Orkestracija raspodjeljenim web aplikacijama na uklopljenim sustavima

Udovičić, Lucian Tin

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

Orkestracija raspodijeljenim web aplikacijama na uklopljenim sustavima

Završni rad

Pula, rujan, 2024 godine

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

Lucian Tin Udovičić

Orkestracija raspodijeljenim web aplikacijama na uklopljenim sustavima

Završni rad

JMBAG: 0165073866, redoviti student
Studijski smjer: Sveučilišni preddiplomski studij informatika

Predmet: Web aplikacije
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: doc. dr. sc. Nikola Tanković

Pula, rujan, 2024 godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisan Lucian Tin Udovičić, kandidat za prvostupnika Informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljeni način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

Lucian Tin Udovičić

U Puli, 8. Rujan 2024. godine

IZJAVA

o korištenju autorskog djela

Ja, Lucian Tin Udovičić dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom „Orkestracija raspodijeljenim web aplikacijama na uklopljenim sustavima“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 8. Rujan 2024. godine

Potpis

Lucian Tin Udovičić

Abstract

This paper presents a software framework that enables software development independent of the underlying microcontroller architecture by applying modern programming techniques to devices with limited computational resources. In addition to running applications, the paper will describe how distributed web applications can be orchestrated on embedded systems using the environment described in this work. It will be shown how abstracting the underlying hardware functionality can lead to the development of universal applications that allow for easy switching between different hardware platforms. Software development for microcontrollers traditionally faces challenges such as limited processing power and memory capacity, requiring innovative solutions for optimal resource utilization. In this context, the development of a new environment for microcontrollers will explore how containerized applications and technologies like WebAssembly enable the development of universal and isolated applications that can be efficiently executed on various resource-constrained devices.

Keywords: WebAssembly, microcontroller, containers, virtualization, C/C++

Sažetak

Rad predstavlja softverski okvir koji omogućuje razvoj softvera neovisno o temeljnoj arhitekturi mikrokontrolera primjenom suvremenih načina programiranja na uređajima s ograničenim računalnim resursima. Osim izvođenja aplikacija, rad će opisati kako se omogućuje orkestriranje raspodijeljenih web aplikacija na uklopljenim sustavima uz pomoć radnog okruženja opisanog u ovom radu. Pokazati će se kako se apstrahiranjem funkcionalnosti temeljnog hardvera mogu razviti univerzalne aplikacije koje omogućuje jednostavno prebacivanje između različitih hardverskih platformi. Razvoj softvera za mikrokontrolere tradicionalno je suočen s izazovima kao što su ograničena procesorska snaga i memorijski kapacitet, zbog čega su potrebna inovativna rješenja za optimalno korištenje resursa. U tom kontekstu, razvojem novog radnog okruženja za mikrokontrolere, istražit će se kako aplikacije u spremnicima i tehnologije poput WebAssemblyja omogućuju razvoj univerzalnih i izoliranih aplikacija koje se mogu učinkovito izvoditi na različitim resursno ograničenim uređajima.

Ključne riječi: WebAssembly, mikrokontroler, spremnici, virtualizacija, C/C++

Sadržaj

1. Uvod

Aplikacije u spremnicima omogućile su programerima dosljedno pakiranje i implementaciju aplikacija u različitim okruženjima. Enkapsulacijom aplikacija i njihovih ovisnosti u prijenosni paket koji može pouzdano raditi bez obzira na osnovnu hardversku ili softversku infrastrukturu, ubrzan je razvoj softvera, implementacija softvera te je pojednostavljeno orkestriranje i upravljanje složenim sustavima [1]. Ova viša razina apstrakcije ne samo da omogućuje ponovnu upotrebljivost koda, već i smanjuje vrijeme razvoja i složenost aplikacija. Omogućuje programerima stvaranje fleksibilnijih i skalabilnijih rješenja koja se lako mogu prilagoditi promjenama hardvera bez značajnih izmjena koda. Široka primjena aplikacije u spremnicima ističe rastući trend prema razvoju neovisnom o platformi, a kako se tehnologija nastavlja razvijati, ovi pristupi postaju sve važniji za osiguravanje kompatibilnosti softvera i hardvera u različitim okruženjima.

Primjena spremnika u razvoju softvera donosi značajne prednosti, osobito u heterogenim sustavima gdje su u uporabi različite hardverske platforme. Korištenje spremnika omogućuje dosljedno radno okruženje, bez obzira na to radi li se o lokalnom razvoju, testiranju u oblaku ili implementaciji u produkciju. Ovo dosljedno okruženje smanjuje mogućnost grešaka povezanih s različitim konfiguracijama sustava i omogućuje lakšu replikaciju i rješavanje problema. Također, spremnici omogućuju skalabilnost sustava, jer se aplikacije mogu jednostavno klonirati i distribuirati na više instanci prema potrebi. Ova fleksibilnost posebno je korisna u situacijama kada se zahtijeva brz odgovor na promjene u opterećenju sustava, čime se omogućuje efikasnije upravljanje resursima.

Osim spremnika, postoji i pristup razvoja funkcija bez poslužitelja [2] koji predstavlja suvremeni pristup razvoju aplikacija gdje se pojedinačne funkcije izvršavaju kao autonomne jedinice u oblačnim okruženjima, bez potrebe za upravljanjem infrastrukturom poslužitelja. Ovaj model omogućuje skaliranje na zahtjev, gdje se resursi dodjeljuju samo kada je funkcija pozvana, čime se optimiziraju troškovi i učinkovitost. Funkcije bez poslužitelja popularne su zbog svoje sposobnosti da pojednostave razvoj i upravljanje aplikacijama te zbog podrške za različite programske

jezike i okvire. Jedna od glavnih prednosti funkcija bez poslužitelja je njihova sposobnost da se automatski prilagode promjenama u opterećenju sustava. Kada se funkcija pozove, sustav automatski alocira potrebne resurse, a nakon izvršenja funkcije, resursi se vraćaju u sustav. Ovo osigurava da sustav uvijek radi optimalno, bez nepotrebnog trošenja resursa. Funkcije bez poslužitelja također olakšavaju razvoj aplikacija, jer programeri ne moraju brinuti o upravljanju infrastrukturom, što im omogućuje da se usredotoče na razvoj same funkcionalnosti aplikacije.

Rad se fokusira na primjenu suvremenih načina razvoja web aplikacija na mikrokontrolerima koji, unatoč ograničenim hardverskim resursima, mogu podržati razvoj univerzalnih i izoliranih aplikacija te njihovu orkestraciju. Ovakav pristup može otvoriti nove mogućnosti za optimizaciju i inovacije u razvoju ugrađenih sustava. Mikrokontroleri su male, niskopotrošne računalne jedinice koje se obično nalaze unutar uređaja koji zahtijevaju automatizirano upravljanje. Integriraju procesor, memoriju i periferne uređaje na jednoj čipovnoj ploči, što im omogućuje obavljanje specifičnih zadataka unutar ugrađenih sustava. Zbog svoje kompaktne veličine i energetske učinkovitosti, mikrokontroleri se često koriste u raznim aplikacijama, od jednostavnih kućanskih aparata poput perilica rublja i termostata do složenih sustava u automobilskoj industriji, medicinskim uređajima i Internetu stvari (IoT). Njihova sposobnost da izvršavaju zadatke u stvarnom vremenu čini ih ključnim komponentama u dizajnu sustava koji zahtijevaju preciznu kontrolu i brze odgovore na promjene u okolini [3].

S obzirom na sve širi raspon primjena mikrokontrolera, njihova uloga u razvoju softverskih rješenja postaje sve važnija. U uređajima kao što su pametni kućanski aparati ili sustavi za automatizaciju zgrada, mikrokontroleri omogućuju lokalnu obradu podataka i donošenje odluka bez potrebe za stalnom vezom s centraliziranim poslužiteljima. To smanjuje latenciju i poboljšava učinkovitost sustava, omogućujući uređajima da funkcioniraju neovisno čak i u slučajevima prekida mrežne veze. U automobilskoj industriji, mikrokontroleri su srce sustava kao što su ABS, zračni jastuci, sustavi za kontrolu stabilnosti i mnogi drugi. U IoT okruženjima, mikrokontroleri omogućuju povezivanje i upravljanje velikim brojem senzora i aktuatora, stvarajući

mreže pametnih uređaja koji mogu međusobno komunicirati i prilagođavati se promjenama u stvarnom vremenu.

Razvoj softverskih okvira i alata za razvoj softvera za mikrokontrolere u velikoj se mjeri konsolidirao u nekoliko popularnih softverskih okvira. Ti okviri omogućuju programerima korištenje istih alata za razvoj aplikacija za različite mikrokontrolere različitih proizvođača. Najbolji primjer za to je Arduino okvir [4], koji podržava veliki broj uređaja različitih proizvođača. Dok sami proizvođači nude svoja integrirana razvojna okruženja i knjižnice specifične za svoje uređaje, Arduino okvir omogućuje brži razvoj softvera na različitim uređajima, posebno kada je programer već upoznat s okvirom.

Povećana potreba za optimiziranim i fleksibilnim rješenjima u području interneta stvari i ugrađenih sustava zahtijeva pristupe koji mogu iskoristiti ograničene resurse mikrokontrolera bez kompromitiranja performansi ili sigurnosti. Tradicionalne metode razvoja softvera za ove uređaje često zahtijevaju pisanje specifičnog koda za svaki mikrokontroler, što dovodi do fragmentacije i otežava skalabilnost i održavanje sustava. Zbog toga, prijenosni i izolirani softverski okviri koji omogućuju razvoj aplikacija neovisnih o platformi postaju ključni za ubrzavanje razvoja i implementacije. Rad uvodi inovativni okvir koji kombinira prednosti spremnika i tehnologija poput WebAssemblyja [5] za stvaranje prilagodljivih, sigurnih i učinkovitih rješenja koja mogu raditi na različitim hardverskim platformama unatoč njihovim ograničenjima. Ovakav pristup može značajno poboljšati razvojne procese i omogućiti brže i pouzdanije prilagodbe u dinamičnim okruženjima kao što su pametni gradovi, industrijska automatizacija i druge IoT aplikacije.

Rad je podijeljen u pet glavnih poglavlja, od kojih svatko obrađuje specifičan aspekt teme rada. Prvo poglavlje pruža pregled i motivaciju za odabir teme te se objašnjava važnost spremnika i WebAssembly tehnologije u razvoju. Teorijska osnova, izbor alata i hardvera obrađuje temeljne teorijske koncepte potrebne za razumijevanje teme, uključujući pregled suvremenih tehnologija spremnika i WebAssemblija. Također se raspravlja o tržišnoj ponudi mikrokontrolera i alatima potrebnim za razvoj softverskog okvira. U trećem poglavlju se detaljno opisuje dizajn i implementacija radnog okvira koji omogućava ciljeve rada. Obrađuje se arhitektura okvira, moduli unutar okvira, način na koji se WebAssembly koristi za izradu aplikacija u spremnicima i

funkcionalnosti radnog okvira koje omogućuju orkestraciju istih. Izazovi, budući smjerovi i poboljšanja identificira glavne izazove s kojima se susrelo tijekom razvoja okvira, te se predlažu potencijalni smjerovi za buduća istraživanja i unaprjeđenja.

	Aplikacije u spremnicima	Funkcije bez poslužitelja	Spremnici u mikrokontrolerima
Prednosti	Dosljedno radno okruženje u različitim fazama razvoja i implementacije. Smanjuje mogućnost grešaka povezanih s različitim konfiguracijama sustava. Omogućuje skalabilnost sustava i efikasnije upravljanje resursima.	Automatsko skaliranje na zahtjev s alokacijom resursa samo prilikom poziva funkcije. Optimizacija troškova i učinkovitosti. Pojednostavljuje razvoj i upravljanje aplikacijama bez brige o infrastrukturi.	Omogućuje lokalnu obradu podataka i donošenje odluka bez stalne veze s poslužiteljima. Smanjuje latenciju i poboljšava učinkovitost sustava. Fleksibilnost u prilagodbi hardverskim promjenama bez značajnih izmjena koda.
Nedostaci	Može zahtijevati više resursa u usporedbi s izravnim izvršavanjem na hardveru. Kompleksnost u upravljanju velikim brojem spremnika u složenim sustavima.	Ograničenja u trajanju izvršavanja funkcija. Ovisnost o pružatelju usluga u oblaku i potencijalni problemi s latencijom.	Ograničeni hardverski resursi mikrokontrolera predstavljaju izazov. Potreba za optimizacijom aplikacija za rad u ograničenim okruženjima.
Skalabilnost	Lako kloniranje i distribucija aplikacija.	Automatsko skaliranje funkcija prema potrebi.	Ograničena hardverom mikrokontrolera.
Primjene	Heterogeni sustavi s različitim hardverskim platformama.	Aplikacije koje zahtijevaju skalabilnost i optimalnu upotrebu resursa.	IoT uređaji, pametni kućanski aparati, industrijska automatizacija.

