Operacijski sustav za internet stvari

Fuček, Tomislav

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:180246

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Tomislav Fuček

Operacijski sustav za Internet stvari (IoT)

DIPLOMSKI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Tomislav Fuček

Matični broj: 35918/07–R

Studij: Organizacija poslovnih sustava

Operacijski sustav za Internet stvari (IoT)

DIPLOMSKI RAD

Mentor/Mentorica:

Prof. dr. sc. Ivan Magdalenić

Varaždin, lipanj 2024.

Tomislav Fuček

Izjava o izvornosti

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

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog rada su operacijski sustavi za Internet stvari. U sljedećim odlomcima detaljnije su opisani razne vrste operacijskih sustava za IoT, usporedba i primjena istih, kao i opis same arhitekture takvih sustava. Rad isto tako sadrži i praktični dio, čija je izrada opisana kroz faze rade, popraćene slikama i komentarima.

Motiv za pisanje ovoga rada i razrade teme OS-a za Internet stvari je upravo sve veća primjena IoT uređaja. Digitalna transformacija obilježila je cijelo jedno desetljeće naše civilizacije, a sukladno tome gotovo svaki uređaj posjeduje čip i neke od hardverskih komponenti. Uz to, sve je veći naglasak na međusobnoj povezanosti takvih uređaja s drugim uređajima i naravno, internetom. Sintezom tih stvari dobivamo IoT uređaj, a takvom percepcijom lako zaključujemo da su upravo ti uređaji nalaze svuda oko nas. Svakodnevno se susrećemo s takvom vrstom uređaja, bez da i razmišljamo kako je, primjerice moderni frižider, izumljen originalno prije gotovo dva stoljeća, danas IoT uređaj.

Ključne riječi: Operacijski sustav, IoT, Rapsberry Pi, Internet of Things, mikrokontroler

Sadržaj

1. Uvod

Budućnost komunikacije leži u Internetu stvari (eng. *Internet of Things*, u daljnjem kontekstu IoT), koji je polako postao jako tražena tehnologija današnjice. Primjene IoT-a su raznolike i kreću se od običnog prepoznavanja glasa do kritičnih svemirskih programa. Kroz zadnjih nekoliko godina uloženo je mnogo napora u dizajniranje operativnih sustava za IoT uređaje, glavni razlog što niti jedan tradicionalni Windows/Unix postojeći operativni sustavi u stvarnom vremenu ne mogu zadovoljiti zahtjeve IoT uređaja. Ovaj rad predstavlja pregled operativnih sustava koji su do sada dizajnirani za IoT uređaje i također ocrtava generički okvir koji donosi bitne značajke poželjne u OS-u prilagođenom za IoT uređaje.

Prvi dio rada odnosi se na usporedbu raznih IoT operacijskih sustava, s naglaskom na područje namjene, i sam dizajn arhitekture spomenutih sustava. Drugi dio orijentiran je na praktični dio, gdje je prikazan koncept izrada Pametne kante za smeće (eng. *smart bin*). Ideja je na jednostavnom primjeru prikazati osnovni koncept onoga što bi mogao biti jedan od koraka prema boljem upravljanju otpadom. Uvođenjem koncepta pametne kante za smeće, prikazuje se kako se IoT tehnologija može integrirati u svakodnevni život, kao i mnoge mogućnosti koje pruža. Kroz ovaj primjer, demonstriraju se osnovne funkcionalnosti koje su bitne za IoT uređaje, poput senzora za praćenje razine otpada, mogućnosti bežične komunikacije za slanje podataka prema aplikaciji za nadzor, te korisničkog sučelja za praćenje i upravljanje uređajem.

2. Metode i tehnike rada

Ideja za ovu temu proizašla je upravo istraživajući na koji način gradovi rješavaju sve veći problem upravljanje otpadom. Zagreb, kao naš glavni grad, posebice ima problem s navedenim problemom. To je jedan realan problem, s kojime se bore brojni Europski gradovi, međutim ne postoji jedno univerzalno rješenje i koncept koji je primjenjiv. Implementacija ovoga projekta u neki realni sustav, zapravo bi bila samo jedan od tehnika za bolji menadžment otpadom.

Hardware, odnosno mikrokontroler korišten kao baza ovog projekta je Raspberry Pi. Upravo radi svoje povoljne cijene i jednostavnosti korištenja odabran je kao baza za ovaj koncept.

Raspbian je odabran operacijski sustav odabran za izvedbu ovog koncepta. Besplatan je, i primarno razvijen i optimiziran za rad s Raspberry Pi kontrolerom, tako je da očiti izbor obzirom na kontroler. Bitna napomena je da on kao takav dolazi s brojnim predinstaliranim programima kao što su IDE-i za programski razvoj, Internet preglednik, video editor, optimizirani za rad na Raspberry-ju.

Programski jezik u kojemu je projekt pisan je Python. On je jedan od najpopularnijih jezika za rad s mikrokontrolerima, i velik broj projekata baziranima na Raspberry Pi uređajima je pisan upravo u Pythonu.

Obzirom da je podatke s mikrokontrolera potrebno prikazati na UI, u kombinaciji s Python korišten je Flask. To je zapravo *lightweight micro-framework* u smislu da pruža osnove, ali dozvoljava fleksibilnost i fino podešavanja aplikacija s proširenjima i ekstenzijama. Obzirom da je UI za ovaj projekt osnovan, idealan je odabir, pošto je pisan u Pythonu i namijenjen za rad s njim.

Za bazu podataka odabran je Firebase radi svoje jednostavnosti, fleksibilnosti i mogućnosti svoje *Realtime Database*, odnosno baze podataka koja omogućuje osvježavanje podataka na klijentima u realnome vremenu.

3. Operacijski sustavi

Operacijski sustav definiramo kao software koji djeluje kao spona, sučelje između krajnjih korisnika računala i njegovih hardverskih komponenti. Možemo reći da je operacijski sustav (OS) program, odnosno set ili kolekcija programa koji omogućuju komunikaciju s računalnom. Svaki računalni sustav mora imati operacijski, odnosno operativni sustav za pokretanje gotovo bilo kojeg drugog programa. U suštini, računalo bez operacijskog sustava je praktički „beskorisno“. Može se reći da je operacijski sustav zapravo okruženje potrebno za pokretanje programa i obavljanje njihovih zadataka, posrednik između hardware-a i korisnika, platforma za pokretanje drugih programa. OS koordinira i upravlja memorijom, procesima računala, kao i svim software-om i hardware-om. [1]

3.1. Funkcije OS-a

Operacijski sustav zadužen je za, i posjeduje velik broj funkcija u radu i upravljanju računalom i programima, a kroz iduće točke navedene su neke od najbitnijih.

- **Upravljanje procesorom** (*Processor Management*) – operacijski sustav upravlja radom procesora, dodjeljuje mu poslove i mora osigurati da svaki pojedini proces ima dovoljan vremenski interval kako bi pravilno izvršio svoje zadatke i adekvatno funkcionirao
- **Upravljanje memorijom** (*Memory Management*) – OS dodjeljuje, odnosno oslobađa memoriju procesima i mora osigurati da svaki proces ima potrebne resurse za svoje zadatke. Potrebno je osigurati alokaciju resursa kako se ne bi desilo da jedan proces koristi i troši memoriju dodjeljuju nekom drugom procesu. OS delegira alokacijom sistemske memorije
- **Upravljanje procesima** (*Job Control*) – ova točka odnosi se na kontrolu više *task-ova*, odnosno zadaća na operacijskom sustavu. Općenito se na sustavu izvodi više programa istovremeno, tako da je potrebno utvrditi koje se aplikacije i programi, moraju izvoditi kojim redoslijedom i koliko vremenom je svakom od procesa potrebno. Upravljanje procesima predstavlja kreiranje i terminiranje procesa, sinkronizaciju i monitoriranje.

