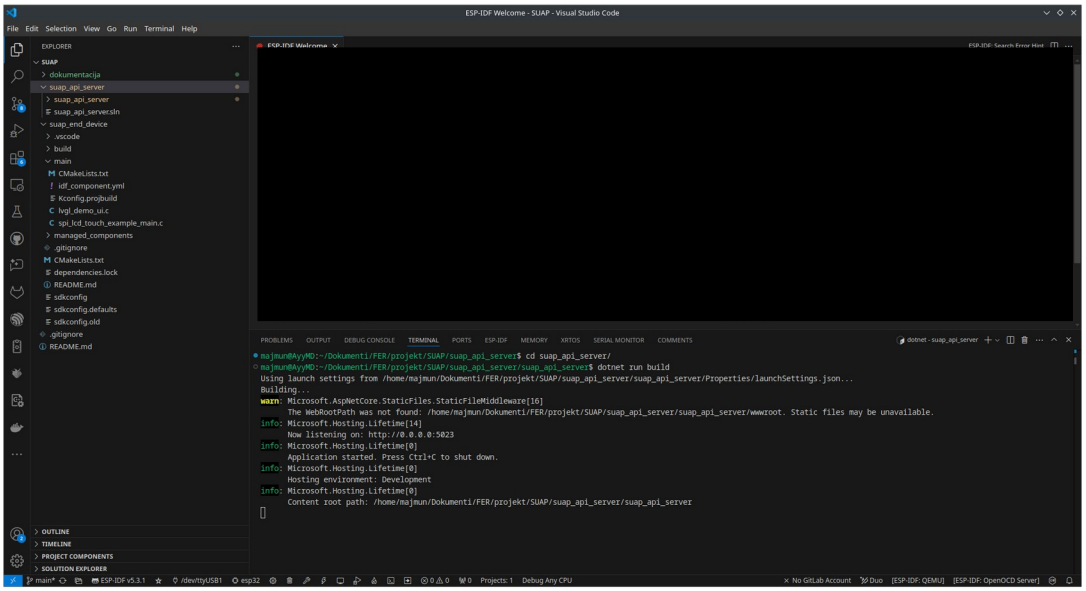
3.10 Aplikacijski poslužitelj

REST API poslužitelj je napisan u programskom jeziku C# uz korištenje radnog okvira .NET. Za potrebe projekta napravljena je implementacija za obradu zahtjeva tipa POST i GET koji služe za primanje podataka s krajnjeg čvora i slanje obrađenih podataka na isti. Podatak s krajnjeg čvora je obični brojač koji se zapisuje u varijablu i radi se njegova statistička obrada. Pamti se njegova zadnja poznata vrijednost, prosjek i brojač pristiglih poruka.

Ti podaci se na zahtjev šalju natrag krajnjem čvoru.

U standardnoj konfiguraciji .NET REST kontroler podatke šalje u fragmentiranom obliku što stvara probleme pri radu s tim podacima na mikrokontroleru. Budući da .NET stvara kontroler tek da dobije zahtjev, podaci koji nisu spremljeni u bazu podataka nestaju između pristiglih zahtjeva pa je, budući da sustav ne koristi bazu podataka, podatke potrebno spremati u statičkom kontekstu da bi se sačuvali između stvaranja kontrolera za obradu zahtjeva.

Uz testnu funkcionalnost za provjeru komunikacije, poslužitelj također vrši registraciju spojenih uređaja i sprema njihove zahtjeve (poruke) tako što ih stavlja u redove čekanja ciljanog uređaja. Svaki skup funkcionalnosti ima svoj kontroler i HTTP pristupnu točku.



Ilustracija 7: Ispis konzole pokrenutog poslužitelja

Kontroler za registraciju radi s metodama *Register* koja je indempotentna (zbog ažuriranja IP adresa) i *GetPeers* koja služi za povlačenje liste susjeda. Podaci tipa *EndDevice* se spremaju u obični niz podataka.

Funkcionalnost kontrolera za grupiranje i čuvanje je nešto malo zahtjevnija. Svakom uređaju se, kad se zaprimi prvi zahtjev namijenjen za njega, dodjeljuje red čekanja. Redovi čekanja su pohranjeni u rječnik i dohvaćaju se na temelju identifikatora perifernog uređaja. Svaki put kad na server stigne neki zahtjev, prvo se provjerava postoji li red čekanja predviđen za taj zahtjev. Ako red čekanja postoji, zahtjev se sprema u njega, a ako ne postoji, prije spremanja se stvara pripadajući red čekanja. Spremanje i dohvaćanje zahtjeva je moguće samo jedan po jedan. Također je moguće provjeriti i broj pohranjenih zahtjeva za svaki uređaj.