Tablica 1 - Usporedba pristupa

2. Teorijska osnova, izbor alata i hardvera

2.1. Teorijska osnova i tržišna ponuda

Ovo poglavlje pruža pregled ključnih koncepata i tehnologija relevantnih za razvoj radnog okvira predstavljenog u ovom radu. Detaljno se razmatraju postojeća istraživanja vezana uz virtualizaciju na mikrokontrolerima, s posebnim naglaskom na primjenu WebAssembly tehnologije u okruženjima s ograničenim resursima. Analiziraju se znanstveni radovi koji pružaju uvid u izazove i moguća rješenja za izvršavanje koda u virtualiziranim okruženjima na ovim uređajima.

Također se istražuje trenutna tržišna ponuda mikrokontrolera koji su najprikladniji za implementaciju u IoT i ugrađenim sustavima te suvremene načine upravljanja i izrade spremnika u računalnim sustavima. Analiziraju se njihove funkcionalnosti te metode koje su poslužile kao inspiracija za razvoj prilagođenih rješenja na mikrokontrolerima.

2.1.1. Teorijska osnova

U istraživanju izvedivosti ciljeva rada, proučeni su radovi sličnih tema i problema kako bi se pronašla najbolja rješenja i izbjegle poteškoće na koje su drugi već naišli. Rad [6] o virtualizaciji, temeljenoj na WebAssemblyju, od autora Gregor Peacha i drugih, pruža uvid u postojeća rješenja srodna ovom radu. Znanstveni rad uvodi eWASM, mehanizam za virtualizaciju temeljen na WebAssemblyju, koji je posebno optimiziran za mikrokontrolere i rješava izazove integriteta memorije i kontrolnog toka u okruženjima s ograničenim resursima. Autori pokazuju da eWASM može izvršavati kod u virtualnom okruženju uz performansno opterećenje unutar 40% u odnosu na izvorni C kod.

Autori Fredrika Erikssona i Sebastiana Grunditza istražuju izvedivost korištenja WebAssembly (Wasm) okruženja na IoT uređajima kao alternativu tradicionalnim Docker rješenjima temeljenim na spremnicima [7]. Autori provode niz testova performansi uspoređujući WebAssembly okruženja s Dockerom, s posebnim naglaskom na brzinu, potrošnju memorije i vrijeme pokretanja na Raspberry Piju. Ovo istraživanje je vrlo relevantno za ovaj rad jer ističe potencijalne prednosti

WebAssemblyja za IoT uređaje, posebno u smislu smanjenog memorijskog otiska i bržeg izvođenja za kratke, intenzivne zadatke, koji su česti u IoT okruženjima.

2.1.2. Tržišna ponuda mikrokontrolera

Na tržištu mikrokontrolera postoji široka ponuda uređaja koji se razlikuju po svojim specifikacijama, performansama i podršci za razvojne alate, omogućujući programerima i inženjerima izbor optimalnog rješenja za njihove specifične potrebe. Najpopularniji mikrokontroleri za primjenu u IoT okruženjima i ugrađenim sustavima, poput *ESP32* [8], *STM32* [9] i *nRF* [10] serije, nude odličan balans između cijene, snage i energetske učinkovitosti, što ih čini idealnim za širok raspon aplikacija. *ESP32*, na primjer, nudi integrirano Wi-Fi i Bluetooth sučelje te podržava razvojne okvire kao što su Arduino [4], FreeRTOS [11] i ESP-IDF [12], što ga čini izuzetno prilagodljivim za različite scenarije primjene. Posebno se ističe *ESP32-C3* model, koji je baziran na RISC-V arhitekturi i predstavlja ekonomičniju verziju *ESP32* mikrokontrolera, zadržavajući ključne značajke kao što su Wi-Fi i Bluetooth povezivost, ali s naglaskom na poboljšanu sigurnost i manju potrošnju energije. Ovaj mikrokontroler omogućuje jednostavnu implementaciju u raznim IoT rješenjima, uključujući one koja zahtijevaju visoku sigurnost, kao što su pametni uređaji u kućanstvu ili industrijski nadzorni sustavi. Njegova kompatibilnost s postojećim *ESP* ekosustavom i alatom ESP-IDF dodatno olakšava tranziciju i razvoj novih aplikacija.

STM32 serija, koju proizvodi STMicroelectronics, ističe se širokim rasponom performansi, od jednostavnih, energetski učinkovitih modela do moćnih mikrokontrolera s naprednim sigurnosnim značajkama, kao što su hardverska enkripcija i integracija s vanjskim sigurnosnim modulima. Ova serija pruža izuzetnu fleksibilnost i mogućnost prilagodbe specifičnim potrebama projekta, čineći je pogodnom za sve, od jednostavnih aplikacija do kompleksnih sustava u industrijskoj automatizaciji i medicinskoj tehnologiji. Osim toga, *STM32* mikrokontroleri podržavaju niz razvojnih alata, uključujući STM32Cube [13], koji olakšava konfiguraciju i razvoj softvera.

nRF serija mikrokontrolera, koju proizvodi Nordic Semiconductor, poznata je po svojoj niskoj potrošnji energije i snažnoj podršci za Bluetooth Low Energy protokol. Ovi

mikrokontroleri idealni su za aplikacije koje zahtijevaju dugotrajnu upotrebu uz minimalnu potrošnju baterije, kao što su nosivi uređaji, pametni senzori i bežični zdravstveni uređaji. *nRF* serija također podržava dodatne komunikacijske protokole poput ANT i Thread, što omogućuje jednostavnu integraciju u složene IoT mreže.

Mikrokontroleri poput Raspberry Pi Pico [14], koji koristi RP2040 čip, nudi fleksibilnost zahvaljujući dvostrukim ARM Cortex-M0+ jezgrama i podršci za razne razvojne okvire i jezike, uključujući C/C++, MicroPython i Rust. Ova platforma omogućuje programerima da koriste najprikladniji jezik i razvojni okvir za svoj projekt, pružajući dodatnu slobodu u dizajnu i implementaciji.

2.1.3. Suvremeni načini upravljanja i izrade spremnika

Popularni alati poput Dockera [15], Podmana [16] i Kubernetesa [17] razvijeni su za aplikacije koje se pokreću na računalima i serverima, pružajući učinkovite metode za izolaciju, distribuciju i upravljanje aplikacijama u različitim okruženjima. Docker, jedan od najraširenijih alata, omogućuje programerima da pakiraju aplikacije s njihovim ovisnostima u lagane, prijenosne spremnike, koji se mogu dosljedno pokretati u bilo kojem okruženju. Omogućuje se jednostavnija replikacija razvojnih okruženja i olakšava se prijenos aplikacija između različitih faza razvoja, od lokalnog sustava do produkcijskog okruženja.

Podman, kao alternativa Dockeru, nudi slične mogućnosti, ali s dodatnim naglaskom na sigurnost, omogućujući rad bez centraliziranog demonskog procesa, što poboljšava sigurnost i kontrolu nad aplikacijama. Ova značajka čini Podman atraktivnim izborom za organizacije koje traže visoku razinu kontrole i sigurnosti prilikom implementacije aplikacija u spremnicima. Kubernetes, s druge strane, služi za orkestraciju spremnika u složenim sustavima, omogućujući automatizirano upravljanje skaliranjem, raspoređivanjem i održavanjem aplikacija u klasterima.

Navedeni alati omogućuju visok stupanj fleksibilnosti i skalabilnosti, omogućujući organizacijama da brzo prilagode svoje aplikacije promjenama u opterećenju ili infrastrukturi. Kubernetes, posebno, omogućuje jednostavno upravljanje velikim brojem spremnika, osiguravajući visoku dostupnost i otpornost sustava. Ovakvi alati postali su standard u modernom razvoju softvera, omogućujući brže, pouzdanije i

učinkovitije procese razvoja, testiranja i implementacije aplikacija u različitim okruženjima. Funkcionalnosti navedenih alata su korištene kao glavne ideje i ciljevi u razvoju radnog okvira rada, za mikrokontrolere.

2.2. Pregled dostupnih razvojnih alata

2.2.1. Biblioteke i alati za izvršavanje WebAssembly koda

WebAssembly [5] se sve više koristi kao način pakiranja i izvođenja aplikacija na mikrokontrolerima zbog svoje male veličine, brzine i mogućnosti izolacije. WebAssembly omogućuje da aplikacije budu kompaktne, brze i sigurne, što ih čini idealnim za primjenu u okruženjima s ograničenim resursima, poput mikrokontrolera. Trenutno dostupni WebAssembly interpreteri pokazuju raznolikost alata koji se fokusiraju na različite potrebe, od aplikacija na serveru do ugradbenih sustava s ograničenim resursima. Mnogi od ovih interpretera prilagođeni su specifičnim scenarijima korištenja, poput rubnog računarstva, IoT okruženja, te su optimizirani za visoke performanse i minimalnu potrošnju resursa. Wasmtime [18] i WasmEdge [19] ističu se kao lagani i brzi interpreteri, posebno optimizirani za rubno računarstvo i IoT okruženja, pružajući podršku za JIT i *ahead-of-time* AOT kompajliranje. WAVM [20] i Wasmer [21] nude visoke performanse s naprednim mogućnostima kompajliranja, čineći ih pogodnima za scenarije gdje je brzina ključna, ali također mogu raditi u resursno ograničenim okruženjima. Wasm3 [22], wasm-micro-runtime [23] i wasmi [24] omogućuju izvođenje WebAssembly modula na mikroprocesorima i ugradbenim sustavima gdje je važna efikasnost korištenja resursa.

Odabir interpretera za mikrokontrolere oslanja se dijelom na rad u kojem autori Stefan Wallentowitz, Sebastian Kersting i Dan Mihai Dumitriu istražuju potencijalnu upotrebu WebAssemblyja u integriranim sustavima [25]. U svom istraživanju, autori su se odlučili za wasm-micro-runtime interpreter zbog njegovih naprednih značajki, uključujući mogućnost AOT interpretacije, koja omogućuje interpretaciju WebAssembly koda unaprijed, što rezultira bržim izvođenjem aplikacija. Njihovi testovi pokazali su da ovaj interpreter može doseći do 50% performansi izvornog koda, što je značajno poboljšanje u kontekstu resursno ograničenih uređaja.

Za potrebe razvoja radnog okvira odlučeno je koristiti Wasm3 okruženje za izvođenje WebAssembly koda. Ova odluka je donesena na temelju nekoliko ključnih faktora. Wasm3 se ističe svojom izuzetno malom memorijskom potrošnjom, što ga čini idealnim za mikrokontrolere s vrlo ograničenim resursima. Wasm3 također pruža jednostavniji postupak implementacije, jednom dijelom zbog autorovog prijašnjeg poznavanja a jednim dijelom zbog jednostavnog dizajna interpretera, što je omogućilo bržu integraciju i razvoj aplikacija bez potrebe za dodatnim složenostima koje donose napredniji interpreteri. Iako Wasm3 možda ne doseže iste razine performansi kao wasm-micro-runtime u AOT načinu, njegova prednost u pogledu fleksibilnosti, stabilnosti i niske potrošnje resursa nadmašuje te nedostatke u kontekstu ciljanih aplikacija. Tijekom razvoja okvira, Wasm3 se pokazao kao dovoljno stabilan i pouzdan, omogućujući neometano izvođenje različitih zadataka unutar ograničenih hardverskih kapaciteta mikrokontrolera.