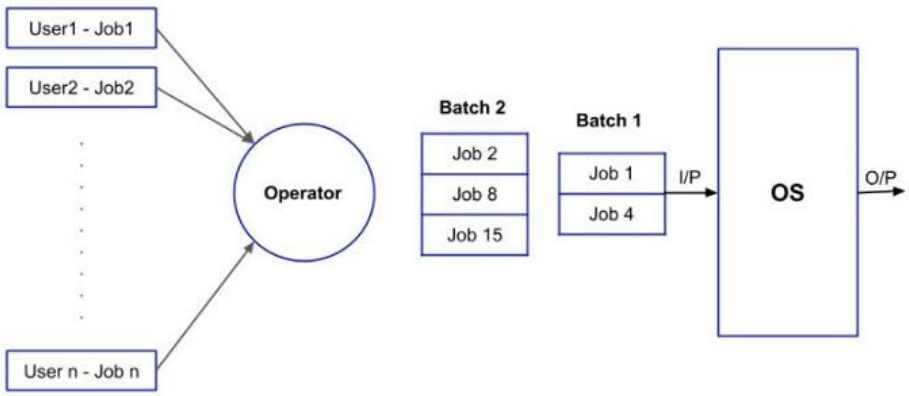
- **Upravljanje uređajima** (*Device Management*) – kontrola rada ulazno/izlaznih uređaja. OS zaprima zahtjeve uređaja i sukladno tome obavlja određene zadatke i vrši komunikaciju s procesima koji to zahtijevaju
- **Upravljanje podacima** (*File Management*) – bitna zadaća operacijskog sustava koji mora osigurati integritet podataka, pratiti podatke o kopiranju, brisanju i pohrani podataka na način da to bude organizirano i efikasno
- **Sigurnost** (*Security*) – mandatorna zadaća sustava koji koristeći razne tehnike zaštićuje od neovlaštenog pristupa, održava Firewall, i prati bilo kakve moguće ranjivosti sustava koje su popraćene porukama za korisnike
- **Detekcija grešaka** (*Error Detection*) – pod ovu funkcionalnost spada provjera bilo kakvih vanjskih prijetnji zlonamjernog software-a, kao i bilo kakvu vrstu hardverski malfunkcija i grešaka. U slučaju detekcije takvih ranjivosti, sustav mora adekvatno upozoriti korisnika tako da se mogu poduzeti radnje za sprječavanje štete sustavu [2]

Slika 1: Funkcije OS-a
Izvor: https://3.bp.blogspot.com/-g0YlNZ2in94/W7MkAF5tl7I/AAAAAAAAW6U/mQ5IsiZNEs8_17CDxdTwfcZBpO19JL6awCK4BGAYYCw/s1600/5-key-functions-of-operating-systems-onlineclassnotes.com.png

3.2. Vrste OS-a

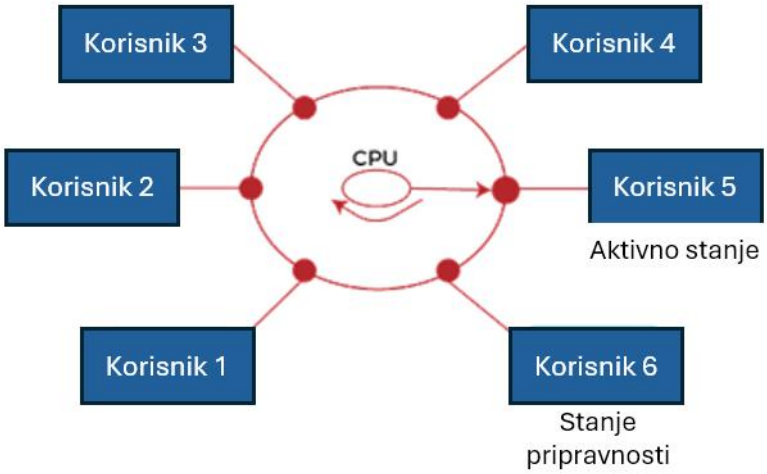
Kada pričamo o vrstama operacijskih sustava, postoji mnogo raznih klasifikacija. Obzirom na izvođenje dretvi postoje jednodretveni (*eng. single-tasking operating systems*) i višedretveni (*eng. multi-tasking operating systems*) operacijski sustavi. Kako im samo ime kaže, jednodretveni operacijski sustavi mogu izvoditi samo jednu dretvu instrukcija. Takva vrsta OS-a je relativno jednostavna pošto ne zahtjeva kompleksne mehanizme za management zadataka i alokaciju resursa. S druge strane, višedrtevni operacijski sustavi mogu izvoditi više dretvni, odnosno programa istovremeno. Takvi sustavi uglavnom koriste višejezgrene procesore, iako je prividna višedretvenost moguća i kod jednodretvenih sustava. [3]

Kod takozvanih *Batch* operacijski sustava, slične vrste zadatka grupirane su zajedno na jednom računalu s više korisnika (eng. *mainframe*). Sustav stavlja sve grupe u red čekanja (eng. *queue*) i zatim ih izvršava jedan po jedan po principu *first come first serve*. Korisnici prikupljaju izlazne podatke po završetku egzekucije. [4]



Slika 2: Batch operacijski sustav
Izvor: https://www.geeksforgeeks.org/batch-processing-operating-system/

Time-sharing operacijski sustavi funkcioniraju na način da svaki zadatak dobiva određeno vrijeme za svoje izvršavanje. Kod takvih sustava CPU se prebacuje među programima koje predaju različiti korisnici prema nekom rasporedu. To su višezadaćni sustavi gdje svaki od korisnika dobiva određeno vrijeme CPU-a. Vrijeme koje svaki zadatak dobije za izvršavanje naziva se kvantum (*eng. quantum*). [5]



Slika 3: Time-sharing OS
Izvor:https://images.shiksha.com/mediadata/ugcDocuments/images/wordpressImages/2023_09_Copy-of-What-is-29.jpg

Distribuirani operacijski sustavi koriste se za posluživanje više korisnika i aplikacija u stvarnom vremenu koristeći brojne središnje procesore. Poslovni, odnosno zadaci obrade su raspodijeljeni među procesorima. Upravo ti vanjski sustavi, odnosno autonomna povezana računala međusobno komuniciraju koristeći zajedničku mrežu. Imaju vlastitu memoriju i CPU i nazivaju se distribuirani sustavi ili „labavo povezani sustavi“. [5]

Postoji još nekoliko vrsta operacijskih sustava. Real-time operacijski sustavi koji služe sustavima u stvarnom vremenu. Koriste se kada se mnogo događaja dogodi u kratkom vremenu, primjerice kao što su simulacije u stvarnom vremenu.

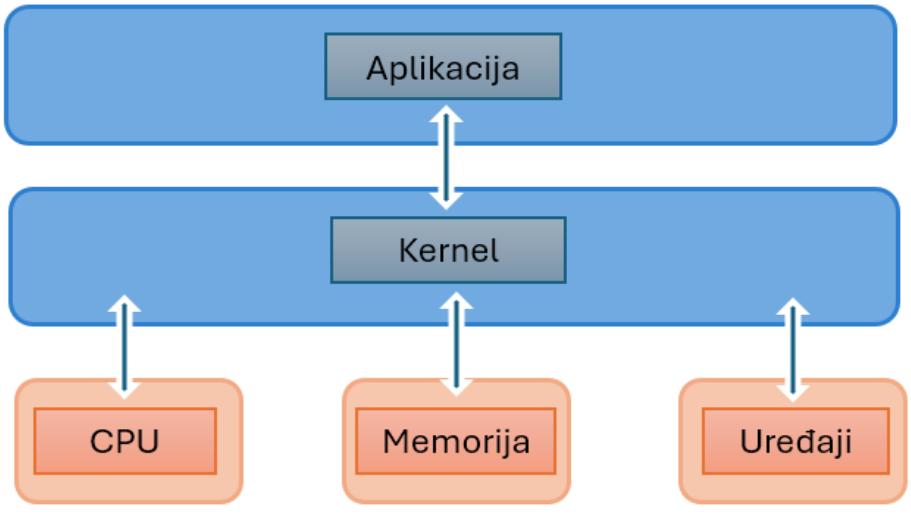
Prema podacima iz 2022., najpopularniji operacijski sustav je Microsoft Windows, koji zauzima oko 30%, odnosno 70% tržišnog udjela ako se u obzir ne uzimaju mobilni operacijski sustavi. Obzirom da je kao OS prisutan još od 1980-tih godina to nije previše čudna informacija. Odmah iza njega je Apple-ov macOS s udjelom nešto većim od 20%, a na trećem mjestu Linux s oko 3% tržišnog udjela. MacOS je obožavan kod korisnika upravo radi jednostavne uporabe, brzine, i konstantnog poboljšavanja korisničkog sučelja. Linux je voljen upravo jer je *open-source* platforma koju teoretski svatko može nadograđivati. To je zapravo familija OS-a s većim brojem raznih distribucija. Velike prednosti su mu što je besplatan, prilagodba je jako jednostavna i nudi niz odličnih opcija za one koji razumiju kako ga koristiti. Uz to, popularan je kao operacijski sustav za stvari kao što su penetracijska testiranja i sigurnost općenito. [6]