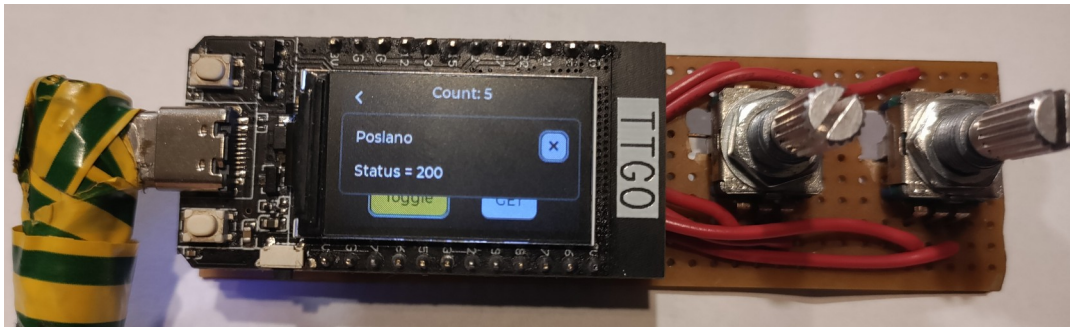
Poslužitelj je svjestan konteksta za svaku vrstu podataka s periferije pohranjenih u zahtjev.

3.11 Opis rada

Korisnik pomoću lijevog enkodera stvara podatak: brojač pritisaka na bilo koji od dva prikazana gumba. (LVGL, 2024) Kada želi posalti podatke na poslužitelj pomoću desnog enkodera bira gumb s natpisom POST i pritisne enkoder. Kada želi dohvatiti podatke poslužitelja, odabere gumb s natpisom GET i napravi isto.

Kao primjer informacijske komunikacije s poslužiteljem šalje se trenutni brojač na krajnjem čvoru sa POST zahtjevom te se ispisuje poruka sa statusom zahtjeva. Obrađeni podaci se mogu povući s poslužitelja korištenjem GET zahtjeva te se korisniku ispisuju u poruci.

Dodatno, korisniku se na ekranu, u skladu s konfiguracijom, prikazuju i gumbi povezani sa zahtjevima koje korisnik sam definira. Ako je potrebno ručno unijeti parametar zahtjeva, to se može napraviti korištenjem klizača koji se prikaže nakon odabira gumba koji je povezan sa zahtjevom za koji je to potrebno. Takvi zahtjevi se šalju na pristupnu točku na REST API poslužitelju koja služi za agregaciju. Odgovor se ne čeka, nego na uređaju postoji zasebni proces koji preuzima zahtjeve s poslužiteljima među kojima se nakon nekog vremena treba naći i odgovor na poslani zahtjev.



Ilustracija 8: Uspješno poslan zahtjev s brojačem

Krajnji uređaj svakih nekoliko sekundi čita broj dostupnih zahtjeva pa ih, na temelju tog broja čita i obrađuje jednog po jednog. Ako je na zahtjev potrebno odgovoriti, odgovor se šalje na poslužitelj, a ciljani uređaj ga povlači sam iz svojeg reda čekanja. Obrada podataka pristiglih s pristupne točke *Datagram* je nešto složenija jer je moguće primiti 2 vrste podataka: broj zahtjeva (poruka) i zahtjev (poruku).

Broj zahtjeva je cijeli broj i on se koristi za orijentaciju koliko poruka treba preuzeti u jednom ciklusu obrade. Ukoliko u međuvremenu stigne još koja poruka, ona se obrađuje u slijedećem ciklusu.

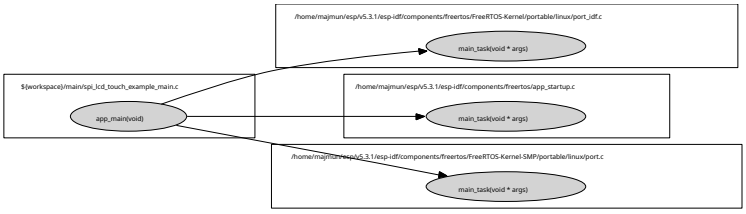
Obrada tih poruka je detaljnije opisana u poglavlju o radu sa zahtjevima.

3.12 Ograničenja

Glavno ograničenje napravljenog sustava je minimalna funkcionalnost potrebna za demonstraciju ideje. Iako je moguće lako dodavati gumbe i njima povezane zahtjeve, promjene vezane za periferne uređaje su komplicirane jer su upravljački programi i njihove instance strogo vezani za konkretan uređaj, a njihovo djelomično apstrahiranje bi zahtjevalo značajno refaktoriranje programskog koda. Dodatno, krajnji čvor izvodi relativno zahtjevan programski kod pa može doći do povremenog usporenog rada sustava. Uz to, firmware koji nastaje prevođenjem tog programskog koda zauzima veliku količinu memorije.

3.13 Prednosti

Iako je funkcionalnost izrađenog sustava trenutno ograničena, budući da se koriste standardne komponente i modularni programski kod, funkcionalnost istog je moguće proširiti bez narušavanja postojeće funkcionalnosti. Hardver koji je potreban za realizaciju sustava je relativno jeftin. Za pokretanje REST API poslužitelja je dovoljno osobno računalo koje na sebi ima instaliran .NET SDK, bežična pristupna točka za lokalnu mrežu i mikrokontroler s dodatnom opremom. Umjesto za rad s ESP32 programski kod se može preraditi za rad s bilo kojim drugim mikrokontrolerom koji može bežično pristupiti lokalnoj mreži.