2.2.2. Dostupni alati za razvoj na *ESP32* platformi

Dostupni alati na tržištu za razvoj i testiranje hardverskih platformi obuhvaćaju širok spektar rješenja koja omogućuju programerima i inženjerima da učinkovito razvijaju, testiraju i implementiraju softverska rješenja na različitim vrstama mikrokontrolera i srodnih uređaja. Na tržištu su dostupni razvojni alati poput Arduino integriranog razvojnog okruženja, koji pruža jednostavno i pristupačno sučelje za brzi razvoj i testiranje, te je osobito popularan među hobistima i edukatorima. S druge strane, napredniji alati poput Espressifovog ESP-IDFa [12] ili PlatformIOa [26] pružaju duboku integraciju s mikrokontrolerima kao što su *ESP32* i omogućuju naprednije opcije za prilagodbu, debagiranje i optimizaciju koda.

Postoje i specijalizirani alati za simulaciju i emulaciju koji omogućuju testiranje aplikacija u virtualnim okruženjima prije njihove implementacije na fizički hardver. Alati poput QEMUa [27] ili Proteusa [28] omogućuju inženjerima da precizno simuliraju rad mikrokontrolera i perifernih uređaja, što ubrzava razvojni proces i smanjuje rizik od grešaka u konačnoj implementaciji. Uz to, alati za analizu i dijagnostiku mrežnog prometa, poput Wiresharka [29], omogućuju dubinsku analizu i optimizaciju komunikacijskih protokola, što je ključno za razvoj mrežnih aplikacija na resursno ograničenim uređajima. Na tržištu su također dostupne razne biblioteke i softverski

okviri koji olakšavaju razvoj specifičnih funkcionalnosti, kao što su upravljanje senzorima, bežična komunikacija ili sigurnosni protokoli. Ovi alati često dolaze s opsežnom dokumentacijom i podrškom zajednice, što dodatno olakšava razvoj i implementaciju složenih sustava. Sve ove opcije omogućuju programerima i inženjerima da odaberu alate koji najbolje odgovaraju njihovim potrebama, bilo da se radi o brzom prototipiranju ili razvoju visokokvalitetnih komercijalnih rješenja.

Odluka za odabir *ESP32* i *ESP32-C3* [8] mikrokontrolera donesena je zbog njihovih izvanrednih karakteristika koje ih čine izuzetno pogodnima za razvoj i implementaciju u raznim ugrađenim sustavima. Prvo, oba mikrokontrolera nude dobru ravnotežu između cijene i performansi, pružajući značajne računalne kapacitete uz nisku potrošnju energije, što ih čini idealnim za aplikacije gdje je energetska učinkovitost ključna. Drugo, *ESP32* sa svojim dual-core Tensilica Xtensa LX6 jezgrama omogućuje istovremeno izvođenje višestrukih zadataka, dok *ESP32-C3*, temeljen na RISC-V arhitekturi, nudi jednostavniju i sigurniju platformu s integriranim sigurnosnim značajkama poput hardverskog kriptografskog modula. Odabrani mikrokontroleri imaju integrirane Wi-Fi i Bluetooth module, što omogućuje jednostavnu bežičnu komunikaciju ključnu za mnoge IoT i mrežne aplikacije te time i web aplikacije. S obzirom na to da je cilj ovog rada bio istražiti i razviti radni okvir koji može podržati i orkestrirati raznolike web aplikacije u mrežno povezanom okruženju, integracija ovih komunikacijskih modula bila je važan faktor u odabiru ovih mikrokontrolera. Konačno, velika zajednica korisnika i opsežna podrška u obliku dokumentacije, biblioteka i razvojnih alata za *ESP32* seriju dodatno su potaknuli izbor ovih mikrokontrolera, omogućujući bržu implementaciju i rješavanje problema tijekom razvoja.

Drugi razlog za korištenje *ESP32* i *ESP32-C3* mikrokontrolera je bila dostupnost robusnih razvojnih okruženja, poput ESP-IDF-a i Visual Studio Codea. ESP-IDF, službeni razvojni okvir tvrtke Espressif, pruža opsežan skup alata i biblioteka koje olakšavaju razvoj, debagiranje i optimizaciju softvera specifično za *ESP32* i *ESP32-C3* mikrokontrolere. Ovaj okvir omogućuje kontrolu nad svim aspektima rada mikrokontrolera, uključujući upravljanje memorijom, komunikacijskim protokolima i energetskom učinkovitošću, što ga čini idealnim za korištenje za razvoj radnog okruženja i web aplikacija za mikrokontrolere.

Za integrirano razvojno okruženje korišten je Visual Studio Code zbog fleksibilnosti, jednostavnosti korištenja te bogatim ekosustavom proširenja. Integracija ESP-IDF-a u VS Code omogućila je korištenje alate za uređivanje koda, upravljanje projektima i debagiranje unutar poznatog sučelja što je dodatno ubrzalo razvoj. Kombinacija ESP-IDFa i Visual Studio Codea pružila je odlično razvojno okruženje koje je omogućilo brzu iteraciju, testiranje i implementaciju softverskih rješenja na mikrokontrolerima. Također je pomoglo što Espressif pruža detaljnu dokumentaciju instalacije i postavljanja okruženja [12] te je cijeli proces instalacije i postavljanja bio veoma brz.

3. Okvir za orkestraciju raspodijeljenih web aplikacija na uklopljenim sustavima

3.1. Dizajn radnog okvira

Radni okvir je dizajniran tako da se može što lakše prilagoditi različitim hardverskim arhitekturama i proizvođačima mikrokontrolera uz minimalnu ovisnost o osnovnom hardveru i razvojnom okruženju pojedinog proizvođača no dostupnost hardverskih značajki mikrokontrolera utječe na dostupne funkcionalnosti radnog okvira. Mikrokontroleri koji nemaju mrežno sučelje nisu u mogućnosti koristiti određene module radnog okvira, ali kao mrežni posrednik mogu se koristiti druga sučelja koja promet s nekompatibilnog sučelja konvertiraju u format pojedinog komunikacijskog modula radnog okvira. Ovo omogućuje veću fleksibilnost u primjeni radnog okvira i olakšava rad u heterogenim mrežama mikrokontrolera. Na taj način, različiti uređaji mogu međusobno komunicirati i surađivati, bez obzira na njihova ograničenja i mogućnosti povezivanja.

Dok je u Dockeru spremnik izolirano okruženje koje sadrži aplikaciju i sve njezine ovisnosti omogućujući njezino dosljedno pokretanje na bilo kojem sustavu, u kontekstu ovog radnog okvira spremnik je WebAssembly modul koji sadrži memorijske blokove i funkcije koje se povezuju sa radnim okvirom kao što Docker spremnici dijele kernel temeljnog operacijskog sustava za sustavne pozive i niskorazinske funkcije. Okvir koristi WebAssembly kako bi omogućio izolirano izvođenje aplikacija, čime se osigurava sigurnost i smanjuje potreba za specifičnim prilagodbama koda za svaki uređaj. Implementiran je sloj hardverske apstrakcije koji pruža standardizirano sučelje za interakciju s različitim hardverskim resursima, što omogućuje veću prenosivost i fleksibilnost aplikacija. Radni okvir također uključuje mrežni sloj koji podržava komunikaciju među uređajima čime se olakšava distribucija aplikacija i komunikacija s ostalim uređajima, mrežnim servisima i web preglednicima. Cijeli sustav je optimiziran za rad u okruženjima s ograničenim resursima, omogućujući učinkovito upravljanje memorijom i potrošnjom energije.

Okvir omogućuje izvršavanje zadataka slanjem HTTP zahtjeva koji sadrže JSON poruke koje opisuju željene radnje i njihove parametre. Ključno je da okvir može istovremeno obraditi ove zahtjeve tako da tekuće operacije nisu blokirane i da čekanje na završetak operacija ne ometa funkcionalnost sustava. To je posebno važno za zadatke koji ovise o događajima iz stvarnog svijeta, a ne o računalnim procesima. Primjer bi bile operacije poput mjerenja senzora ili pomicanja motora u određene položaje, te ostale operacije koje mogu blokirati izvršavanje drugih zadataka ali obično ne zahtijevaju značajnu računalnu snagu.

Temelj projekta je FreeRTOS, okvir za operacijski sustav u stvarnom vremenu koji omogućuje istovremeno izvršavanje zadataka na jednojezgrenim ili višejezgrenim mikrokontrolerima. Primarni programski jezici korišteni u projektu su kombinacija C i C++ jezika, pri čemu je većina komponenti napisana u C++ jeziku radi lakše komponentizacije, dok su nižerazinske funkcije, koje komuniciraju s hardverom ili zahtijevaju malo dublju optimizaciju memorije, napisane u C jeziku. Na vrhu FreeRTOSa razvijen je TCP poslužitelj koji omogućuje distribuciju više istovremenih zahtjeva FreeRTOS zadacima koji se istovremeno izvode na mikrokontroleru. Implementiran je jednostavan i minimalni HTTP parser od 200 linija koda za obradu i generiranje HTTP zahtjeva i odgovora, dok je biblioteka ArduinoJSON [30] korištena za parsiranje i generiranje JSON poruka. Kako bi se dodatno poboljšale mogućnosti okvira, integrirane su značajke za obradu podataka u stvarnom vremenu, optimiziran je memorijski otisak radi boljih performansi, te je osigurano robusno rukovanje greškama kako bi se održala stabilnost sustava pod različitim opterećenjima.

U modulima mrežne komunikacije, memorijska optimizacija se svela na pažljivo upravljanje resursima kako bi se smanjila potrošnja memorije, što uključuje smanjenje veličine međuspremnika, optimizaciju algoritama za rukovanje podacima te korištenje učinkovitijih metoda za serijalizaciju i deserijalizaciju podataka. Ove optimizacije također uključuju uklanjanje nepotrebnih kopija podataka i smanjenje redundancije, čime se omogućuje brža i učinkovitija obrada podataka uz minimalnu potrošnju memorije. Korištenje statičkih alokacija umjesto dinamičkih, gdje je to bilo moguće, smanjena je fragmentaciju memorije čime su se poboljšale performanse mrežnog modula, omogućujući bržu komunikaciju i bolju upotrebu dostupnih resursa. Javno

dostupna biblioteka korištena je za izvršavanje WebAssembly koda na mikrokontroleru. Wasm3 [22] je posebno dizajniran za pokretanje WebAssembly [5] koda na širokom rasponu platformi, uključujući mikrokontrolere. Iako nije brz kao izvorni kod, pruža sigurno virtualno okruženje u kojem se programi izvršavaju bez izravnog pristupa temeljnom izvršnom okruženju. Njegova mala veličina i minimalni memorijski otisak omogućuju učinkovito izvođenje na platformama od web preglednika do 8-bitnih mikrokontrolera. Korištenjem Wasm3ja umjesto naprednijih interpretera olakšalo je dodatnu optimizaciju određenih poziva funkcija čime se smanjila potrošnja memorije, zamjenjujući povećano vrijeme izvođenja manjim memorijskim otiskom. Umjesto održavanja "vrućeg" spremnika spremnog za pokretanje u svakom trenutku, ovaj okvir koristi pristup "hladnog pokretanja", gdje se sva memorija virtualnog okruženja alocira tek na zahtjev za izvršavanje. Ovaj pristup, iako stvara određenu latenciju, značajno smanjuje ukupnu potrošnju memorije što je bilo bitnije zbog ograničenih resursa mikrokontrolera.

Projekt, ovisnosti, postupkom pokretanja i izvorni kod dostupan je na GitHubu [31].

3.2. Moduli dostupni unutar radnog okvira

Primjenom suvremenih principa dizajna sustava, radni okvir je podijeljen na više modula, čime se postiže odvajanje odgovornosti koje potiče bolju organizaciju koda i lakše održavanje sustava. Modularni pristup omogućuje jednostavniju prilagodbu i proširenje funkcionalnosti jer se svaki modul može neovisno razvijati, testirati i zamijeniti bez utjecaja na druge dijelove sustava. Moduli kao klase odgovorne za pojedini dio sustava, povezuju se s međuklasama koje izlažu sučelje pojedinog modula. Rezultat je jednostavna prilagodba HTTP serverskog modula s modulom drugog komunikacijskog sustava osim TCP server modula. Za korištenje drugog komunikacijskog sučelja potrebana je implementacija modula komunikacije na novom sučelju koji koristi istu međuklasu. Razlog za korištenje međuklasa kao ljepila modula je također što karakteristike pojedinih uređaja mogu zahtijevati različitu implementaciju i inicijalizaciju modula. Najbolji primjer je postavljanje parametara modulacija širine impulsa za upravljanje servo motorima, *ESP32* sadrži hardverski generatore takta koji ne postoje u *ESP32-C3* mikrokontroleru te ih je potrebno drugačije inicijalizirati.