3.3. Dizajn operacijskih sustava

Temljeni i najbitniji dio operacijskih sustava je Kernel. Esencijalni je dio svakog OS-a, a definiramo ga kao jezgru operacijskog sustava. U trenutku kada pokrenemo računalo, BIOS (*Basic Input/Output System*) izvodi inicijalizaciju. Nakon što se početna inicijalizacija završi, *bootloader*, program koji obavlja niz funkcija kao što je POST (eng. *Power-on self-test*), koji provjerava da hardware-ske komponente rade kako treba, pokreće i kernel. BIOS u tom trenutku zapravo predaje kontrolu kernelu, koji se učitava u zaštićeni memorijski prostor pošto je zadužen za krucijalne zadatke. U slučaju da se kernel ne može učitati, računalo se ne može pokrenuti. Nakon toga kernel učitava ostale komponente OS-a kako bi se dovršilo pokretanje sustava i kontrola učinila dostupna korisnicima kroz sučelje. [7]

U suštini kernel je zadužen za pokretanje i upravljanje aplikacijama, upravljanje osnovnim hardware-skim komponentama i omogućava korisnicima sučelja potrebna za interakciju s računalom. Na detaljnijoj postizanje navedenog se vrši kroz iduće zadatke:

- Učitavanje i upravljanje manje kritičnim komponentama OS-a
- Kontrola memorije, koja uključuje:
    - Rukovanje i kontrola konflikata i pogrešaka u alokaciji memorije
    - Upravljanje i optimizacija hardware resursa, kao što je CPU i *cache*
    - Delegiranje koji memorijski prostor svaki aplikacijski proces koristi
- Pristup i upravljanje ulazno/izlaznim jedinicama



Slika 4: OS Kernel
Izvor: Vlastita izrada

Postoji tri arhitekture kernela, odnosni jezgri operacijskih sustava. To su mikrojezgrena, monolitna i hibridna arhitektura. Mikrojezgrena arhitektura podrazumijeva da se sve usluge jezgre nalaze u adresnom prostoru jezgre. Za komunikaciju takva vrsta OS-a koristi pakete podataka, razne signale i funkcije određenim procesima. To je minimalistička arhitektura gdje je veličina jezgre smanjena i funkcionalnosti su minimalne. Sve što je moguće delegira se izvan jezgre. Glavni koncept ove strukture je minimalizam i maksimiziranje modularnosti. Često su takve jezgre kompleksne za implementaciju i dizajn takvog sustava iziskuje pažljivo planiranje. [7]

Monolitne jezgre veće su prethodno opisanih jer sadrže i korisničke servise, odnosno usluge u istom adresnom prostoru. Možemo reći da cijeli operacijski sustava u takvoj arhitekturi djeluje kao jedan program. Isto tako koristi brži komunikacijski protokol za izvršavanje procesa između hardware-a i software-a. Međutim, takav dizajn predstavlja i veći sigurnosni rizik za sustav jer ako neka usluga ili servis zakaže, potencijalno postoji mogućnost da se cijeli sustav sruši/ugasi. [8]

Hibridne jezgre je zapravo spoj dvije prethodno opisane arhitekture. Ima veću modularnost, i dijelovi OS ipak dobivaju memorijsku zaštitu. Ovakav dizajn kombinira mikrojezgrenu i monolitnu strukturu, balansira performanse i efikasnost monolitne, i modularnost i pouzdanost mikrojezgrene strukture. Stvorio ga je Apple XNU (Not Unix) i koristi se kao arhitektura svih njegov operacijskih sustava (macOS, iOS, watchOS…). [8]

4. IoT – Internet of Things

Internet stvari (*eng. Internet of Things,* u danjem kontekstu IoT) je sustav međusobno povezanih računala, mehaničkih i/ili digitalnih strojeva, i zapravo svih objekata koji imaju neke jedinstvene identifikatore i mogućnost prijenosa podataka preko mreže bez potrebe interakcije između čovjeka i računala. IoT opisuje mrežu fizičkih objekata u koje su ugrađeni razni senzori i software u svrhu povezivanja i razmjene podataka s drugim uređajima i sustavima putem interneta. [9]

Stvar u Internetu stvari može biti bilo koji objekt koji ima ugrađene senzore i preko dodijeljene IP adrese komunicira i razmjenjuje podatke preko mreže (interneta). Taj objekt može biti osoba s implantatom monitora srca, *smart* TV, nosivi *smart* uređaji (satovi, narukvice, pa čak i poneke naočale), *smart home* uređaji, pametni frižideri i hladnjaci, itd.

Slika 5: Interent of Things
Izvor: https://1629777.fs1.hubspotusercontent-na1.net/hub/1629777/hubfs/MicrosoftTeams-image%20%2885%29.png?width=1110&name=MicrosoftTeams-image%20%2885%29.png

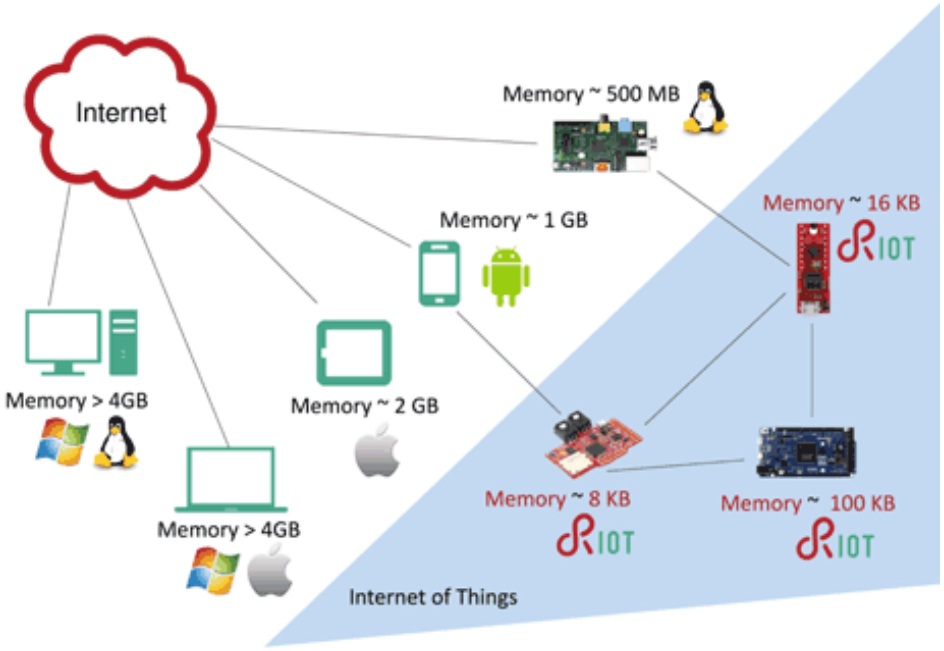
Dakle, za IoT uređaje možemo reći da to nestandardni uređaji koji se bežično povezuju na mrežu i mogu prenositi podatke preko iste. Takve uređaje uglavnom možemo monitorirati i kontrolirati na daljinu. Postoje razne klasifikacije IoT uređaja, a možemo ih podijeliti prema namjeni/primjeni:

- Potrošački IoT uređaji - kućanski aparati, rasvjeta, *voice assist*
- Komercijalni - zdravstvo, transport
- Vojni - biometrija, dronovi, razni roboti
- Industrijski (energija i manufaktura – pametna agrikultura, big data
- Infrastruktura – pametni gradovi (*smart cities*), naglasak na raznim senzorima i automatici, infrastruktura i sustavi za upravljanje [11]

5. Operacijski sustavi za IoT

U principu operacijski sustav za IoT uređaje nije nužan. Međutim, IoT uređaji vremenom su postali sve složeniji. Čvorovi imaju više senzora, količina podataka za obradu je sve veća i potreba za slanjem podataka preko mreže je velika. Neki uređaji imaju i sučelja koja često uključuju razne grafičke zaslone, pa čak i biometriju. Uređaji koji su temeljeni na 8 lii 16-bitnim arhitekturama, prelaze na 32-bitne kako troškovi padaju a složenost sustava raste. Upravo radi ovakvih promjena, korištenje IoT uređaja bez OS-a je poprilično teško, a ujedno i neefikasno. U konačnici i samo programiranje takvih uređaja je lakše korištenjem operacijskih sustava pošto OS kao takav rješava sistemske *low-level* zadatke. Za nekakvo „zlatno pravilo“ možemo reći da sustavi koji konzumiraju manje od 16kB RAM memorije ne zahtijevaju operacijski sustav. Takvih sustava je sve manje. [12]

U idućim poglavljima detaljnije je opisan dizajn i arhitektura IoT sustava, neke od specifikacija koje je potrebno uzeti u obzir pri odabiru OS-a, arhitektura takvih sustava, kao i najpopularniji operacijski sustavi s svojim prednostima.