Ilustracija 9: Ulazni dijagram funkcija krajnjeg čvora (uz dokument je priložena originalna slika koja se može povećati)

3.14 Opis programskog koda

Pogramski kod za SUAP je podijeljen u 2 dijela: programski kod za krajnji čvor i programski kod za REST API poslužitelj. Svaki dio programskog koda je zasebni projekt koji koristi svoju razvojnu okolinu.

Razvojna okolina za krajnji čvor je ESP-IDF, a za REST API poslužitelj se koristi radni okvir .NET i njegova pripadajuća radna okolina.

3.14.1 REST API poslužitelj

Za REST API poslužitelj je korišten objektno orijentirani jezik C# u kombinaciji s radnim okvirom .NET što omogućava relativno brzo i jednostavno razvijanje REST API sučelja za HTTP poslužitelj. Programski kod je podijeljen u više datoteka u nekoliko mapa. Uz programske datoteke je bitna još i datoteka s postavkama poslužitelja *appsettings.json*.s

3.14.1.1 Program.cs

Ovo je početna datoteka u kojoj se programski opisuje kako program treba nakon prevođenja pokrenuti i podesiti REST API poslužitelj. Programski kod izgleda poput uputa za korištenje koje govore glavnom programu što treba raditi.

3.14.1.2 SuapApiController.cs

Prima HTTP zahtjeve s krajnjeg čvora i obrađuje ih. Radi se o zahtjevima koji služe za testiranje komunikacije između poslužitelja i krajnjeg uređaja. To je klasa na temelju koje se stvara objekt s dvije funkcije i jednom članskom varijablom tipa RecentData. To su funkcije *public IActionResult GetData()* koja uzima obrađene podatke iz članske varijable, serijalizira ih u JSON string i šalje krajnjem čvoru i *funkcijapublic IActionResult PostData([FromBody] int broj)* koja od krajnjeg čvora prima poslanu vrijednost brojača, obrađuje ju i sprema izračunate vrijednosti u člansku varijablu.

3.14.1.3 DatagramController.cs

Služi kao kontroler za rad sa zahtjevima koji pristižu s krajnjim uređajima. Koristi jednu statičku člansku varijablu koja je rječnik u koji su pohranjeni redovi čekanja za svaki krajnji uređaj. Ima 3 funkcije koje služe za primanje zahtjeva, preuzimanje jednog zahtjeva i čitanje broja zahtjeva u redu čekanja za svaki pojedini krajnji uređaj. Za primanje zahtjeva se brine funkcija *public IActionResult PushMessage([FromBody] MainDatagram json_data)* koja za svaki primljeni zahtjev ako ne postoji radi red čekanja. Funkcija *public IActionResult PopMessage(int deviceID)* kao argument prima identifikator uređaja koji daje HTTP GET zahtjev i, ako postoji, vraća mu najstariju poruku koja je u njegovom redu čekanja. Broj poruka u redu čekanja za neki uređaj se dohvaća pomoću funkcije *public IActionResult MessageCount(int deviceID)*.

3.14.1.4 RegistrationController.cs

Za registraciju i vođenje evidencije o registriranim uređajima se koriste dvije funkcije: *public IActionResult Register([FromBody] EndDevice node)* koja radi registraciju uređaja i *public IActionResult GetPeers()* čija je jedina svrha bilo kojem uređaju koji pošalje zahtjev poslati popis registriranih uređaja.

3.14.1.5 ErrorViewModel.cs

Ovo je objekt koji služi za ispis greški s REST API kontrolera na konzolu u svrhu debugiranja.

3.14.1.6 RecentData.cs

Služi za strukturirano spremanje obrađenih podataka koji dolaze do krajnjeg čvora. Spremaju se zadnji poznati brojač s krajnjeg čvora, prosjek svih primljenih vrijednosti brojača i broj istih tih pristiglih vrijednosti.

3.14.1.7 ActuatorData.cs

Ovo je objekt za čuvanje podataka o stanju aktuatora.

3.14.1.8 BodyData.cs

Ovo je apstraktna klasa koja se odnosi na 3 vrste podataka koje se mogu ugraditi u podatkovno polje tijela zahtjeva. Automatski prepoznaje radi li se o podatku tipa *SensorData*, *ActuatorData* ili *UserData* na temelju identifikatora tipa ugrađenog u te objekte i na temelju toga određuje kontekst tih podataka.

3.14.1.9 DatagramBody.cs

Ovo je objekt čije članske varijable (osim podataka tipa *BodyData*) služe kao zaglavlje u kojem se opisuje što krajnji uređaj treba raditi s porukom i identifikator perifernog uređaja na koji se odnose ugrađeni podaci. Također evidentira lokalni logički sat uređaja s kojeg se šalje.