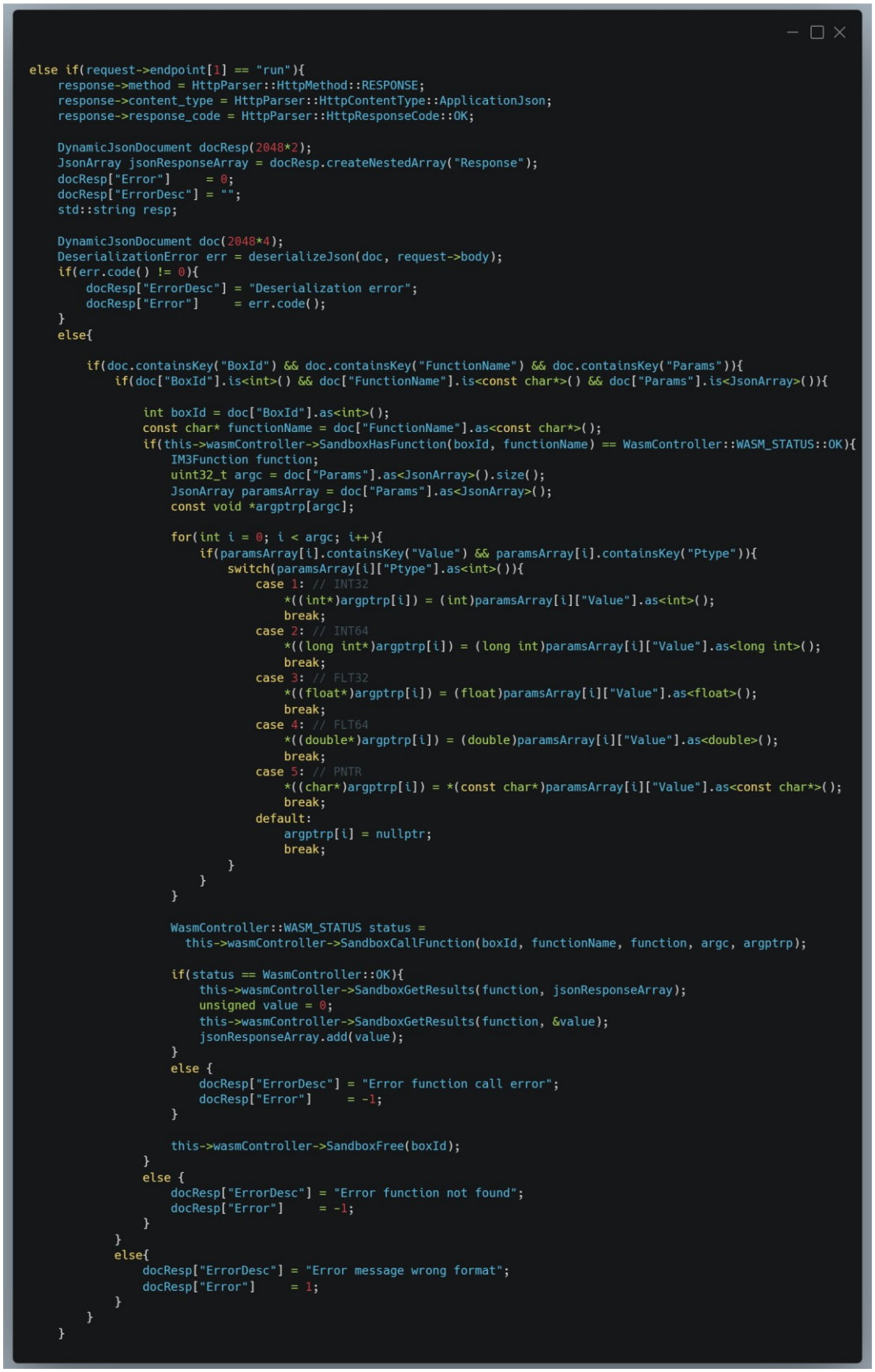
Modul za istovremeno obrađivanje zahtjeva služi za povezivanje ostalih modula sa komunikacijskim modulom i za izvršavanje funkcija modula ovisno o primljenom zahtjevu. Također se koristi za formiranje odgovora na zahtjeve i kontinuirano posluživanje traženih podataka u realnom vremenu. Povezivanjem komunikacijskog modula s ovim modulom pomoću među klase, omogućuje se korištenje TCP ili UDP utičnica za prihvaćanje naredbi.

Modul virtualizacije i izvršavanja spremnika upravlja virtualizacijom WebAssembly spremnika, orkestracijom spremnika po vanjskim zahtjevima za izvršavanje funkcija i samim izvršavanjem funkcija unutar virtualnog Wasm3 okruženja. Ovaj modul omogućuje izolaciju aplikacija i njihovih resursa, upravljanje radnim opterećenjima kroz koordinaciju više spremnika te kontrolu nad izvršavanjem specifičnih funkcija unutar tih spremnika.

Modul za upravljanje nižom hardverskom razinom mikrokontrolera služi za upravljanje i dohvat podataka o značajkama mikrokontrolera poput količine slobodne radne memorije, temperature jezgri i opterećenosti jezgri. Povezivanjem ovog modula s modulom za obrađivanje zahtjeva, te u konačnici s komunikacijskim modulom, može se dohvaćati i pratiti hardversko stanje mikrokontrolera u realnom vremenu.

```
class NetworkController {
 private:
  // INTERFACES
  esp_netif_t* ap_netif;
  esp_netif_t* sta_netif;

  std::map<int, wifi_ap_conf_t> ApConfigurations;
  std::map<int, wifi_st_conf_t> StConfigurations;

  std::map<int, TcpHandlerType_t> TcpConfigurationTypes;
  std::map<int, void*> TcpConfigurations;
  std::map<int, tcp_raw_task_handler_t*> TcpRawConfigurations;
  std::map<int, int> TcpActiveHandlerIdToPortNumberMap;

  int startingPortNum = 10000;
  int rollingPortNum = 0;

  int current_ap_config_id = -1;
  int current_st_config_id = -1;
  wifi_mode_t wifi_mode = WIFI_MODE_NULL;
  wifi_config_t current_configuration;

  esp_err_t WifiRefresh();

  static void TcpRawTaskHandlerBase(void* pvParameters);

  esp_err_t TcpInstantiateRawHandlerForHandler(int handler_id);
  esp_err_t TcpInstantiateRawHandlerForHandler(int handler_id, int port);

  int TcpAllocatePortNumber();
  int TcpRegisterPortNumberAllocation(int port);
  void TcpDellocatePortNumber(int port);

 public:
  Preferences_::PreferencesController* preferencesController = NULL;
  NetworkController(Preferences_::PreferencesController* preferencesController);

  int WifiScan();
  esp_err_t WifiStart();
  esp_err_t WifiStop();

  esp_err_t ApStart(int config_id);
  esp_err_t ApStop();

  int ApAddConfiguration(wifi_ap_conf_t config);
  void ApRemoveConfiguration(int config_id);
  wifi_ap_conf_t ApGetConfiguration(int config_id);

  esp_err_t StStart(int config_id);
  esp_err_t StStop();
  int StAddConfiguration(wifi_st_conf_t config);
  void StRemoveConfiguration(int config_id);
  wifi_st_conf_t StGetConfiguration(int config_id);

  static void wifi_event_handler(void* arg, esp_event_base_t event_base,
                                 int32_t event_id, void* event_data);
  static void event_handler(void* arg, esp_event_base_t event_base,
                            int32_t event_id, void* event_data);

  esp_err_t TcpAddHandler(void* handler, TcpHandlerType_t type);
  esp_err_t TcpRemoveHandler(int handler_id);

  int TcpBindHandlerToPort(int handler_id);
  bool TcpBindHandlerToPort(int handler_id, int port);
  esp_err_t TcpKillHandler(int handler_id);
  int TcpGetPortForHandlerId(int handler_id);

  esp_err_t TcpBindRawHandlerToPort(tcp_raw_task_handler_t* raw_task);
  esp_err_t TcpKillRawHandlerForPort(int port);

  ~NetworkController();
};
```

Slika 1 - Isječak koda zaglavlja mrežnog modula

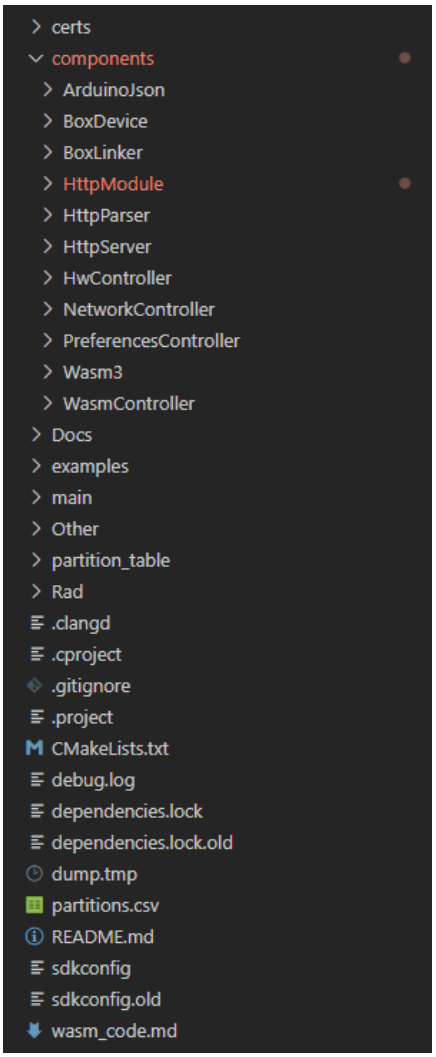


```
else if(request->endpoint[1] == "run"){
    response->method = HttpParser::HttpMethod::RESPONSE;
    response->content_type = HttpParser::HttpContentType::ApplicationJson;
    response->response_code = HttpParser::HttpResponseCode::OK;

    DynamicJsonDocument docResp(2048*2);
    JsonArray jsonResponseArray = docResp.createNestedArray("Response");
    docResp["Error"]     = 0;
    docResp["ErrorDesc"] = "";
    std::string resp;

    DynamicJsonDocument doc(2048*4);
    DeserializationError err = deserializeJson(doc, request->body);
    if(err.code() != 0){
        docResp["ErrorDesc"] = "Deserialization error";
        docResp["Error"]     = err.code();
    }
    else{

        if(doc.containsKey("BoxId") && doc.containsKey("FunctionName") && doc.containsKey("Params")){
            if(doc["BoxId"].is<int>() && doc["FunctionName"].is<const char*>() && doc["Params"].is<JsonArray>()){

                int boxId = doc["BoxId"].as<int>();
                const char* functionName = doc["FunctionName"].as<const char*>();
                if(this->wasmController->SandboxHasFunction(boxId, functionName) == WasmController::WASM_STATUS::OK){
                    IM3Function function;
                    uint32_t argc = doc["Params"].as<JsonArray>().size();
                    JsonArray paramsArray = doc["Params"].as<JsonArray>();
                    const void *argptrp[argc];

                    for(int i = 0; i < argc; i++){
                        if(paramsArray[i].containsKey("Value") && paramsArray[i].containsKey("Ptype")){
                            switch(paramsArray[i]["Ptype"].as<int>()){
                                case 1: // INT32
                                    *((int*)argptrp[i]) = (int)paramsArray[i]["Value"].as<int>();
                                    break;
                                case 2: // INT64
                                    *((long int*)argptrp[i]) = (long int)paramsArray[i]["Value"].as<long int>();
                                    break;
                                case 3: // FLT32
                                    *((float*)argptrp[i]) = (float)paramsArray[i]["Value"].as<float>();
                                    break;
                                case 4: // FLT64
                                    *((double*)argptrp[i]) = (double)paramsArray[i]["Value"].as<double>();
                                    break;
                                case 5: // PNTR
                                    *((char*)argptrp[i]) = *(const char*)paramsArray[i]["Value"].as<const char*>();
                                    break;
                                default:
                                    argptrp[i] = nullptr;
                                    break;
                            }
                        }
                    }

                    WasmController::WASM_STATUS status =
                      this->wasmController->SandboxCallFunction(boxId, functionName, function, argc, argptrp);

                    if(status == WasmController::OK){
                        this->wasmController->SandboxGetResults(function, jsonResponseArray);
                        unsigned value = 0;
                        this->wasmController->SandboxGetResults(function, &value);
                        jsonResponseArray.add(value);
                    }
                    else {
                        docResp["ErrorDesc"] = "Error function call error";
                        docResp["Error"]     = -1;
                    }

                    this->wasmController->SandboxFree(boxId);
                }
                else {
                    docResp["ErrorDesc"] = "Error function not found";
                    docResp["Error"]     = -1;
                }
            }
            else{
                docResp["ErrorDesc"] = "Error message wrong format";
                docResp["Error"]     = 1;
            }
        }
    }
```