Slika 6: IoT uređaji
Izvor: https://devopedia.org/iot-operating-systems

5.1. Arhitektura IoT-a

Kada pričamo o Internetu stvari, zapravo i ne postoji univerzalna arhitektura koja se striktno slijedi. Ona zapravo ovisi o njegovoj implementaciji i funkcionalnostima koje su potrebne. Ipak, postoji osnovni slijed procesa koji se prati i na temelju kojega se gradi Internet stvari. Arhitekturu sustava možemo podijeliti u četiri segmenta/sloja:

1. **Sloj senzora** – senzori i aktuatori prisutni su u ovom sloju. Oni prihvaćaju podatke, parametre kao što su primjerice temperatura, vlaga u zraku, zvučni signali, koji se obrađuju i emitiraju preko mreže. Glavna zadaća ovog sloja jest upravo obrada podataka i prosljeđivanje istih drugom sloju da se određene radnje mogu učiniti. [11]
2. **Mrežni sloj** – možemo ga definirati kao sloj koji povezuje senzore i *'middleware'*. Njegova zadaća jest prihvatiti podatke od senzora i proslijediti ih u *middleware* sloj koristeći tehnologije kao što su WiFi, 3G, 4G, infracrveno povezivanje, itd. Ovaj sloj naziva se i komunikacijskim slojem jer je odgovora za komunikaciju i prosljeđivanje parametara na daljnju obradu. Prijenos podataka mora se obaviti na siguran način čuvajući povjerljivost podataka. [13]
3. **Sloj obrade podataka** – ovaj sloj predstavlja procesorsku jedinicu IoT sustava. Podaci se analiziraju i obrađuju prije slanja u podatkovne centre gdje aplikacije pristupaju tim podacima. Aplikacije, često poslovne, nadziru i upravljaju tim podacima i pripremaju ih na daljnje radnje i obradu. Radnje koje se događaju u ovome sloju su računanje, obrada, pohrana i mogućnost poduzimanja određenih radnji, kao i donošenje odluka temeljenih na izračunima napravljenih nad podacima koji su došli od senzora. [12]
4. **Aplikacijski sloj** – zadnji sloj u arhitekturi IoT-a. U ovom sloju podacima se i upravlja i koriste ih korisnici krajnjih aplikacija. Ovaj sloj zadužen je za pružanje resursa korisniku. Primjerice, u aplikaciji pametnog doma, ako korisnik pritisni tipku za paljenje svijetla ili podizanje roleta, aplikacijski sloj zadužen je za pružanje korisniku resursa specifičnih za aplikaciju. Specificira razne uporabe za IoT kao što su pametne kuće, pametni gradovi, zdravstvo, obrana, itd. [13]

Slika 7: Arhitektura IoT-a
Izvor: https://www.hiotron.com/iot-architecture-layers/

U početku klasifikacija arhitekture bila je troslojna. Ona se koristila u početnim, inicijalnim fazama IoT industrije. To su bili aplikacijski, mrežni i perceptivni sloj. U nekim klasifikacijama spominje se i pet sloj, *Business Layer*, odnosno poslovni sloj. On se temelji na analogiji da uspjeh uređaja ne ovisi o korištenim tehnologijama, već o tome kako se isporučuje potrošačima. Ipak je svaki informacijski sustav zapravo podsustav poslovnome sustavu. Poslovni sloj uključuje izradu dijagrama, grafova i analize rezultata. [14]

5.2. Karakteristike IoT operacijskih sustava

Tradicionalni operacijski sustavi kao što su Windows i macOS, nisu dizajnirani prema specifičnim potrebama koje IoT uređaji zahtijevaju. Takva vrsta operacijskih sustava nije optimizirana za podršku zahtjeva kao što su niska potrošnja energije, limitirana memorija i efikasno procesuiranje podataka. Takve karakteristike IoT uređaji upravo zahtjevaju. Upravo iz tih razloga korištenjem optimiziranih OS-a dizajniranih za te specifične potrebe dozvoljava da IoT uređaj maksimiziraju učinkovitost. [15]

Iz navedenih razloga takvi operacijski sustavi dizajnirani su da budu lagani i da je učinkovitost maksimizirana kako bi se mogli koristiti u raznim uređajima. Isto tako bitna stavka je tzv. OTA (eng. *over-the-air updates*), kako bi se uređaji mogli bežično ažurirati. Modularnost je još jedna karakteristika ovakvih sustava, pošto je umreženost i povezivanje s drugim uređajima značajaka IoT uređaja.

Što se tiče komunikacijskih protokola, koji dirigiraju načinom na koji uređaji komuniciraju, najčešće korišteni uz Bluetooth su Zigbee i Z-Wave. Oni koji djeluju kao alternative Bluetooth tehnologiji i funkcioniraju relativno slično. [15]

Sigurnost je mandatorna stavka dizajniranja i arhitekture IoT operacijskih sustava jer, kao što je već opisano, uređaji su većinski međusobno povezani i potrebno je imati određene ugrađene sigurnosne značajke za sprečavanje neovlaštenog pristupa i zaštite podataka.

Upravljanje memorijom i podacima krucijalan je dio dizajna OS-a pošto se takvi sustavi generiraju velike količine podataka kojima je potrebno pristupati i vršiti određene analize kada se za time ukaže potreba. Iz tih razloga IoT operacijski sustavi moraju biti dizajnirani da budu učinkoviti u korištenju memorije, koja je kao resurs često ograničena. Obično koriste samo nekoliko stotina kilobajta RAM memorije. [15]

Slika 8: Karakteristike IoT OS-a
Izvor: https://blog.helpwire.app/iot-operating-systems/

Za takvu vrstu optimizacije koriste se tehnike kao što su *real-time-processing* koja omogućuje da se određeni zadatak izvrši u nekom specifično određenom vremenu. Druga tehnika je naglasak na modularnom dizajnu koji omogućuje da sustav bude lagan na način da su uključene samo najbitnije komponente.

Najčešće korištene arhitekture su upravo monolitni ili mikrojezgreni RTOS operacijski sustavi (eng. *Real-Time-Operating-System*). *Scheduler* ili planer jako je bitan dio svakog operacijskog sustava, a kada pričamo o dizjanu specifičnom za IoT, koriste se tzv. *Preemptive schedulers.* Takva vrsta algoritma omogućuje sustavu egzekuciju više procesa na način da dozvoljava procesoru prekidanje izvođenja trenutnog zadatka, alociranje resursa, i početak izvođenja nekog prioritetnijeg zadatka. Svaki proces ima određeni prioritet koji je određen prema važnosti ili hitnosti. Takve provjere i evaluacije su konstantne. *Time-slicing* jedna je od mogućnost takvih planera, u kojima se alocira određeno vrijeme CPU-a za izvršavanje nekog procesa koje se naziva *time-slice* ili vremenski kvantum (eng. *time quantum*). U slučaju da se zadatak ne izvrši u to određeno vrijeme, planer ga prekida i dodjeljuje resurse drugom zadatku. Na taj način osigurana je „poštena“ raspodjela vremena između više zadataka. [16]

Programski modeli koji se koriste za IoT osperacijske sustave su *event-driven protothreads*, *multi-threading* i *event-driven signle threading*. Prva vrsta, takozvane protoniti vođene događajima, tehnika je programiranja korištena kod sustava s jako ograničenim memorijskim resursima. To su najčešće ugrađeni sustavi IoT uređaja. U suštini protoniti su oblik kooperativnog *multitaskinga* koji omogućuje izvršavanje više uzastopnih niti unutar jedne niti

operacijskog sustava. Funkcioniraju na temelju eksplicitnog popuštanja (eng. *explicit yield*). Svaka protonit je zapravo serija ili niz instrukcija koje se izvršavaju dok se postigne točka popuštanja. Na toj točki protonit odustaje od kontrole i na taj način dozvoljava drugim protonitima, odnosno *event-handlerima* da se izvrše. Na taj način efikasno se koriste resursi bez troškova višedretvenosti.