3.14.1.10 MainDatagram.cs

Ovo je objekt koji sprema podatke o pošiljatelju i primatelju. Također sprema i podatke o mreži. Kao podobjekt ima ugrađen podatak tipa *DatagramBody* sa svim pripadnim podacima.

3.14.1.11 SensorData.cs

Ovaj objekt pohranjuje senzorsko očitanje i mjernu jedinicu za isto.

3.14.1.12 UserData.cs

Ovdje se spremaju podaci vezani za razmjenu poruka s korisnikom. Spremaju se poruka za korisnika, zastavica je li potreban odgovor i korisnikov odgovor.

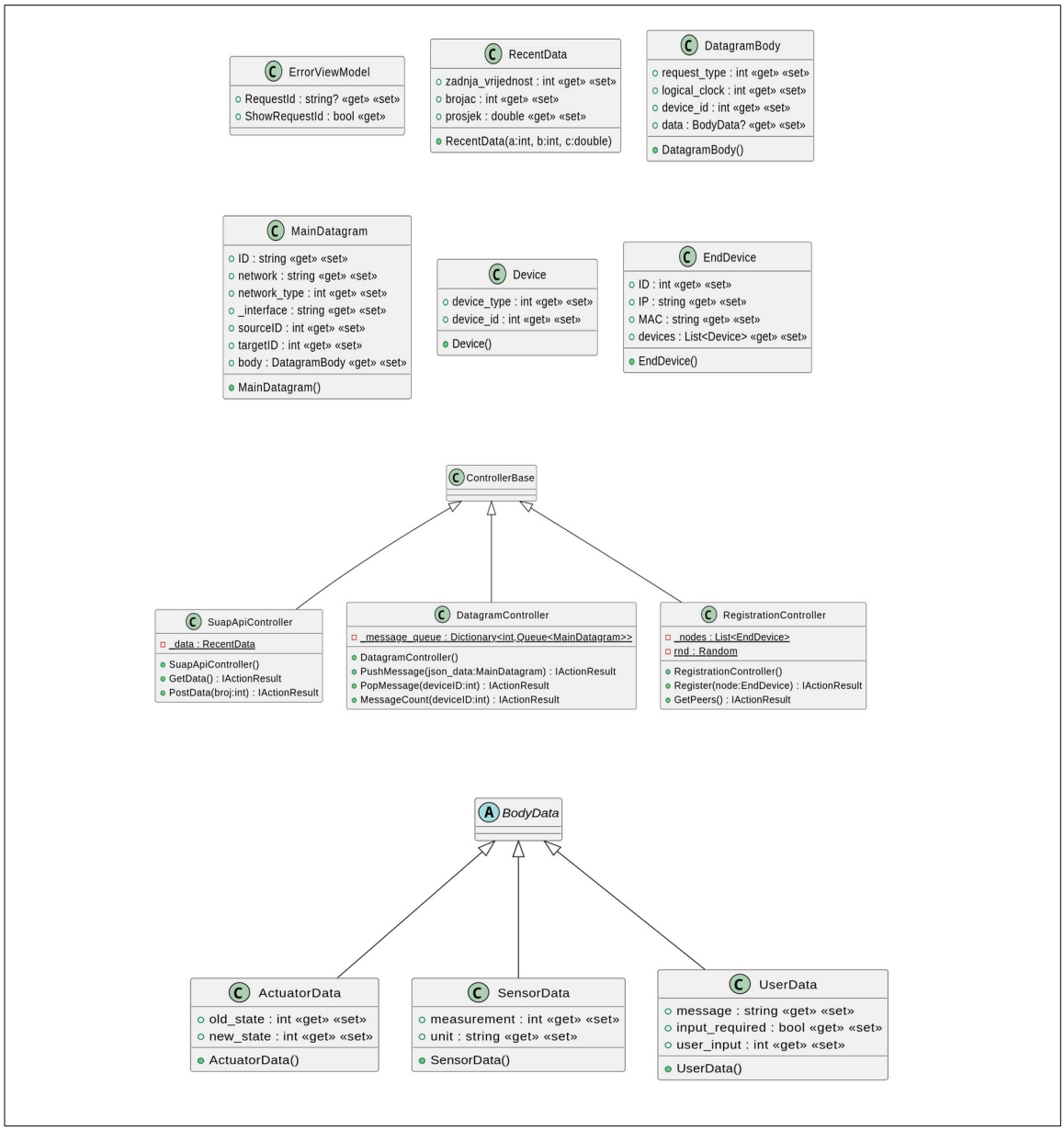
3.14.1.13 Device.cs

Predstavlja periferni uređaj spojen na *EndDevice*. Pamti se njegova vrsta i identifikator i za njih su definirane metode za postavljanje i čitanje.

3.14.1.14 EndDevice.cs

Ovaj objekt predstavlja uređaj koji je registriran na poslužitelj. Pamti njegov identifikator, IP adresu, MAC adresu i spojene periferne uređaje. Za svaku varijablu su

definirane metode za postavljanje i čitanje.