Slika 2 - Isječak koda koji odgovara na HTTP zahtjeve za izvođenje spremnika

Komunikacijski modul služi za istovremeno obrađivanje zahtjeva te za upravljanje TCP utičnicama no može se proširiti s ostalim komunikacijskim protokolima poput UDP, I2C ili SPI protokolom. U trenutku kada se primi zahtjev preko TCP utičnice, primljeni sadržaj se proslijedi FreeRTOS zadatku u kojem se primljeni podaci dalje obrađuju ovisno o putanji primljenoj unutar HTTP poruke. Ovim modulom se također može omogućiti rad web aplikacija time što se spremnicima izlažu funkcije za TCP komunikaciju no, zbog hardverskih limitacija odabranih mikrokontrolera, nije moguće imati više od 16 otvorenih utičnica istovremeno. Još jedna limitacija je bila izlaganje ulazne TCP utičnice pojedinom spremniku. Načina na koji se memorija dijeli sa spremnikom predstavlja veliki izazov jer bi se dolazni podaci morali kopirati iz međuspremnika u virtualno okruženje spremnika ili bi se morale izložiti funkcije za upravljanje utičnicama na hardverskoj razini. Oba predložena rješenja imaju velike mane te se u testiranju izvedbe rješenja nisu uspjele postići dovoljno dobre performanse da se spremnicima omogući direktan pristup mrežnom prometu stoga se moralo zadržati na pozivima pojedinih funkcija.

Modul za upravljanje i dohvat podataka perifernih uređaja povezanih s mikrokontrolerom služi za upravljanje uređaja povezanih s mikrokontrolerom na način da, umjesto da pruža niže metode upravljanja poput postavljanja modulacije širine impulsa ili serijske komunikacije, modul izlaže metode više razine apstrakcije. Primjer izloženih metoda je upravljanje servo motorom gdje umjesto da je izložen skup metoda za konfiguraciju izlaza mikrokontrolera i ostalih internih postavki, izložena je jedino metoda za postavljanje stupnja položaja.



Slika 3 - Prikaz strukture projekta radnog okvira za mikrokontrolere

3.3. WebAssembly moduli kao temelj spremnika

WebAssembly moduli [5] predstavljaju temeljni element za aplikacije u spremnicima, omogućujući univerzalnu, prijenosnu i sigurnu platformu za izvođenje koda u različitim okruženjima. Kao binarni format dizajniran za učinkovitost i brzinu, WebAssembly omogućuje da se aplikacije pokreću izravno u web preglednicima, ali i izvan njih, na bilo kojem uređaju ili operativnom sustavu koji podržava njegovo izvođenje. Ova tehnologija pruža značajne prednosti u pogledu performansi i sigurnosti, jer se

WebAssembly moduli izvršavaju unutar izoliranog okruženja, čime se minimizira rizik od neovlaštenog pristupa osnovnim sustavskim resursima.

Kao temelj spremnika, WebAssembly omogućuje da aplikacije budu lagane, prijenosne i neovisne o specifičnoj hardverskoj arhitekturi ili operativnom sustavu. To olakšava distribuciju i implementaciju aplikacija u različitim okruženjima, bez potrebe za opsežnim prilagodbama ili promjenama u kodu. WebAssembly moduli mogu biti dodani u spremnike zajedno sa svim svojim ovisnostima, omogućujući dosljedno izvršavanje u bilo kojem okruženju koje podržava ovu tehnologiju. Navedene prednosti čine WebAssembly idealnim za širok raspon aplikacija, web aplikacija do ugrađenih sustava, gdje je potrebno osigurati visok stupanj interoperabilnosti i performansi.

Više programskih jezika podržava kompilaciju u WebAssembly što omogućuje programerima da pišu aplikacije u jezicima koje već poznaju a zatim ih kompajliraju u WebAssembly zajedno sa svim ovisnostima za spremnike i implementaciju. To ne samo da ubrzava proces razvoja, već i proširuje mogućnosti primjene WebAssemblyja u raznim industrijama. Zbog svojih brojnih prednosti, WebAssembly postaje ključna tehnologija u razvoju modernih aplikacija koje zahtijevaju visoku razinu sigurnosti, performansi i prenosivosti, čineći ga osnovom za buduće napredne web aplikacije.

Spremnici u kontekstu ovog rada su WebAssembly moduli koji sadrže sve funkcije, memorijske stranice i globalne varijable potrebne baznom radnom okviru. Funkcije služe za dohvat, predaju ili mijenjanje funkcionalnosti niže razine na mikrokontroleru, omogućujući interakciju između softvera i hardvera. Memorijske stranice služe za pohranu podataka i stanja aplikacije, osiguravajući da svi potrebni podaci budu dostupni unutar modula tijekom izvršavanja ili za prijenos veće, kontinuirane količine podataka između radnog okruženja spremnika i baznog radnog okvira. Globalne varijable, koje su definirane unutar modula tj. spremnika, koriste se ili im se pristupa također izvan tog modula, unutar virtualnog okruženja interpretera, radi mijenjanja ponašanja algoritama i stanja unutar WebAssembly modula ili na hardverskoj razini mikrokontrolera.

3.4. Povezivanje hardverske razine s virtualnim okruženjem

Funkcije dostupne u okviru osmišljene su za apstrahiranje koda specifičnog za uređaj, poput postavljanja ulaznih i izlaznih pinova, iniciranja komunikacije i slanja naredbi definiranih protokolima specifičnih uređaja spojenih na mikrokontroler, poput senzora ili aktuatora. Umjesto apstrahiranja niskorazinskih funkcija, poput konfiguracije modulacije širine impulsa, apstrahira se prema krajnjem rezultatu, kao što je pomicanje servo motora na određeni položaj ili dobivanje podataka o temperaturi. Apstrahiranjem funkcija na višoj razini, temeljeno na njihovim stvarnim ishodima umjesto detaljno, na razini naredbi specifičnog uređaja, omogućuje se programerima da se fokusiraju na implementaciju poslovne logike, čime se smanjuje količina repetitivnog koda potrebnog za postavljanje uobičajenih funkcija i minimizira se potreba za detaljnim poznavanjem kako se pojedini zadaci trebaju implementirati na pojedinom mikrokontroleru, poput čitanja podataka sa senzora ili slanja poruke putem TCP protokola. Ovaj pristup ne samo da pojednostavljuje proces razvoja, već također poboljšava prenosivost koda između različitih platformi mikrokontrolera jer razvoj algoritama unutar spremnika ne ovisi o nižoj hardverskoj razini nego o stvarnim ishodima. Također se omogućuje dosljednije ponašanje aplikacija bez obzira na temeljni hardver jer je kod implementiran u radnom okviru umjesto u spremniku. Rezultat ovog pristupa omogućuje programerima kreiranje robusnije i održivije aplikacije s kraćim vremenom izlaska na tržište jer ne moraju poznavati funkcije specifičnog mikrokontrolera nego se mogu fokusirati na stvarne ishode.

Povezivanjem temeljne hardverske razine s virtualnim okruženjem koje sadrži funkcije, memorijske module i globalne varijable, pristupilo se na način da se omogući fleksibilna i dinamička kontrola nad aplikacijama u spremnicima, osiguravajući da se promjenama u stanju ili ponašanju modula mogu lako upravljati i prilagođavati tijekom rada, što doprinosi povećanju učinkovitosti i prilagodljivosti sustava u raznim uvjetima i zahtjevima. Funkcije, memorijski moduli i globalne varijable povezuju se s virtualnim okruženjem u trenutku alokacije virtualnog okruženja. Virtualno okruženje se alocira nanovo za svaki primljeni zahtjev a dealocira se nakon obrade i predaje rezultata poziva definiranog u primljenom zahtjevu. Virtualnim okruženjem upravlja modul

virtualizacije i izvršavanja spremnika koji se brine za pravovremenu alokaciju i dealokaciju resursa virtualnog okruženja kako bi što manje memorijskih resursa bilo alocirano i samo u vremenu koje je potrebno za izvršavanje zahtjeva. Također su pojedine funkcije i dijelovi Wasm3 biblioteke [22] optimizirane za specifične potrebe radnog okvira. Optimizacijom korištenja memorijskih resursa, ponekad jednostavnom a ponekada kao rezultat višednevnog testiranja za smanjenja do čak par kilobajta, omogućilo se istovremeno alociranje više virtualnih okruženja tj. istovremeno izvršavanja više spremnika nego što je bilo moguće u početnoj verziji radnog okruženja.

3.5. Razvoj spremnika

U skladu s industrijskim standardima za razvoj softvera, cilj je bio razviti aplikacije za spremnike u više programskih jezika te prikazati kako se može s programskim jezicima koji nisu bili zamišljeni za programiranje mikrokontrolera, razviti aplikacije koje se mogu pokretati na njima. Međutim, u vrijeme pisanja ovog rada, dostupni alati za transpiliranje koda u međujezik prije kompajliranja u WebAssembly ili za izravno kompajliranje u WebAssembly pokazali su se ili nedovoljno zrelim ili su pružali ograničene rezultate. Konkretno, autor se suočio s izazovima pri pokušaju pokretanja aplikacija napisanih u programskim jezicima Go, JavaScript, Cython i Rust na mikrokontrolerima. Ovi izazovi uglavnom su bili posljedica memorijski intenzivnih zahtjeva određenih jezika, poput upravljanja smećem (garbage collection), dodavanja dodatnog koda za podršku glavnog programu i potrebe različitih knjižnicama. Ti su čimbenici značajno povećali memorijski otisak i veličinu koda spremnika, čineći ih izvedivima samo u teoriji, ali nepraktičnima za stvarne primjene zbog performansi interpretera tj. hardverskih limitacija , dok u nekim slučajevima i potpuno nemogućima zbog ograničenja memorije na mikrokontrolerima. Rezultat je da su jedini izvedivi programski jezici za korištenje, kao glavni jezici za spremnike, C i C++. Za njihovu kompilaciju je odabran Emscripten [31], alat za kompajliranje koji pretvara C i C++ kod u WebAssembly ili JavaScript. Najveći prednost Emscripten alata je što omogućuje detaljnu kontrolu nad načinom generiranja WebAssembly koda, nekorištene funkcije se mogu zadržati u krajnjoj binarnoj datoteci, mogu se potpuno maknuti automatski generirani dijelovi koda emscriptena koji bi se inače koristili u JavaScriptu, te se može