Kod višedretvenosti, više niti mogu se izvršavati unutar jednog procesa. Svaka nit predstavlja niz instrukcija za izvršavanje. Dozvoljava paralelno izvršavanje čiji su benefit performanse, brzina odaziva i općenito korištenje višejezgrenih procesora. Svaka nit ima poseban stog i vlastite resurse. *Scheduler* ili planer određuje CPU vrijeme svakoj niti. Ovakav pristup zahtjeva korištenje sinkronizacijskih mehanizama kao što su zastavice i semafori za koordiniranje dijeljenih resursa kako bi se izbjegli uvjeti za „utrke“ između niti za resurse.

Kod programskog modela jednodretvenosti vođenih događajima, koji je poznat i kao *event-loop* ili *event-driven* programiranje, jedna je dretva odgovorna za izvršavanje više događaja. Funkcionira na način da program ulazi u *event-loop* gdje čeka da se neki događaj dogodi (pritiska gumba, dolazak podataka iz senzora ili paketa iz mreže). Kada je događaj detektiran, poziva se *event-handler* da taj isti događaj obradi. Nakon što se on obradi, program se vraća u petlju i čeka idući događaj. Ovaka pristup je jednostavan i efikasan u rukovanju asinkronih događaja. Međutim, kod dizajna ovakvih sustava potrebno je pripaziti na da se izbjegnu eventualna blokiranja i neke dugotrajne operacije koji bi potencijalno mogle utjecati na brzinu odaziva sustava.

5.3. Usporedba IoT operacijskih sustava

Kao što je već spomenuto, operacijski sustavi za IoT uređaje dizajnirani su da rade pod ograničenjima limitirane memorije i procesorske moći IoT uređaja. To nisu tipični operacijski sustavi i specijalno su dizajnirani za da pouzdano rade na IoT uređajima i podržavaju specifične *use-case* scenarije. Zapravo ne postoji standardizirani operacijski sustav koji takvi uređaji koriste. U idućim odlomcima opisani su neki od operacijskih sustava, njihova namjena kao i prednosti/nedostaci.

1. **Contiki OS** je open-source operacijski sustav za IoT uređaje koji je poznat po jednostavnom povezivanju malih i ekonomičnih mikrokontrolera koji nemaju veliku snagu s internetom. Contiki ima reputaciju jako korisnog sustava koji se koristi za kompleksne bežične sustave i memorijski je izuzetno efikasan. Njegov *scheduler*, odnosno planer izlaže veličine reda čekanja tako da aplikacija može staviti mikrokontroler u *sleep-mode*. Na taj način procesi za štednju energije postavljaju CPU u stanje mirovanja kada je red čekanja prazan. Mehanizam za postizanje rada niske snage naziva se ContikiMAC. Uz pomoć tog protokola, čvorovi i dalje primaju i prenose radio signale dok su u *low-power* modu. Uz to, koristi elegantan mehanizam za buđenje sustava. Periodično se čvorovi „probude“ i osluškuju radio kanal za prijam od susjednog čvora. U slučaju da signal postoji, čvor nastavlja osluškivati i priprema se za primanje paketa. Kada čvor zaprimi podatkovni okvir, šalje nazad okvir potvrde. Contiki Os dizajniran je programskim modelom koji se bazirana na protonitima ili protojezgrama (eng. *protothreads*). Kao pto je već opisano, radi se o memorijski efikasnoj programskoj apstrakciji koja dijeli značajke *event-driven* i *multithreading* koncepata i mehanizama za maksimalnu učinkovitost. *Protothreads* je mehanizam paralelnog programiranja gdje se više komputacija izvršava istovremeno, unutar vremenskih intervala koji se preklapaju, radije nego uzastopno jedna iza drugog. [17]
2. **FreeRTOS** Amazonov je open-source proizvod, odnosno OS otvorenog koda. Ima jako mali tzv. *memory footprint* i upravo to ga čini jako pogodnim za male mikrokontrolere gdje je memorija oskudna. Amazon je mnogo resursa uložio u sigurnost podataka. FreeRTOS ima ugrađen WFI (*Wait For Interrupt*). Ovaj mehanizam omogućava mikrokontroleru da ostane u dubokom stanju čuvanja energije sve dok se to stanje ne

prekine nekim vanjskim podražajem (eng. *interrupt*), ili dok RTOS kernel ne prebaci neki zadatak u stanje pripravnosti (eng. *Ready State*) [14]. Uz to, FreeRTOS koristi *Idle task hook* mehanizam koji postavlja mikrokontroler u *low-power* mod koji smanjuje potrošnju energije. To pomalo limitira energetsku učinkovitost jer sustav se periodički „budi“ i procesuira prekide, što je u suštini provjera ima li sustav što za napraviti. Ovaj OS isto tako osigurava stabilnost s LTC (eng. *Long Term Support*) podrškom, odnosno dugoročnom podrškom za njegove *library-e* koje održava AWS kao takav. Postoji velika zajednica okupljena oko ovog operacijskog sustava, postoji velik broj knjižnica s primjenama u mnogim industrijskim granama. To kombinirano s sigurnim povezivanjem na lokalnom i cloud nivou daje prednost ovom operacijskom sustavu u velikim međunarodnim kompanijama gdje se on koristi. Tablica ispod prikazuje listu provjera za kvalitetu koda LTS knjižnica. [19]

#	Category	Checks
1	Complexity Score	Functions shall have a GNU Complexity score less than 8.
2	Coding Standard	Functions shall comply with the MISRA 2012 coding standard.
3	Static Checking	Functions shall pass Coverity static checking.
4	APSEC review and pentest	Libraries must pass AWS security review.
5	Code Testing, including memory safety proofs	All code shall have extensive unit and function tests, with Gcov reports detailing test coverage, as well as CBMC memory safety proofs.
6	Requirements Documentation	All libraries shall have documented requirements, which may include resource, dependency, and porting requirements (as applicable).
7	Design Documentation	All libraries shall have design documentation, including application and cloud interface, state machines, and synchronization (as applicable).
8	Compiler Warning	The code shall compile with GCC using the -Wall and -Wextra command line options without generating compiler warnings.

Slika 9: Popis provjera kvalitete koda FreeRTOS knjižnica
Izvor: https://www.freertos.org

3. **RIOT** je operacijski sustav za IoT za kojeg se često kaže da je Linux IoT svijeta. Kao i brojni drugi operacijski sustavi za Internet stvari, ovaj OS je isto tako otvorenog koda. Postoji priključak koji omogućuje da ga se koristiti kao Linux i kao macOS proces. [18] *Scheduler* ovog OS-a predstavlja takozvani *tickless planer*, za razliku od FreeRTOS operacijskog sustava. Većina sustava periodički se „budi“ i provjera ima li zadataka

koristeći *timer tick*s mehanizme. Međutim, čest je slučaj da sustav nema zadataka i zapravo njegovo prebacivanje iz sustava čuvanja energije je ponekad bespotrebno. RIOT-ov *tickless scheduler* izbjegava takozvane *timerticks* i samim time povećava vrijeme provede u stanju mirovanja. Ovaj sustav isto tako predstavlja mehanizme za očuvanje energije i energetske učinkovitosti na razini mikrokontrolera. Sustav smanjuje brzinu rada procesora kada se određeni periferni uređaji ne koriste, a isto tako postoji i mogućnost isključenja napajanja za periferiju u situacijama kada je to potrebno. Uz to, ovim metodama moguće je postaviti memoriju u stanje niske potrošnje ili u ne odmah dostupno stanje. Isto tako, moguće je i isključiti napajanje određenim dijelovima memorije koji se ne koriste ili čak cijelom uređaju osim njegovog fizičkog pin-a, odnosno utora koji se koristi za njegovo „buđenje“. [20]