Ilustracija 10: Dijagram razreda za REST API poslužitelj. Veze među razredima se stvaraju implicitno tijekom prevođenja.

3.14.2 Krajnji čvor

Programski kod krajnjeg čvora je podijeljen u nekoliko datoteka spremljenih u nekoliko mapa od kojih svaka sadrži programski kod koji implementira jednu funkcionalnu cjelinu. Zbog različite funkcionalnosti implementirane po datotekama, datoteke koje implentiraju sličnu funkcionalnost su stavljene u istu mapu i koriste zajednička programska zaglavlja ako to ima smisla.

Datoteka *spi_lcd_touch_example_main.c* u sebi sadrži kod za postavljanje i inicijalizaciju uređaja. Nalazi se mapi *main* unutar projekta i sadrži glavni program. Zajedničke varijable i dijeljeni podaci se nalaze u datoteci *global_variables.h*.

- void app_main
  - glavni potprogram
  - radi inicijalizaciju sustava
  - stvara pomoćne dretve
  - pokreće grafičko sučelje i interakciju s korisnikom
  - redoslijed inicijalizacija:
    - inicijalizacija ekrana i potrebih upravljačkih programa
    - inicijalizaija enkodera
    - povezivanje s bežičnom mrežom
    - pokretanje čitanja konfiguracije i grafičkog sučelja

Funkcije za stvaranje grafičkog sučelja i rad s korisnikom su definirane u datoteci *user_interface.c*. Korisničko sučelje se sastoji od 2 osnovna dijela: Statički definiranih elemenata koji uvijek moraju raditi i služe za testiranje komunikacije s poslužiteljem i dinamički generiranih elemenata na temelju konfiguracijske datoteke.

- static int extract_relevant_data
  - iz pristiglog JSON stringa izvlači relevantne brojčane podatke

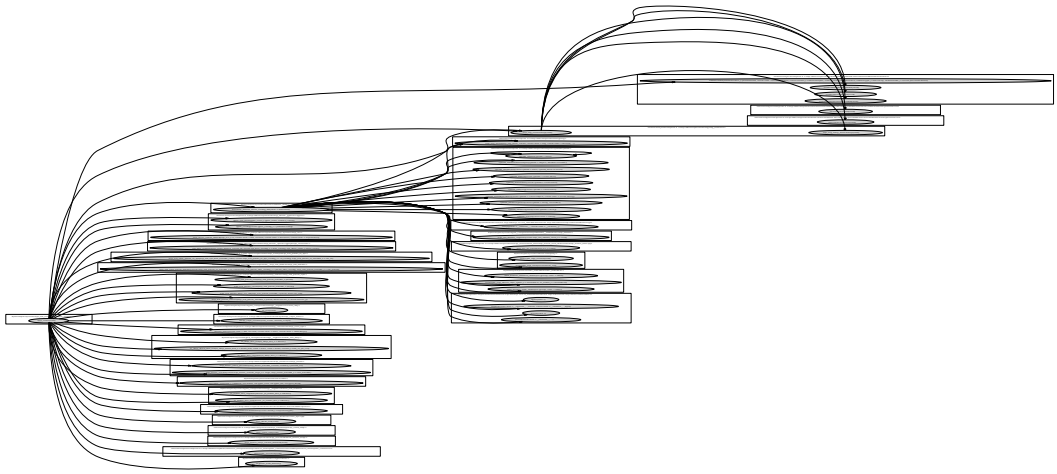
- esp_err_t client_event_http_handler
  - callback funkcija za rad s ESP-ov http klijenta
  - izvlači podatke iz pristiglog paketa i prepisuje ih u memoriju
- static void mevent_cb
  - callback funkcija za gumb za zatvaranje poruke kad se prikaže
  - zatvara poruku i briše ju iz memorije
- static void event_handler
  - callback funkcija za gumbe s natpisima *Button* i *Toggle*
  - kad se pritisne bilo koji od gornja 2 gumba povećava brojač za 1
- static void get_event_handler
  - callback funkcija
  - kad korisnik pritisne gumb s natpisom *GET* pokreće slanje zahtjeva za obrađenim podacima na poslužitelj
  - Kad se obavi slanje stvara poruku s primljenim podacima
- static void post_event_handler
  - callback funkcija
  - kad korisnik pritisne gumb s natpisom *POST* pokreće slanje brojača na poslužitelj
  - Kad se obavi slanje stvara poruku s povratnom informacijom
- void example_lvgl_demo_ui
  - stvara elementa grafičkog sučelja i stavlja ih na njihova mjesta
  - postavlja callback funkcije
  - grupira elemente
  - poziva se prije dodavanja dinamički generiranih elemenata
- void user_event_cb
  - obrađuje događaj na gumbu iz konfiguracije
  - dorađuje podatke koji se šalju na poslužitelj
  - šalje podatke na poslužitelj