upravljati razinom optimizacije koda s zastavicama emcc naredbe. Pomoću naredbe emcc, koja je dostupna instalacijom Emscriptena moguće je kompajlirati C i C++ kod za svaki primjer u WebAssembly, primjer je vidljiv u slici 4. Nakon što je WebAssembly datoteka generirana, koristeći Linux naredbu xxd [32], može se dobiti heksadecimalni ispis datoteke u sintaksi C polja, također vidljiv u slici 4.

```
user:~$ emcc -O3 -s WASM=1 -s SIDE_MODULE=1 hash.cpp -o hash.wasm
user:~$ xxd -i hash.wasm
unsigned char hash_wasm[] = {
  0x00, 0x61, 0x73, 0x6d, 0x01, 0x00, 0x00, 0x00, 0x00, 0x0f, 0x08, 0x64,
  0x79, 0x6c, 0x69, 0x6e, 0x6b, 0x2e, 0x30, 0x01, 0x04, 0x14, 0x02, 0x00,
  0x00, 0x01, 0x04, 0x01, 0x60, 0x00, 0x00, 0x02, 0x24, 0x02, 0x03, 0x65,
  0x6e, 0x76, 0x0d, 0x5f, 0x5f, 0x6d, 0x65, 0x6d, 0x6f, 0x72, 0x79, 0x5f,
  0x62, 0x61, 0x73, 0x65, 0x03, 0x7f, 0x00, 0x03, 0x65, 0x6e, 0x76, 0x06,
  0x6d, 0x65, 0x6d, 0x6f, 0x72, 0x79, 0x02, 0x00, 0x01, 0x03, 0x02, 0x01,
  0x00, 0x06, 0x1a, 0x05, 0x7f, 0x00, 0x41, 0x10, 0x0b, 0x7f, 0x00, 0x41,
  0x00, 0x0b, 0x7f, 0x00, 0x41, 0x04, 0x0b, 0x7f, 0x00, 0x41, 0x08, 0x0b,
  0x7f, 0x00, 0x41, 0x0c, 0x0b, 0x07, 0xb5, 0x01, 0x07, 0x11, 0x5f, 0x5f,
  0x77, 0x61, 0x73, 0x6d, 0x5f, 0x63, 0x61, 0x6c, 0x6c, 0x5f, 0x63, 0x74,
  0x6f, 0x72, 0x73, 0x00, 0x00, 0x18, 0x5f, 0x5f, 0x77, 0x61, 0x73, 0x6d,
  0x5f, 0x61, 0x70, 0x70, 0x6c, 0x79, 0x5f, 0x64, 0x61, 0x74, 0x61, 0x5f,
  0x72, 0x65, 0x6c, 0x6f, 0x63, 0x73, 0x00, 0x00, 0x14, 0x42, 0x4f, 0x58,
  0x5f, 0x44, 0x45, 0x56, 0x49, 0x43, 0x45, 0x5f, 0x54, 0x59, 0x50, 0x45,
  0x5f, 0x4e, 0x4f, 0x4e, 0x45, 0x03, 0x01, 0x1c, 0x42, 0x4f, 0x58, 0x5f,
  0x44, 0x45, 0x56, 0x49, 0x43, 0x45, 0x5f, 0x54, 0x59, 0x50, 0x45, 0x5f,
  0x43, 0x4f, 0x4c, 0x4f, 0x52, 0x5f, 0x53, 0x45, 0x4e, 0x53, 0x4f, 0x52,
  0x03, 0x02, 0x1c, 0x42, 0x4f, 0x58, 0x5f, 0x44, 0x45, 0x56, 0x49, 0x43,
  0x45, 0x5f, 0x54, 0x59, 0x50, 0x45, 0x5f, 0x4c, 0x49, 0x47, 0x48, 0x54,
  0x5f, 0x53, 0x45, 0x4e, 0x53, 0x4f, 0x52, 0x03, 0x03, 0x15, 0x42, 0x4f,
  0x58, 0x5f, 0x44, 0x45, 0x56, 0x49, 0x43, 0x45, 0x5f, 0x54, 0x59, 0x50,
  0x45, 0x5f, 0x53, 0x45, 0x52, 0x56, 0x4f, 0x03, 0x04, 0x15, 0x42, 0x4f,
  0x58, 0x5f, 0x44, 0x45, 0x56, 0x49, 0x43, 0x45, 0x5f, 0x54, 0x59, 0x50,
  0x45, 0x5f, 0x44, 0x48, 0x54, 0x31, 0x31, 0x03, 0x05, 0x0a, 0x04, 0x01,
  0x02, 0x00, 0x0b, 0x0b, 0x1a, 0x01, 0x00, 0x23, 0x00, 0x0b, 0x14, 0x01,
  0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00, 0x04,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};
unsigned int hash_wasm_len = 319;
```

Slika 4 - Kompilacija i generiranje C polja wasm datoteke

3.6. Postavljanje i pokretanje spremnika

Omogućena su dva načina na koja se spremnici mogu prebaciti na mikrokontrolere, a razlikuju se prema vrsti memorije koju zauzimaju na mikrokontroleru. Prvi način koristi statički alociranu memoriju tj. flash memoriju, definiranjem C polja koje sadrži WebAssembly kod. Drugi način je dinamička alokacija memorije unutar dostupne radne memorije, slanjem HTTP naredbe za stvaranje spremnika. Prednost pristupa sa statičkom alokacijom u glavnom programu, spremljenog u flash memoriji mikrokontrolera, je ta što program ostaje nepromijenjen dok god je isti kod na mikrokontroleru, no to je ujedno i njegova najveća mana jer ograničava fleksibilnost. Dinamičkom alokacijom putem HTTP poziva dobivamo pristup većoj količini memorije u slučaju da je RAM proširen na vanjski čip. Međutim, kako je RAM volatilna memorija, kontinuirana pohrana spremnika ovisi o stalnom napajanju mikrokontrolera. Dok oba pristupa imaju svoje prednosti i mane, moguće rješenje koje se nalazi između oba pristupa bilo bi povezivanje vanjske nevolatilne memorije s koje bi se spremnici mogli učitati nakon nestanka napajanja. Primjer bi bio povezivanje SD kartice na kojoj bi se spremali spremnici nakon primanja HTTP naredbe za pohranu. Time bi se omogućilo manje zauzeće radne memorije jer bi se spremnik dinamički alocirao samo u trenutku kada je potrebno njegovo izvođenje. Ovakva kombinacija pristupa mogla bi omogućiti brži oporavak sustava nakon nestanka napajanja, jer bi se spremnici učitavali iz vanjske memorije, smanjujući vrijeme ponovnog pokretanja aplikacija te bi osigurali njihov opstanak i u slučaju nestanka struje.

3.7. Orkestracija spremnika

U radnom okviru se orkestracija postiže tako što radni okvir omogućuje upravljanje i koordinaciju aplikacija u spremnicima na mikrokontrolerima. Radni okvir omogućuje paralelno pokretanje više spremnika, upravljanje njihovim resursima, i interakciju s hardverom. Orkestracija se postiže automatiziranim mehanizmima za dodjelu i oslobađanje resursa, kao i dinamičkim upravljanjem memorijom, što osigurava učinkovit rad sustava i minimizira rizik od preopterećenja ili sukoba resursa. HTTP API omogućava vanjskim aplikacijama ili korisnicima da kontroliraju izvršavanje spremnika,

što omogućuje prilagodbu i upravljanje sustavom u stvarnom vremenu, olakšavajući integraciju u različita okruženja i scenarije primjene.

Aplikacije unutar spremnika se izvode na HTTP naredbe te se time mogu povezati s ostalim web aplikacijama, raspodijeljenim na više različitih uređaja, također, ovisno o količini poziva i slobodnih resursa, radno okruženje alocira virtualna okruženja za izvođenje spremnika.

Wasm3 podržava i funkcionalnost mjerenja korištenih resursa poput količine izvršenih naredbi i količinu korištene memorije. Iako što radno okruženje ne koristi te funkcionalnosti, moguće je u budućnosti implementirati limite pojedinih spremnika te time usporiti njihovo izvršavanje ili implementirati bolje algoritme orkestracije postojećih resursa i aktivnih spremnika.

3.8. Primjer korištenja radnog okruženja

Osnovni primjer bio bi program namijenjen za navodnjavanje biljaka. U ovom scenariju, solenoid kontrolira protok vode prema određenom dijelu polja, a dva senzora, koja mjere vlažnost tla, povezana su s *ESP32-C3* mikrokontrolerom koji koristi ovaj okvir. U stvarnom svijetu, takav bi sustav vjerojatno bio napajan baterijama i solarnim panelom, a mogao bi se stvoriti mrežni sustav koji bi omogućio razmjenu podataka između uređaja i središnje bazne stanice. Algoritam za navodnjavanje bi se izvršavao unutar spremnika na mikrokontroleru i mogao bi se ažurirati bežično putem mreže uređaja.

Iskorištavanjem mogućnosti izvođenja funkcija u specifičnim spremnicima i dinamičkog dodavanja ili uklanjanja spremnika putem HTTP API poziva, dodatne funkcionalnosti mogu se uvesti u uređaje prema potrebi. Na primjer, preopterećenjem poziva uređaja okvira, simuliranim pozivima uređaja, algoritam se može testirati u kontroliranom okruženju bez potrebe za postavljanjem bilo kakvog hardvera na terenu, učinkovito simulirajući cijeli sustav. Dodatna funkcionalnost mogla bi se integrirati dodavanjem kamere i kreiranjem algoritma za provjeru štetnika i praćenje zdravlja biljaka. Ovaj proces slijedi isti pristup, gdje se dodaje spremnik koji sadrži sav potreban kod za obradu podataka o slici biljke i donošenje odluka o zdravlju biljaka. Kako postaje

dostupno više podataka, spremnici se mogu prilagoditi za specifične vrste biljaka, omogućujući vrlo specijalizirano praćenje i njegu. Nadalje, kako postaju dostupni noviji mikrokontroleri s nižom potrošnjom energije i većim računalnim sposobnostima, ovi spremnici, neovisni o uređaju, mogli bi učinkovito raditi na različitim hardverskim platformama bez ograničenja temeljenih na osnovnoj tehnologiji. Ovaj modularni i skalabilni pristup osigurava da se sustav može razvijati i poboljšavati tijekom vremena, prilagođavajući se novim zahtjevima i tehnološkim napredcima. S ciljem vjerodostojnog prikaza mogućnosti radnog okvira, dizajnirani su rješenja za općenite probleme uz pomoć kojih će se objasniti korištenje radnog okvira i kako omogućuje orkestriranje aplikacijama.

3.8.1. Primjer rasterećenja hash funkcija na mikrokontrolere

U ovom primjeru se pokazuje kako se zadaci generiranja hash vrijednosti mogu rasteretiti na mikrokontroler kako bi se smanjilo opterećenje drugih uređaja. Hash funkcije, kao što su *SHA-256* ili *MD5*, često zahtijevaju značajne računalne resurse, posebno kada se koriste za obradu velikih količina podataka u stvarnom vremenu [33]. Umjesto da se ovi zadaci izvršavaju na centralnom procesoru, oni se mogu prebaciti na mikrokontroler koji je posebno konfiguriran za učinkovit rad s kriptografskim funkcijama. Primjerice, u sustavu s ograničenim resursima, mikrokontroler poput *ESP32* može biti korišten za izračunavanje hash vrijednosti podataka koji dolaze sa senzora prije nego što se ti podaci pošalju na obradu ili pohranu na glavnom sustavu. Ovaj pristup omogućuje glavnom procesoru da se fokusira na druge kritične zadatke, dok mikrokontroler preuzima računanje hash funkcija. Ovakvo prebacivanje zadataka može poboljšati sigurnost sustava, jer se hashiranje obavlja bliže izvorištu podataka, smanjujući mogućnost manipulacije podacima prije nego što budu zaštićeni.

Osim samih algoritama, u glavnu C++ datoteku je također povezano zaglavlje radnog okvira koje sadrži sve dostupne funkcije. Struktura projekta mikrokontrolera je prikazana u slici 3 dok je sadržaj djela zaglavlja radnog okvira prikazan u slici 7. Projekt također sadrži pomoćnu C++ datoteku koja preopterećuje funkcije zaglavlja radnog okvira te time simulira njegove funkcije. Time se program mogao testirati na računalu umjesto da se nakon svake izmjene morao prebaciti na mikrokontroler.