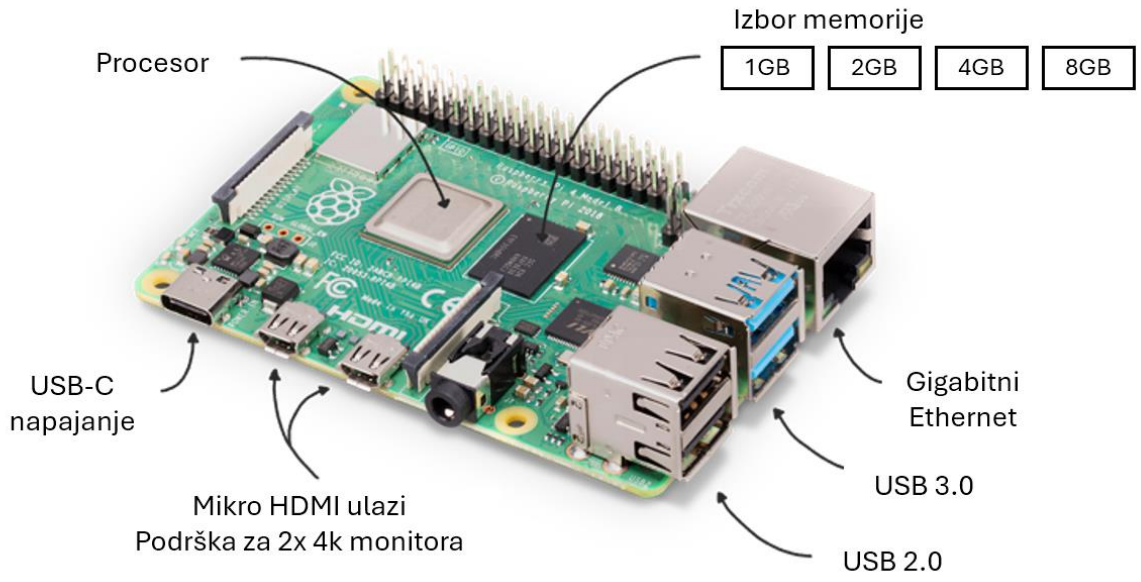
4. **TinyOS 19** je kao i mnogo prethodno opisanih operacijskih sustava za IoT, otvorenog koda. Tiny OS je ugrađeni (eng. *embedded*) operacijski sustav baziran na komponentama. Napisan je u nesC programskom jeziku, što možemo reći da predstavlja dijalekt C programskog jezika. Koristi se kao platforma za bežične uređaje male snage i uživa jako veliku potporu među developer-ima kao i veliku popularnost općenito. Dizajn ovog OS-a usmjeren je na komunikaciji i modularnosti programskih modula gdje se aplikacije i servisi distribuiraju preko skupova ograničenim resursima i uređaja koje prenose podatke iz i u fizički svijet preko mreže. U trenutku pokretanja aplikacije na uređaju, ona zatim radi istodobno s procesorom dok se ne generira odgovor. U međuvremenu, postoji mogućnost da drugi uređaji isto tako trebaju neki servis, i zahtijevaju od sustava „žongliranje“ s nekoliko tokova događaja. Inače OS koristi više *thread-ova* ili niti, svaki s svojim stogom. OS se prebacuje između niti, spremajući njihove registre, dok se niti koordiniraju međusobno koristeći zastavice i semafore. To zna biti problematično jer ponekad više niti trebaju podatke iz globalnih varijable i potencijalno komuniciraju međusobno. To može dovesti do utrka i zastoja procesa, a kompleksni planer mora zadovoljiti zadane rokove. Iz tih razloga Tiny OS koristi *event-driven execution*, odnosno izvršavanje potaknuto događajima. Tipičan je za ugrađene sustave (eng. *embedded systems*) jer jedan stog podržava više konkurentnih aktivnosti. Sustav zove specifične event-handlere ovisno o tipu događaja i svaki *handler* poduzima neku radnju i ažurira stanje sustava. Naravno, kasniji događaji dirigiraju iduće operacije. Nedostatak ovog pristupa je što je potrebno jako dobro

razumjeti sustav, napraviti i modificirati ovakav sustav, što za sobom povlači otežavanje ponovne uporabe koda i bilo kakvu prilagodbu. [21]

5. **Mbed-OS** još jedan popularan OS korišten u sklopu IoT-a. Dolazi s ugrađenim mehanizmima za uštedu energije. Sustavi koristi više modova; *Sleep mode, DeepSleep Mode, PowerDown i DeepPower Down* mod. Svaki od navedenih funkcionira slično, no treba napomenuti da prilikom *DeepSleep* i *DeepPower Down* modova, potrebno je više vremena da se uređaj ponovno uključi/probudi.[17] Postoji relativno velika zajednica okupljena oko ove vrste OS-a. Postoje brojne mogućnosti povezivanja kao što je Bluetooth, NFC, RFID, Ethernet, Wi Fi, Cellular… Operacijski sustav podržava velik broj *library-a* i postoji velik broj uputa, tutoriala i primjera za njegovo korištenje. OS spada u RTOS tip sustava (eng. *real-time-operating-system*) i podržava višejezgreno izvršavanje software-a u stvarnom vremenu. [22]
6. **MicroPython** je implementacija Python programskog jezika s fokusom za rad s mikrokontrolerima. Sadrži svoj kompajler i *runtime*. Odličan je izbor za početnike, dok je s druge strane dovoljno napredan da može imati primjenu i u nekim industrijama. Moguće ga je proširiti s nekim *low-level* funkcijama koje se koristi u C i C++, te na taj način proširiti relativno brz razvoj koji nudi viši programski jezik s nekim naprednijim funkcionalnostima koje nude programski jezici niže razine. [23]

6. Raspberry PI

Raspberry Pi je uređaj koji se naziva *single-bord computer*, odnosno računalo s jednom pločom. Zapravo za njega možemo reći da je malo, jeftino računalo koje se može priključiti na eksterni monitor/ekran/TV skupa s mišem i tipkovnicom. Nastao je kao projekt za promicanje učenja programiranja u osnovnim školama, međutim popularizirao se mnogo više od očekivanog. Počeo se prodavati na tržištima koja uopće nisu planirana tržišta prodaje, kao što je to recimo industrija robotike. Danas se jako široko koristi a popularan je upravo zbog svoje niske cijene, modularnosti, otvorenog dizajna i laganog povezivanja s vanjskim uređajima putem USB-a i HDMI-a. Pomoću njega mogućnosti su slične onima što bi se moglo očekivati od stolnog računala/laptopa. [24] Korisnici mogu surfati internetom, igrati igre, koristiti Office alate, slušati muziku, a pošto se primarno koristi kao uređaj za interakciju s vanjskim svijetom, korišten je u velikom broju raznih produkata. Neki od projekta koji su popularni za hobiste su Ad Blocker (bloker reklama na internetu), razni projekti s kamerom kao što su sigurnosne kamere (monitoriranje), botovi na internetu, primjerice Twitter ili Facebook botovi, mogućnost setup-a baze podataka na uređaj, itd.



Slika 10: Raspberry Pi 4
Izvor: https://assets.raspberrypi.com/static/raspberry-pi-4-labelled-f5e5dcdf6a34223235f83261fa42d1e8.png

Prvi puta je predstavljen 2012. godine, Raspberry Pi 1 Model B. Od tada na tržištu se promijenilo više modela kako je uređaj napredovao i razvijao se, a trenutno je Raspberry Pi 400 najnoviji model. Dolazi u kompaktnoj izradi kao tipkovnica i računalo u jednom.