    - stvara poruku za korisnika
    - korisniku prikazuje status slanja
- void slider_event_cb
    - obrada događaja na klizaču
    - pomak označene točke na klizaču
- void config_event_handler_with_slider
    - prepoznavanje pritisnutog gumba
    - stvaranje klizača
    - podešavanje klizača
    - logiranje pritisnutog gumba
- void config_event_handler_without_slider
    - prepoznavanje pritisnutog gumba
    - slanje podataka na poslužitelj
    - stvaranje poruke za korisnika
- int add_button_with_slider
    - stvaranje gumba
    - postavljanje teksta na gumb
    - pridruživanje enkodera gumbu
    - postavljanje callback funkcije za obradu koja zahtjeva klizač
    - poziva se iz petlje koja čita konfiguraciju
- int add_button_with_slider
    - stvaranje gumba
    - postavljanje teksta na gumb
    - pridruživanje enkodera gumbu
    - postavljanje callback funkcije za obradu koja ne zahtjeva klizač
    - poziva se iz petlje koja čita konfiguraciju

Funkcije za komunikacije su definirane u datoteci *http_handling.c* za funkciju za

izvlačenje podataka iz poslužiteljskog odgovora, dok su funkcije koje se pozivaju iz programa definirane u datoteci sa zaglavljima *http_handling.h* jer su statičke.

- esp_err_t client_event_http_handler
  - prepoznaje vrstu http događaja
  - preuzima podatke iz HTTP odgovora
  - rješava problem fragmentacije
  - akumulira podatke iz fragmenata
  - prepoznaje greške u obradi
- static int post_rest_function
  - inicijalizira HTTP klijent
  - priprema podatke za slanje
  - postavlja zaglavlja
  - šalje podatke
  - sprema odgovor s poslužitelja
  - obrađuje status slanja
- static int get_rest_function
  - inicijalizira HTTP klijent
  - postavlja zaglavlja
  - prima podatke
  - sprema odgovor s poslužitelja
  - obrađuje status primanja



Ilustracija 11: Izlazni dijagram funkcija krajnjeg čvora (uz dokument je priložena originalna slika koja se može povećati)

Funkcije za rad s periferijom vezane za upravljačke programe su spremljene u datoteke koje se nalaze u mapi *main/drivers*. To su datoteke za rad s ekranom, enkoderima i pohranom.

Programski kod za rad s ekranom je spremljen u dvije datoteke: *display.c* i *display.h*. U datoteci *display.c* su implementirane funkcije *example_lvgl_lock* i *example lvgl_unlock* i potrebne su zbog toga što operacije unutar programske biblioteke LVGL misu sigurne za višedretveno izvođenje pa je operacije dodavanja i brisanja elemenata potrebno izvesti u osiguranom okruženju i natjerati dretve da dijeljenim resursima pristupaju jedna po jedna korištenjem semafora, dok su ostale funkcije statičke i spremljene su u datoteku *display.h*.

- bool example_lvgl_lock
  - pokušava rekurzivno zauzeti semafor u skladu s vremenskim ograničenjem
- void example_lvgl_unlock
  - rekurzivno oslobađa semafor i vraća ga ostalim dretvama
- static void example_lvgl_port_task

    - beskonačna petlja
    - stvaranje vremenske reference za LVGL
- static void example_increase_lvgl_tick
    - mjerenje koliko vremena je prošlo od početka izvođenja
- static void example_lvgl_touch_cb
    - callback funkcija za ekran osjetljiv na dodir
    - očitavanje pozicije pritiska prepoznavanje stanja
    - ne koristi se jer implementirani sustav nema ekran osjetljiv na dodir
- static void example_lvgl_port_update_callback
    - zamjena kordinata i rotacija prikaza na ekranu u ovisnosti i postavkama
    - rotacija prepoznatih koordinaza za ekran osjetljiv na dodir
- static void example_lvgl_flush_cb
    - pomicanje početne toke iscrtavanja prikaza
    - osvježavanje prikaza na ekranu
- static bool example_notify_lvgl_flush_ready
    - provjerava je li grafičko sučelje spremno za osvježavanje

Programski kod za rad s enkoderom se nalazi u datotekama *encoder.c* i *encoder.h*. U datoteci sa zaglavljem se nalazi funkcija *example_pcnt_on_reach* koja resetira brojač impulsa kad enkoder napravi predviđeni broj pomaka, a sve ostale potrebne funkcije su implementirane u datoteci *encoder.c*.

- void encoder1_read
    - očitavanje lijevog enkodera
- void encoder2_read
    - očitavanje desnog enkodera
- void encoder_init
    - inicijalizacija GPIO sklopovlja
    - inicijalizacija projača impulsa
    - inicijalizacija sklopovlja i programske podrške za rad s enkoderom

Rad s datotečnim sustavom littlefs je opisan u jednoj funkciji koja se nalazi u datoteci *littlefs.h*.