Slika 5 - Prikaz testiranja hash algoritama na računalu, pomoću gcc kompajlera

Testiranje na računalu se radilo kompilacijom programa u izvršnu datoteku pomoću GCC (GNU Compiler Collection.) kompajlera [33], postupak kompilacije i izvršavanja je vidljiv u slici 5. Nakon uspješnog testiranja na računalu, aplikacija je kompajlirana u WebAssembly modul ali bez datoteke u kojoj su preopterećene funkcije zaglavlja radnog okvira, postupak je vidljiv u slici 4. Razlog toga je što će radni okvir na mikrokontroleru preopteretiti izložene funkcije i globalne varijable. Kompilacija u WebAssembly modul napravljena je koristeći EmScripten kompajlera koristeći naredbu emcc te pomoćne zastavice. Primjer naredbe za kompilaciju je vidljiv u slici 4 dok slika 6 prikazuje kod kompajliranog WebAssembly modula u WAT formatu [34]. U istoj slici su vidljive funkcije koje su označene kao “export“ vrijednosti te se povezuju pri alokaciji virtualnog okruženja za izvođenje spremnika.

```
(module
  (type (;0;) (func (param i32) (result i32)))
  (type (;1;) (func (result i32)))
  (type (;2;) (func))
  (import "env" "GetRandomInt" (func (;0;) (type 1)))
  (import "env" "__memory_base" (global (;0;) i32))
  (import "env" "memory" (memory (;0;) 1))
  (func (;1;) (type 2))
  (func (;2;) (type 0) (param i32) (result i32)
    (local i32 i32)
    i32.const 75843
    local.set 1
    local.get 0
    i32.load8_u
    local.tee 2
    if  ;; label = @1
      loop  ;; label = @2
        local.get 2
        i32.extend8_s
        local.get 1
        i32.const 33
        i32.mul
        i32.add
        local.set 1
        local.get 0
        i32.load8_u offset=1
        local.set 2
        local.get 0
        i32.const 1
        i32.add
        local.set 0
        local.get 2
        br_if 0 (;@2;)
      end
    end
    local.get 1)
  (func (;3;) (type 0) (param i32) (result i32)
    (local i32 i32)
    call 0
    local.set 1
    local.get 0
    i32.load8_u
    local.tee 2
    if  ;; label = @1
      loop  ;; label = @2
        local.get 2
        i32.extend8_s
        local.get 1
        i32.const 33
        i32.mul
        i32.add
        local.set 1
        local.get 0
        i32.load8_u offset=1
        local.set 2
        local.get 0
        i32.const 1
        i32.add
        local.set 0
        local.get 2
        br_if 0 (;@2;)
      end
    end
    local.get 1)
  (global (;1;) i32 (i32.const 16))
  (global (;2;) i32 (i32.const 0))
  (global (;3;) i32 (i32.const 4))
  (global (;4;) i32 (i32.const 8))
  (global (;5;) i32 (i32.const 12))
  (export "__wasm_call_ctors" (func 1))
  (export "__wasm_apply_data_relocs" (func 1))
  (export "_Z4hashPKc" (func 2))
  (export "_Z10hashRandomPKc" (func 3))
  (export "hash256" (func 2))
  (export "hash256Random" (func 3))
  (export "BOX_DEVICE_TYPE_NONE" (global 1))
  (export "BOX_DEVICE_TYPE_COLOR_SENSOR" (global 2))
  (export "BOX_DEVICE_TYPE_LIGHT_SENSOR" (global 3))
  (export "BOX_DEVICE_TYPE_SERVO" (global 4))
  (export "BOX_DEVICE_TYPE_DHT11" (global 5))
  (data (;0;) (global.get 0) "\01\00\00\00\02\00\00\00\03\00\00\00\04\00\00\00\00\00\00\00")
```

Slika 6 - Prikaz dekompajliranog WebAssembly modula koji sadrži hash funkcije

Postavljanjem spremnika, na mikrokontrolere, na načinima navedenim u poglavlju 3.8., moguće je poslati naredbu pojedinom spremniku. Prije nego što pokrenemo spremnik moramo saznati koji je njegov id te koje funkcije sadrži. Pozivom na adresu servera, koji je pokrenut na mikrokontroleru, možemo pronaći sve željene podatke o sveukupnom sadržaju dostupnih spremnika u mikrokontroleru dok, uz pomoć druge putanje, možemo dobiti i detaljne informacije o pojedinom spremniku. Slika 8 prikazuje odgovor na upit za dohvat podataka o sveukupnom popisu spremnika na mikrokontroleru no dostupna je i putanja koja vraća odgovor o detaljima određenog spremnika.

```
extern "C"{

        extern void DevicePutInt32(int id, int value);
        extern int DeviceGetInt32(int id);
        extern int DeviceGetDHT11Value(int id);
        extern int DeviceGetColorSensorValue(int id);
                extern int DeviceMoveServo(int id, int position);
                extern int DeviceGetSensorValue(int id);
                extern int GetRandomInt();

        int BOX_DEVICE_TYPE_NONE = 0;
        int BOX_DEVICE_TYPE_COLOR_SENSOR = 1;
        int BOX_DEVICE_TYPE_LIGHT_SENSOR = 2;
        int BOX_DEVICE_TYPE_SERVO = 3;
        int BOX_DEVICE_TYPE_DHT11 = 4;

}
```

Slika 7 - Prikaz zaglavlja radnog okvira

Hash funkcija se pokreće na način da se pošalje naredba mikrokontroleru, HTTP pozivom na određenu adresu. Sadržaj poruke čini opis željene akcije u JSON formatu. Osim oznake spremnika i naziva funkcije, mora se i unijeti popis parametara funkcije

s tipom parametra. Podaci o funkciji su vidljivi u prethodno opisanom pozivu za dohvat detalja pojedinog spremnika.

```
[
  {
    "Name": "Fib32",
    "ErrorId": false,
    "FunctionCount": 1,
    "ModuleCount": 1,
    "Functions": [
      {
        "Index": 0,
        "Name": "fib",
        "ArgCount": 1,
        "RetCount": 1,
        "Types": [
          1,
          1
        ]
      }
    ]
  },
  {
    "Name": "MoveWithSensorDataV2",
    "ErrorId": false,
    "FunctionCount": 2,
    "ModuleCount": 1,
    "Functions": [
      {
        "Index": 0,
        "Name": "MoveWithouSensorDataV2",
        "ArgCount": 1,
        "RetCount": 1,
        "Types": [
          1,
          1
        ]
      },
      {
        "Index": 1,
        "Name": "MoveWithSensorDataV2",
        "ArgCount": 2,
        "RetCount": 1,
        "Types": [
          1,
          1,
          1
        ]
      }
    ]
  },
  {
    "Name": "MoveWithSensorDataV4",
    "ErrorId": false,
    "FunctionCount": 1,
    "ModuleCount": 1,
    "Functions": [
      {
        "Index": 0,
        "Name": "MoveWithSensorDataV4",
        "ArgCount": 2,
        "RetCount": 1,
        "Types": [
          1,
          1,
          1
        ]
      }
    ]
  }
]
```


Slika 8 - Prikaz popisa dostupnih spremnika i funkcija dohvaćen upitom

3.8.2. Upravljanje senzorom temperature i servo motorom

Ovom primjeru je cilj pokazati kako se može spremnik ažurirati putem mrežnog sučelja, bez gašenja uređaja, te time promijeniti logiku upravljanja servo motorima i dohvata podataka o temperaturi i vlažnosti zraka. Hardver se sastoji od DHT11 [35] senzora za temperaturu i vlažnost zraka te 2 servo motora [36] koja su povezana s mikrokontrolerom. Unutar zaglavlja radnog okvira dodane su funkcije koje apstrahiraju rad s senzorom i servo motorom dok su unutar radnog okvira te funkcije implementirane i dodane u popis funkcija koje se povezuju kad se inicijalizira spremnik. Uređajima se pristupa na način da parametri funkcija sadrže id koji je definiran u samom mikrokontroleru, time se može koristiti jedna funkcija za mijenjanje ili primanje stanja više istih uređaja. U ovom primjeru servo A ima id 34 a servo B ima id 35 dok senzor temperature i vlažnosti zraka ima id 31, dakle svaki uređaj ima svoj jedinstveni id pomoću kojeg ga se može adresirati te pozivati njegove funkcije dostupne u zaglavlju radnog okvira prikazanog u slici 7. Sadržaj HTTP Post naredbe za izvršavanje funkcije na servo motoru s oznakom 34 je prikazan u slici 10.

Slika 9 - Primjer zahtjeva za postavljanje novog spremnika (Box) te odgovor s dodijeljenom oznakom

Slika 10 - Primjer JSON zahtjeva poslanim HTTP POST metodom na rutu /wasm/run i odgovora dobivenog izvršavanjem zahtjeva

Spremnici se ažuriraju HTTP Post naredbom na adresu za ažuriranje koja je prikazana u slici 9 zajedno sa primjerom odgovora mikrokontrolera. Naredba sadrži postojeći id spremnika te sam spremnik koji je kompajlirana WebAssembly datoteka konvertirana u Base64 tekstualni format. Konverzija se radi zbog lakšeg korištenja tekstualnih podataka umjesto da se šalje izvorna binarna datoteka čime bi se otežalo testiranje i generiranje JSON poruka. Za potrebe ovog primjera stvorena su dva spremnika, prvi omogućuje upravljanje servo motorima samo ako je temperatura i vlažnost zraka manja od poslane. Drugi primjer omogućava postavljanje servo motora u “otvoren“ položaj ,koji je programer definirao, neovisno o vrijednosti senzora no također i vraća vrijednosti senzora. Kod drugog primjera je prikazan u slici 11. Svejedno što su oba primjera vrlo jednostavna, dosljedan prikazuju primjer rada radnog okvira i funkcionalnosti koje omogućuje programerima. Dok se orkestracijom različitih spremnika dostupnih putem mrežnog sučelja omogućuje odgovor na više istovremenih poziva, korištenjem standardnih mrežnih komunikacijskih protokola omogućuje se

komunikacija s web aplikacijama, mrežnim servisima te ostalim uređajima dostupnima u mreži.

```
#include "framework.h"


void CloseServoIfTempOver40D(int servoId){
        if((int)(DeviceGetDHT11Value(4) / 100) > 40){
                DeviceMoveServo(servoId, 180);
        }
        else{
                return;
        }
}

int OpenServoAndReturnHumidity(int servoId){
        int servoMoveResponseCode = DeviceMoveServo(servoId, 90);
        if(servoMoveResponseCode < 0) return servoMoveResponseCode + 50000;
        else return DeviceGetDHT11Value(4) % 10;
}
```

Slika 11 - Kod spremnika koji omogućuje jednostavne funkcije mijenjanja pozicije servo motora ovisno o temperaturi

4. Izazovi, budući smjerovi i poboljšanja

Iako okvir razvijen za mikrokontrolere *ESP32* i *ESP32-C3* nudi značajne prednosti, kao što su prenosivost i jednostavnost razvoja, također donosi nekoliko izazova. Jedan od glavnih problema je upravljanje memorijom. Iako je *ESP32* nadograđen dodatnom količinom *SRAM*-a, učinkovito upravljanje memorijom ostaje ključno, posebno kada se više spremnika pokreće istovremeno. Ograničena memorija na *ESP32-C3*, sa samo 400 KB *SRAM*-a, dodatno otežava ovaj problem, što može dovesti do iscrpljivanja memorijskih resursa prilikom izvođenja složenih zadataka ili obrade velikih skupova podataka.

Drugi problem je kašnjenje uzrokovano pristupom hladnog pokretanja. Iako ova metoda smanjuje potrošnju memorije dodjelom resursa samo kada je to potrebno, također uvodi kašnjenja u izvršavanje zadataka. To može predstavljati problem u aplikacijama u stvarnom vremenu gdje je potrebna trenutna reakcija, poput obrade podataka senzora ili kontrole motora.

Paralelnost i raspoređivanje zadataka također predstavljaju izazove. Okvir mora učinkovito upravljati višestrukim konkurentnim zahtjevima bez uzrokovanja uskih grla ili blokiranja drugih ključnih operacija. To zahtijeva sofisticirani mehanizam raspoređivanja koji osigurava da se zadaci izvršavaju pravodobno, bez preopterećenja jezgri mikrokontrolera.

Korištenje interpretera poput Wasm3ja [22] za izvršavanje WebAssembly koda uvodi još jedan sloj složenosti. Iako je Wasm3 učinkovit za širok raspon platformi, inherentno je sporiji od nativnog izvršavanja koda, što može predstavljati ograničenje u aplikacijama gdje je kritična izvedba. Slabije performanse mogu utjecati na ukupnu učinkovitost sustava, posebno u zadacima koji zahtijevaju veliku računalnu snagu ili brzu obradu.

Sigurnosni izazovi javljaju se prilikom izvršavanja WebAssembly koda, jer sustav mora osigurati da izolirano okruženje sprječava bilo kakav neovlašten pristup osnovnom hardveru. To je posebno važno u sustavima koji bi mogli biti raspoređeni u nesigurnim ili udaljenim lokacijama.

Ažuriranje i održavanje sustava na velikom broju mikrokontrolera predstavlja izazov. Proces distribucije ažuriranja, osiguravanja konzistentnosti među uređajima i otklanjanje bilo kakvih problema koji se pojave mogu biti složeni i zahtijevati puno vremena. Osim toga, problemi skalabilnosti mogu se pojaviti kako se sustav širi, što zahtijeva pažljivo razmatranje načina upravljanja i koordinacije većih mreža mikrokontrolera bez ugrožavanja performansi ili pouzdanosti.