Iako jako popularan među hobistima, Raspberry Pi koristi se i u mnogim industrijama u profesionalne svrhe. Imaju svoj cijeli program za poduzeća, a proizvode koriste njihovi partneri u preko 40 različitih zemalja. Postoji i njihovi zaštitni znak, *Powered by Raspberry Pi*, koji poduzeća mogu staviti na proizvode koji koristi Raspberry. Kompanija kroz svoj program za poduzeća nudi i pomoć pri integriranju Raspberry Pi uređaja u proizvode. Svi Raspberry proizvodi opsežno su dokumentirani i daju uvid kako svaki od njih funkcionira. U idućem odlomku napravljen je kratki pregled nekih od Raspberry Pi uređaja. [24]

6.1. Generacije Raspberry Pi uređaja

Raspberry Pi Zero malen je i jeftin uređaj odličan za male projekte koji ne uzimaju mnogo prostora. Nažalost, cijene uređaja jako su skočile početkom 2022. godine radi manjka i velike potražnje čipova globalno. U početcima, 2017. godine kada je lansiran, za otprilike 10$ mogao se dobiti čip s WiFi-jem i Bluetooth-om. Čip dolazi s 512MB RAM memorije. 5$ koštao je uređaj bez mogućnosti bežičnog spajanja na internet. Sada cijena varira između 35$ i 70$.

Raspberry Pi Model A serija fizički je nešto veći od Pi Zero i ima bolji i jači procesor. Iako ima isto RAM memorije kao i Pi Zero, dolazi s ugrađenim USB, HDMI i audio *port-ovima*. Kada je lansiran, mogao se naći za otprilike 25$, danas se cijene kreću između 70$ i 100$.

Raspberry Pi Model B serija naprednija je od A serije. Dolazi s četiri USB port-a, Ethernet port-om i noviji modeli čak podržavaju dualne monitore. Isto tako koristi četvernojezgreni procesor, a ovisno o modelu, može imati do 4GB RAM memorije. Iako to za današnje standarde ne zvuči primamljivo, treba uzeti u obzir da kada se tek pojavio na tržištu, za 35$ moglo se imati *quad-core* procesor i surfati na dva monitora.

Raspberry Pi 4 model bio je najjači model kojega je tvrtka lansirala. Jako je popularan, ima mogućnost proširenja RAM memorije do 8GB. Isto tako postoji mogućnost spajanja dva 4k monitora s čipom. Čip je moguće proširiti Micro-SD karticom za dodatnu memoriju, a isto tako

dolazi s Gigabitnim Ethernet-om. Također, napaja se preko USB-C port-a, što omogućuje dodatno povezivanje s eksternim, perifernim uređajima kada koristi adekvatno napajanje.

Raspberry Pi Pico maleni je čip kojemu je početni cijena bila 4$. Za manje od 30kn, moglo se dobiti uređaj s dvojezgrenim procesorom i 264kB RAM memorije i podrškom za do 16 MB flash memorije izvan čipa. Dolazi s bežičnim 2.4GHz LAN-om i Blutooth-om što ga čini, obzirom na cijenu, idealnim za neke male projekte.

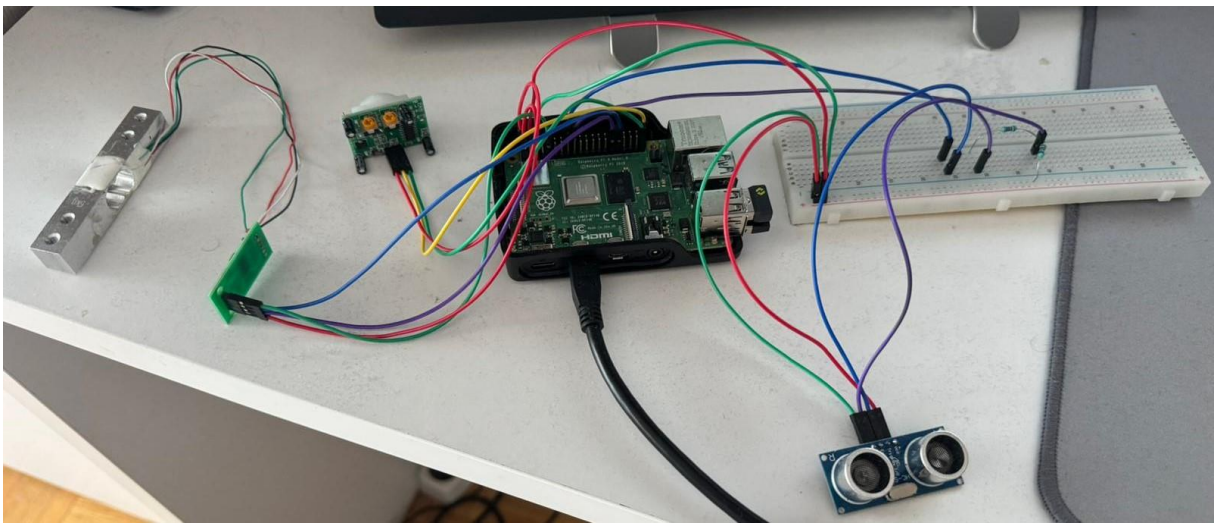
Raspberry Pi 400 najnoviji je model i dolazi u kompaktnom riješenu kao tipkovnica. Dolazi s četverojezgrenim 64-bitnim procesorom, 4GB RAM-a, mogućnosti bežičnog povezivanja s internetom i mogućnosti reprodukcije 4K videa. [25]

Slika 11: Raspberry Pi 400
Izvor: https://assets.raspberrypi.com/static/keyboard-lg-0e68b53708ad11b6dc0fff016f211a11.png

7. Praktični dio – Izrada pametne kante za smeće

Praktični dio ovog rada prikazuje izradu pametne kante za smeće (eng. *smart bin*). Ideja je na jednostavnom primjeru uredske kante za smeće prikazati osnovu ideju koja može služiti kao polazna točka boljeg waste managementa, odnosno upravljanja ili gospodarenja otpadom. Upravo se o upravljanju otpadom mnogo priča u zadnjih nekoliko godina, traže se alternative klasičnim rješenjima i dosta financijskih sredstava se ulaže u infrastrukturu i poboljšanje upravljanja otpadom kao grane javnih službi diljem svijeta.

Kroz ovo poglavlje detaljno su opisane faze izrade pametne kante za smeće, počevši od inicijalnog koncepta i dizajna, preko odabira potrebnih komponenti i tehnologija, pa sve do implementacije rješenja korak po korak. U pogledu hardverskog dijela, opisane su karakteristike korištenih senzora za detekciju razine otpada, te načini njihove integracije u kantu za smeće. U programskom dijelu, objašnjena je implementacija potrebnog softverskog sustava, uključujući načine funkcioniranja korištenih senzora, povezivanje s mikrokontrolerom, i na kraju povezivanje s vanjskim sustavom, odnosno web aplikacijom.



Slika 12: Finalna konfiguracija rješenja
Izvor: vlasitita izrada

Odabrani operacijski sustav za izradu ovog projekta je Linux, odnosno Raspbian Pi OS razvijen specifično za Raspberry Pi mikrokontrolere. Raspbian je besplatan OS temeljen na Debian distribuciji napravljen konkretno za Raspberry Pi uređaje. Uz operacijski sustav, on dolazi s preko 35.000 paketa različitog software-a i za jednostavnu instalaciju. Upravo radi

jednostavnosti, optimizacije i velikog broja unaprijed instaliranih programa koji dolaze s ovim OS-om, izbrana je za implementaciju ovog projekta. Neke od glavnih značajki ovog OS-a su:

- **Optimizacija** – prilagođen specifično za ARM arhitekturu Raspberry PI-a s naglaskom na učinkovito iskorištavanja ograničenih resura uređaja
- **Sučelje** – LXDE desktop okruženje (*Lightweight X11 Desktop Environment*) besplatno i jednostvano za korištenje s relativno malim zahtjevima za resursima [26]
- **Programi** – velik broj predinstaliranih i edukativnih alata kao što su programska okruženja, Chromium web preglednik i razni uredski alati
- **Upravljanje** – sadrži APT (Advanced Package Tool) sustav koji se koristi za instalaciju, ažuriranje i brisanje programskih paketa na Debian baziranim Linux distribucijama
- **Podrška** – opsežna dokumentacija s velikim brojem tutoriala dostupnih na službenoj web stranici kao i široka zajednica

7.1. Senzori

Za izradu pametne kante za smeće, korišteni su tri različita tipa senzora kako bi se osigurala precizna i sveobuhvatna analiza otpada. Prvo, senzor pokreta omogućuje otkrivanje aktivnosti kada je kanta u upotrebi, odnosno kada se nešto u kantu odlaže. što pruža korisne informacije o vremenskim intervalima kada se kanta koristi. Zatim, senzor za mjerenje težine omogućuje precizno praćenje mase otpada unutar kante. Na kraju ultrazvučni senzor mjeri udaljenost između poklopca kante za smeće i tvari koje se u njoj nalaze. Ideja je da se spomenuti parametri šalju prema web aplikaciji, koja će iste prikazivati u realnom vremenu, s mogućnosti praćenja povijesnih podataka kroz vrijeme.