- void initialise_device_from_config_fille
  - podešavanje upravljačkog programa
  - traženje particije
  - čitanje konfiguracijske datoteke
  - pokretanje stvaranja korisničkog sučelja
  - izvlačenje podataka iz konfiguracije
  - stvaranje gumba definiranih konfiguracijom

Najkompliciraniji dio izgrađene programske podrške je čitanje obrada i stvaranje zahtjeva (poruka) za rad s periferijom. Ta funkcionalnost je opisana pomoću 3 datoteke: *datagram_generator.c*, *datagram_parser.c* i *main_request.parser.c*. Nalaze se u mapi *main/datagram_handlling*.
U datoteci *datagram_generator.c* se stvara JSON string na temelju podataka koji se šalju kao argumenti funkcija.

- const char* generate_datagram
  - završna funkcija za pakiranje metapodataka u poruku
- const char* generate_datagram_body
  - na podatke s uređaja dodaje podatke o perifernom uređaju I tipu poruke
- const char* generate_sensor_datagram
  - pakiranje senzorskog očitanja
- const char*generate_actuator_datagram
  - pakiranje stanja aktuatora
- const char* generate_user_datagram
  - pakiranje podataka za interakciju s korisnikom

Podaci se iz JSON stringa vade pomoću funkcija definiranih u datoteci *datagram_parser.c* korištenjem biblioteke cJSON.

- int parse_datagram
  - izvlači metapodatke iz parsiranog JSON stringa
- int parse_datagram_body
  - izvlači podatke iz tijela zahtjeva
- int parse_sensor_datagram
  - izvlači podatke iz senorskog očitanja
- int parse_actuator_datagram
  - izvlači podatke o stanju aktuatora
- int parse_user_datagram
  - izvlači podatke o interakciji s korisnikom

U datoteci *main_request_parser* se nalaze funkcije koje koordiniraju rad sa zahtjevima koji se preuzimaju s poslužitelja.

- void update_logical_clock
  - ažuriranje logičkog sata
- int parse_request
  - glavna metoda za obradu zahtjeva
  - poziva sve funkcije potrebne za uspješno provođenje obrade
  - prepoznaje vrste zahtjeva
  - prepoznaje vrstu periferije
  - poziva funkcije za očitavanje
  - pokreće stvaranje prikaza poruke korisniku
  - pokreće pakiranje podataka i pripremu slanja na poslužitelj
  - priprema za slanje na poslužitelj
- int read_data_from_sensor
  - očitava senzor
- int set_actuator_state

    - postavljanje aktuatora u novo stanje
- int get_actuator_state
    - čitanje stanja aktuatora
- static void mevent_cb
    - callback funkcija
    - prepoznavanje odabranog odgovora
- int display_message_to_user
    - korisniku prikazuje poruku
    - ako treba od korisnika preuzima odgovor
    - čekanje odgovora je vremenski ograničeno

Da bi sve to moglo funkcionirati, potrebno je definirati procese koji obavljaju potrebne zadatke. Definirani su proces za registraciju uređaja na poslužitelj i preuzimanje liste susjeda te proces za preuzimanje zaprimljenih zahtjeva iz liste čekanja na poslužitelju. Programski kod tih procesa se nalazi u datotekama *registration.c* i *request_getter.c* koje su spremljene u mapu *main/tasks*.

U datoteci *registration.c* se nalaze funkcije vezane za registraciju na poslužitelj i dohvaćanje liste susjeda.

- int register_end_device
    - sistematizira podatke u uređaju
    - šalje podatke za registraciju na poslužitelj
    - izvlači dodijeljeni indentifikator
- int get_neighbours
    - preuzima listu susjeda s poslužitelja
    - parsira listu i sprema podatke u podatkovni niz
- void find_neighbours
    - osnovna funkcija na temelju koje se stvara proces

    - beskonačna petlja

Datoteka *request_getter.c* pohranjuje tri funkcije koje su potrebne za pokretanje razmjene zahtjeva između uređaja.

- int getMessage
    - preuzima poruku s poslužitelja
- int getMessageCount
    - s poslužitelja povlači broj dostupnih poruka
- void get_messages()
    - glavna funkcija za proces koji se brine za razmjenu poruka
    - beskonačna petlja
    - povlačenje broja poruka
    - poziva povlačenje jedne poruke s poslužitelja i njezinu obradu onoliko puta koliko poruka je bilo dostupno u trenutku provjere njihovog broja

3.15 Zakjučak

Mikrokontroler ESP32 može izvoditi vrlo složen programski kod iako mu to nije normalan način primjene. Iako ima ograničenu procesorsku snagu bez obzira što posjeduje dvije procesorske jezgre, obavljanje relativno složenih funkcija koje su izvan područja umreženih senzora i aktuatora za koje se standardno koristi omogućava relativno velika količina ugrađene memorije (520 KB RAM i 4 MB ROM) uz usporeno izvođenje programa.

Tjeranje mikrokontrolera do krajnjih granica je dobar način za privikavanje na rad sa sustavima koji imaju vrlo stroga ograničenja koja treba poštovati ukoliko se želi napraviti sustav koji radi na predvidljiv i pouzdan način. SUAP je samo jedan od primjera koji pokazuje kako napraviti što više posla s ograničenim resursima.