Dok primjeri stvarnih koristi sustava, predloženi u poglavlju 3.8., iskorištavaju mrežu mikrokontrolera za izvođenje distribuiranih računalnih zadataka i nude značajne prednosti, također predstavljaju nekoliko izazova. Jedan od glavnih problema je pouzdanost mreže; u distribuiranom sustavu, ako čak i mali postotak uređaja zakaže, to bi moglo poremetiti cijeli proces izračuna, što dovodi do nepotpunih ili netočnih rezultata. Također, upravljanje potrošnjom energije postaje ključno jer su ovi uređaji često napajani baterijama, a intenzivni računalni zadaci mogu brzo isprazniti baterije, smanjujući vijek trajanja sustava.

Izazov kašnjenja u komunikaciji i ograničenja propusnosti je također bitan. U velikim razmjerima, vrijeme potrebno za komunikaciju i sinkronizaciju uređaja moglo bi uzrokovati kašnjenja, posebno u zadacima koji zahtijevaju obradu podataka u stvarnom vremenu. Konzistentnost podataka je problem; osiguravanje da svi uređaji rade s najnovijim informacijama diljem mreže može biti teško, osobito kada uređaji rade u izoliranim ili okruženjima s lošom povezivošću.

Sigurnost je još jedan značajan problem. Ranljivosti u sustavu mogle bi biti iskorištene, omogućujući napadačima da ubace zlonamjerni kod ili ometaju rad sustava, osobito u scenarijima gdje su uređaji fizički dostupni ili rade u nesigurnim lokacijama.

Pokretanje aplikacija u spremnicima koji poslužuju web preglednike ili servise na mreži je bilo teže izvesti na učinkovit način zbog načina dijeljenja memorije s virtualnim okruženjem te se, umjesto na manje uspješnu direktnu vezu s spremnikom, fokusiralo na izvršavanje konkretnih funkcija spremnika. Budući izazov je efikasno dijeljenje kontinuiranih dijelova memorijskog prostora između radnog okvira i virtualnog okruženja bez kompromitiranja sigurnosti.

Na kraju, skalabilnost i balansiranje opterećenja ključni su problemi. Kako se broj uređaja povećava, sustav mora učinkovito rasporediti zadatke kako bi spriječio da neki uređaji budu preopterećeni dok su drugi nedovoljno iskorišteni. To zahtijeva sofisticirane algoritme za dinamičko upravljanje raspodjelom opterećenja. Otkrivanje grešaka i dijagnosticiranje kvarova u tako distribuiranom sustavu može biti iznimno izazovno, jer izoliranje izvora problema među mnogim uređajima može zahtijevati značajno vrijeme i napor, što komplicira održavanje i pouzdanost sustava.

5. Zaključak

U zaključku, rad istražuje upotrebu WebAssemblyja i spremnika za implementaciju, orkestraciju i upravljanje web aplikacijama na mikrokontrolerima s ograničenim resursima, nudeći potencijalne prednosti u pogledu prenosivosti, fleksibilnosti i učinkovitosti razvoja. Ovaj pristup omogućuje dosljedno izvođenje na različitim hardverskim platformama i pojednostavljuje proces razvoja dopuštajući dinamička ažuriranja i testiranja u virtualnim okruženjima. S pozitivne strane, smanjuje potrebu za opsežnim hardverskim postavkama i omogućuje kompatibilnost između platformi ali postoje i mnoge mane. U radu su spomenuti izazovi poput kašnjenja uvedenih metodom hladnog pokretanja i troškova performansi korištenjem interpretera poput Wasm3ja, u usporedbi s izvršavanjem izvornog koda, te također još mnogo izazova za buduća rješenja. Iako ograničenja predstavljaju prepreke, mogućnost standardizacije i optimizacije razvoja na različitim uređajima ostaje značajna prednost. Rad predstavlja obećavajući korak naprijed u modernizaciji razvoja aplikacija za uklopljene sustave, iako su potrebne daljnje optimizacije kako bi se u potpunosti iskoristio njegov potencijal u svim slučajevima upotrebe. Budući radovi mogu se usmjeriti na optimizaciju procesa hladnog pokretanja i istraživanje hibridnih pristupa koji balansiraju performanse i ograničenja resursa. Moglo bi se također istražiti i korištenje naprednijih tehnika za upravljanje memorijom i povećanje performansi, kao što su *Just-In-Time* (JIT) kompilacija ili metode predmemoriranja podataka dostupne u drugim interpreterima. Unatoč izazovima, ovaj pristup otvara nove mogućnosti za skalabilne i učinkovite sustave temeljene na mikrokontrolerima, što bi moglo imati veliki utjecaj na budući razvoj ugrađenih sustava i IoT aplikacija.

Literatura

[1] D. Merkel, »Docker: lightweight Linux containers for consistent development and deployment,« *Linux J.,* svez. 2014, br. 239, p. Article No.: 2, 2014.

[2] I. B. a. P. C. a. K. C. a. P. C. a. S. F. a. V. I. a. N. M. a. V. M. a. R. R. a. A. S. a. P. Suter, »Serverless Computing: Current Trends and Open Problems,« *Research Advances in Cloud Computing,* pp. 1-20, 2017.

[3] A. a. G. M. a. M. M. a. A. M. a. A. M. Al-Fuqaha, »Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications,« *IEEE Communications Surveys & Tutorials,* svez. 17, br. 4, pp. 2347-2376, 2015.

[4] Arduino, »Arduino Dokumentacija,« [Mrežno]. Dostupno: https://www.arduino.cc/en/Guide/Introduction. [Pokušaj pristupa 28 8 2024].

[5] W. C. Group, »WebAssembly Dokumentacija,« [Mrežno]. Dostupno: https://webassembly.github.io/spec/core/intro/introduction.html. [Pokušaj pristupa 8 28 2024].

[6] G. a. P. R. a. W. Z. a. P. G. a. H. C. a. C. L. Peach, »eWASM: Practical Software Fault Isolation for Reliable Embedded Devices,« *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* svez. 39, br. 11, pp. 3492-3505, 2020.

[7] F. G. S. Eriksson, »Containerizing WebAssembly : Considering WebAssembly Containers on IoT Devices as Edge Solution,« Dissertation, 2021.

[8] Systems, Espressif, »Izbor mikrokontrolera tvrtke Espressif Systems,« [Mrežno]. Dostupno: https://www.espressif.com/en/products/socs. [Pokušaj pristupa 28 08 2024].

[9] STMicroelectronics, »Izbor mikrokontrolera tvrtke STMicroelectronics,« [Mrežno]. Dostupno: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html. [Pokušaj pristupa 28 8 2024].

[10] Semiconductor, Nordic, »Izbor mikrokontrolera tvrtke Nordic Semiconductor,« [Mrežno]. Dostupno: https://www.nordicsemi.com/Products/Wireless/WiFi/Products. [Pokušaj pristupa 28 8 2024].

[11] FreeRTOS, »Uvod u FreeRTOS,« [Mrežno]. Dostupno: https://www.freertos.org/Why-FreeRTOS. [Pokušaj pristupa 28 8 2024].

[12] Systems, Espressif, »Dokumentacija ESP-IDF radnog okruženja,« Espressif Systems, [Mrežno]. Dostupno: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/. [Pokušaj pristupa 28 8 2024].

[13] STMicroelectronics, »Dokumentacija razvojnog okruženja STM32Cube,« [Mrežno]. Dostupno: https://www.st.com/en/development-tools/stm32cubeide.html. [Pokušaj pristupa 27 8 2024].

[14] Raspberry Pi Ltd., »Opis mikrokontrolera tvrtke Raspberry Pi,« [Mrežno]. Dostupno: https://www.raspberrypi.com/documentation/microcontrollers/silicon.html. [Pokušaj pristupa 29 8 2024].

[15] Docker Inc., »Opis Docker alata,« [Mrežno]. Dostupno: https://docs.docker.com/get-started/docker-overview/. [Pokušaj pristupa 28 8 2024].

[16] Podman, »Opis Podman alata,« [Mrežno]. Dostupno: https://docs.podman.io/en/latest/. [Pokušaj pristupa 30 8 2024].

[17] Kubernetes, »Uvod u Kubernetes platformu,« [Mrežno]. Dostupno: https://kubernetes.io/docs/concepts/overview/. [Pokušaj pristupa 30 8 2024].

[18] Bytecode Alliance, »Dokumentacija Wasmtime biblioteke,« [Mrežno]. Dostupno: https://wasmtime.dev/. [Pokušaj pristupa 27 8 2024].

[19] Cloud Native Computing Foundation Sandbox, »Dokumentacija za razvoj s WasmEdge,« [Mrežno]. Dostupno: https://wasmedge.org/docs/. [Pokušaj pristupa 27 8 2024].

[20] A. Scheidecker, »Opis WAVM virtualnog stroja,« [Mrežno]. Dostupno: https://wavm.github.io/. [Pokušaj pristupa 25 8 2024].

[21] Wasmer, »Opis Wasmer radnog okruženja,« [Mrežno]. Dostupno: https://docs.wasmer.io/runtime. [Pokušaj pristupa 24 8 2024].

[22] V. Shymanskyy, »Kuharica biblioteke wasm3,« [Mrežno]. Dostupno: https://github.com/wasm3/wasm3/blob/main/docs/Cookbook.md.

[23] Bytecode Alliance, »Opis WebAssembly Micro Runtime projekta,« [Mrežno]. Dostupno: https://bytecodealliance.github.io/wamr.dev/. [Pokušaj pristupa 24 8 2024].

[24] Wasmi Labs, »Opis Wasmi WebAssembly interpretera,« [Mrežno]. Dostupno: https://wasmi-labs.github.io/blog/posts/wasmi-v0.32/. [Pokušaj pristupa 29 8 2024].

[25] B. K. a. D. M. D. S. Wallentowitz, »Potential of WebAssembly for Embedded Systems,« *Mediterranean Conference on Embedded Computing (MECO),* pp. 1-4, 2022.

[26] PlatformIO, »Opis projekta PlatformIO,« [Mrežno]. Dostupno: https://docs.platformio.org/en/latest/what-is-platformio.html. [Pokušaj pristupa 31 8 2024].

[27] QEMU, »Emulator i virtualizator Qemu,« [Mrežno]. Dostupno: https://www.qemu.org/. [Pokušaj pristupa 31 8 2024].

[28] Lab Center, »Proteus IoT Builder modul za simulaciju i dizajn hardvera,« [Mrežno]. Dostupno: https://www.labcenter.com/iotbuilder/. [Pokušaj pristupa 24 8 2024].

[29] Wireshark Foundation, »Popis funkcionalnosti dostupan unutar programa Wireshark,« [Mrežno]. Dostupno: https://www.wireshark.org/about.html. [Pokušaj pristupa 29 8 2024].

[30] B. Blanchon, »Projekt ArduinoJSON,« [Mrežno]. Dostupno: https://arduinojson.org/. [Pokušaj pristupa 29 8 2024].

[31] L. T. Udovičić, »Projekt završnog rada,« [Mrežno]. Dostupno: https://github.com/wasmboxes/EspBase.

[32] Emscripten, »Alati dostupni u Emscripten alatnom okviru,« [Mrežno]. Dostupno: https://emscripten.org/docs/tools_reference/index.html. [Pokušaj pristupa 29 8 2024].

[33] J. Weigert, »Dokumentacija korištenja naredbe xxd,« [Mrežno]. Dostupno: https://linux.die.net/man/1/xxd. [Pokušaj pristupa 29 8 2024].

[34] S. William, Cryptography and Network Security - Principles and Practice, 7th Edition, Pearson Education, 2007.

[35] WebAssembly Community Group, »Dokumentacija WebAssembly tekstualnog formata,« [Mrežno]. Dostupno: https://webassembly.github.io/spec/core/text/index.html. [Pokušaj pristupa 29 8 2024].

[36] Mouser Electronics, »Tehnička dokumentacija senzora DHT11,« [Mrežno]. Dostupno: https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf. [Pokušaj pristupa 21 8 2024].

[37] Mouser Electronics, »Opis proizvoda SG90 servo motora,« [Mrežno]. Dostupno: https://www.mouser.ca/ProductDetail/DFRobot/SER0043?qs=5aG0NVq1C4zNL6r3%252B8mR2g%3D%3D. [Pokušaj pristupa 28 8 2024].

[38] Free Software Foundation Inc., »GNU Projekt i gcc alat,« [Mrežno]. Dostupno: https://gcc.gnu.org/. [Pokušaj pristupa 29 8 2024].

Popis slika i tablica