Prebacivanje funkcionalnost čuvanja poruka u redu čekanja s mikrokontrolera na poslužitelj osigurava čuvanje poruke i slučaju ispada ciljanog čvora. Također, to značajno smanjuje potrebne resurse ma mikrokontroleru jer procesor ne treba u memoriji održavati programski stog za HTTP poslužitelj.

SUAP u svojoj biti nije strogo definiran sustav, nego je ideja kako napraviti fleksibilan sustav koristeći pripremljene programske elemente pomoću konfiguracije, bez potrebe za velikom količinom programiranja.

4. Literatura

1) Arduino Team, One board to rule them all: History of the Arduino UNO, 9.12.2021., *One board to rule them all: History of the Arduino UNO*,https://blog.arduino.cc/2021/12/09/one-board-to-rule-them-all-history-of-the-arduino-uno/ , 26.1.2025.
2) AVR microcontrollers, 30.7.2025., *AVR microcontrollers*, https://en.wikipedia.org/wiki/AVR_microcontrollers, 26.1.2025.
3) Espressif Systems Co., ESP32-WROOM-32 Datasheet, 13.2.2023, *ESP32-WROOM-32 Datasheet*, https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf, 26.1.2025.
4) Espressif Systems Co., Pulse Counter (PCNT), Pulse Counter (PCNT), *https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/pcnt.html*, 10.12.2024.
5) LVGL, Message box (lv_msgbox), 6.3.2024., *Message box (lv_msgbox)*, https://docs.lvgl.io/8.3/widgets/extra/msgbox.html, 12.1.2025.
6) LVGL, Button (lv_btn), 6.3.2024., *Button (lv_btn)*, *https://docs.lvgl.io/8.3/widgets/core/btn.html*, 12.1.2025.
7) littlefs-project, littlefs, 15.3.2025., *littlefs*, https://github.com/littlefs-project/littlefs, 29.5.2025.
8) Dave Gamble, cJSON 13.3.2024., *cJSON*, https://github.com/DaveGamble/cJSON, 15.1.2025.
9) Espressif Systems Co., ESP LCD ILI9341, *ESP LCD ILI9341*, https://components.espressif.com/components/espressif/esp_lcd_ili9341, 12.1.2025.
10) Espressif Systems Co., Component: Button, *Component: Button*, https://components.espressif.com/components/espressif/button/versions/4.1.3, 15.1.2025.
11) joltwallet, What is LittleFS?, 27.4.2025., *What is LittleFS?*,

https://components.espressif.com/components/joltwallet/littlefs/versions/1.0.0, 29.5.2025.

12) LVGL, Introduction, 6.3.2024, *Introduction*, https://docs.lvgl.io/8.3/intro/index.html, 12.1.2025.

13) LVGL, Button matrix (lv_btnmatrix), 6.3.2024, *Button matrix (lv_btnmatrix)*, https://docs.lvgl.io/8.3/widgets/core/btnmatrix.html, 20.4.2025.

14) LVGL, Slider (lv_slider), 6.3.2024, *Slider (lv_slider)*, *https://docs.lvgl.io/8.3/widgets/core/slider.html*, 20.5.2025.

5. Naslov

Ugradbeni računalni sustav za upravljanje uređajima na mreži

Embedded computer system for managing devices on a network

6. Sažetak

Ugradbeni računalni sustav za upravljanje uređajima na mreži je fleksibilni koncept upravljanja pomoću poruka. Zamišljen je kao rješenje problema beskrajnog broja uređaja ovisnih o pretplati koji nisu međusobno kompatibilni. U osnovi može prenositi bilo koje tipove podataka, ali je sustav trenutno ograničen na senzorska očitanja, stanja aktuatora i komunikaciju s korisnikom. Uz automatsko čitanje poruka, korisnik u konfiguracijskoj datoteci može definirati gumbe koji su povezani s unaprijed definiranim porukama kojima može sustavu narediti da se ponaša kako on želi. Poruke se preko mreže šalju u red čekanja na REST API poslužitelju s kojega ih uređaji periodički čitaju i na temelju obrade poduzimaju tražene akcije.

Embedded computer system for managing devices on a network if a flexible concept for managing using messages. It is solution for a problem of endless number of devices dependent on subscription incompatible with each other. It can carry basically and type of data but is currently limited to sensor readings, actuator states and communication with user. Besides automatic reading if messages, user can define buttons connected with predefined messages for making the system behave by user’s will using configuration file. Messages are transferred using the network to the REST API server with queue for each device from which devices then periodically pull the messages and perform required actions based on processed message.

7. Ključne riječi

mreža; upravljanje; konfiguracijska datoteka; korisničko sučelje; komunikacija; poruka; poslužitelj; uređaj

network; management; configuration file; user interface; communication; message; server; device

8. Prilozi

- izvorni kod poslužitelja
- izvorni kod krajnjeg čvora
- dijagrami pozivanja funkcija krajnjeg čvora
- upute za korištenje