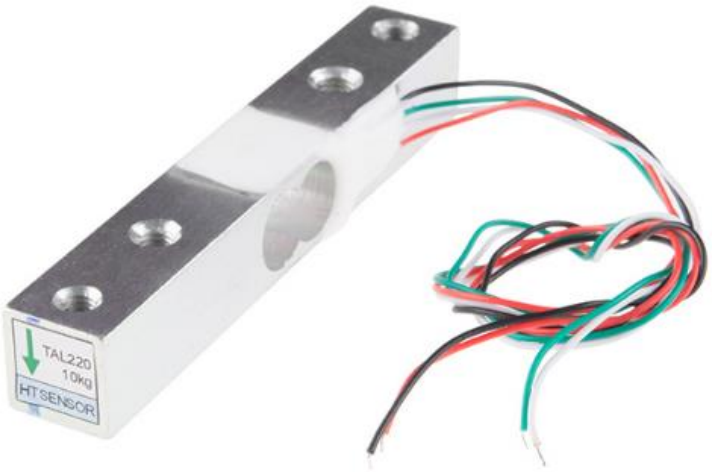
Slika 13: Senzor pokreta
Izvor: https://www.graylogix.in/wp-content/uploads/2021/06/PIR-Sensor-2.jpg

Senzor pokreta ili PIR sezor (eng. *Passive Infrared*), što stoji za pasivne infracrvene senzore, široku su korišteni uređaji specifično dizajnirani za detekciju kretanja preko infracrvenog zračenja koje emitiraju objekti. Dakle, svaki objekt s temperaturom koja je iznad apsolutne nule emitira spomenuto infracrveno zračenje, a senzori su podešeni da detektiraju ta zračenja. Jezgre ovih senzora sastoje se od piroelektričnih elemenata koji mjere promjene u infracrvenim razinama. Prilikom kretanja unutar vidnog polja senzora, uzrokuje se promjena infracrvenog zračenja i ti ju elementi detektiraju, i pri tome stvaraju signal. Obradom signala označuje se kretanje. Takvi senzori obično su opremljeni Fresnelovom lećom (eng. *Fresnel lens*) koja fokusira infracrvene signale ne senzor i pri tome povećava domet i osjetljivost.

Obzirom na pouzdanost, nisku cijenu i relativno malu potrošnju energije, ovakvi senzori imaju veliku primjenu. Neki od primjera korištenja ovakvih senzora su sigurnosni sustavi za

otkrivanje uljeza, sustavima vezanim za pametni dom (eng. *smart home*) itd. Broj lažnih alarma je minimalan, pod uvjetom da su ispravno instalirani i nisu izloženi vanjskim faktorima koji bi mogli uzrokovati smetanje, kao što su brze promjene u temperaturi ili jaka sunčeva svjetlost.

Senzor za mjerenje težine jedna je od funkcionalnosti pametne kante za smeće. On omogućuje precizno praćenje razine otpada unutar kante, pružajući korisne informacije o težini otpada. Ta informacija može biti korisna za analizu obrasca odlaganja otpada ili za procjenu količine otpada koja se generira u određenom vremenskom razdoblju.



Slika 14: Senzor težine
Izvor: https://cdn.sparkfun.com/assets/learn_tutorials/3/8/2/13329-01Crop.jpg

Ključni senzor za ovaj projekt je ultrazvučni senzor koji omogućuje mjerenje postotka ispunjenosti kante. Ovaj senzor koristi se za precizno određivanje količine otpada u kanti, pružajući korisnicima jasne informacije o stanju napunjenosti. Ta informacija je od vitalnog značaja za efikasno upravljanje i praćenje kapaciteta kante, omogućujući optimalno planiranje i organizaciju procesa pražnjenja.

Za određivanje i mjerenje udaljenosti ultrazvučni senzor koristi visokofrekventne zvučne valove. Senzor se sastoji do odašiljača i prijamnika. Raspon valova koje emitiraju ovi senzori su od 20kHz do 40kHz. Kada emitirani valovi iz odašiljača naiđu na objekt, reflektiraju se nazad to prijamnika. Izračunom vremena koje je potrebno valovima da prevale put između trenutka kada su detektirali objekt, do trenutka kada su se vratili do prijamnika, određuje se precizna udaljenost do objekta.

Upravo radi točnosti i preciznost, kao i mogućnosti mjerenja udaljenosti bez izravnog i fizičkog kontakta, primjena ove vrste senzora, isto kao i senzori za detekciju pokreta, naširoko su korišteni. Primjerice senzori za parkiranje u automobilima, u raznim robotima za navigaciju, i jako su pogodni za mjerenje razine tekućine u spremnicima i silosima. Ključna prednost su upravo radne okoline gdje senzori ovog tipa mogu djelovati, a to su i prašnjava okruženja u kojima druge vrste senzora, primjerice optički, zakazuju. Ipak, materijalni koji imaju mogućnost apsorpcije zvučnih valova predstavljaju problem i smanjuju točnost mjerenja.

Slika 15: Ultrazvučni senzor
Izvor:
https://cdn.shopify.com/s/files/1/0559/1970/6265/files/What_is_Ultrasonic_Sensor_fac21db5-c2aa-4894-b7e5-05aa9a1bb468_480x480.png?v=1662815986

Kroz iduća poglavlja opisana je implementacija programskog rješenja za ono što ćemo nazvati pametnom kantom za smeće. Ideja je konfigurirati senzore na način da prilikom detekcije pokreta, senzor mjeri udaljenosti što predstavlja punoću kapaciteta kante, senzor težine mjeri koliko su teški predmeti odloženi u kantu, i naravno mjeri se broj objekata koji se u kanti nalaze. Uz sve te parametre možemo i pretpostavljati koji su predmeti u kantu bačeni. Primjerice, ako netko u kantu ubaci recimo ciglu, vidjeti ćemo da je broj objekata mali, s velikom težinom prisutnom u kanti i, ovisno o kapacitetu kantu, možemo zaključiti da se radi o nekoliko malih, težih predmeta koji zauzimaju malo mjesta, ili recimo velik broj laganih objekata koji zauzimaju dosta kapaciteta, što bi zapravo bio indikator kućnog otpada.

Temeljem tih podataka koji se mjere u stvarnome vremena, možemo poduzimati određenje radnje, i spremati ih za kasnije, gdje se ti podaci mogu pratiti kroz vrijeme i raditi određene analize. Primjerice, tokom ljetnih mjeseci možemo očekivati trend smanjene količine otpada radi godišnjih odmora i odlaska na ljetovanja. Tokom proljetnih mjeseci možemo očekivati veći

broj glomaznijeg otpada koji se akumulirao tokom zimskih mjeseci koji su popraćeni raznim praznicima (Božić, Nova godina) i pretežito su to mjeseci toplijeg vremena kada ljudi kreću u preinake i adaptacije stanova, uređivanje vrtova i voćnjaka, itd.

7.2. Implemetacija rješenja

Kroz ovo poglavlje opisana je implementacija opisanog rješenja. Detaljno su opisane faze izrade, popraćene slikama i objašnjenjima programskog koda, kao i koracima kako spojiti sam hardware i senzore s mikrokontrolerom. Započeti ćemo s spajanje i konfiguriranjem senzora za detekciju pokreta.

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)

motionPin = 21
GPIO.setup(motionPin, GPIO.IN)

def motion_detection(channel):
    if GPIO.input(channel):
        print("Motion detected")
    else:
        print("No motion detected")

GPIO.add_event_detect(motionPin, GPIO.BOTH, callback=motion_detection)

try:
    while 1:
        time.sleep(0.1)
except KeyboardInterrupt:
    print("Quit")
    GPIO.cleanup()
finally:
    GPIO.cleanup()
```

Slika 16: Programski kod senzora pokreta
Izvor: vlasitita izrada

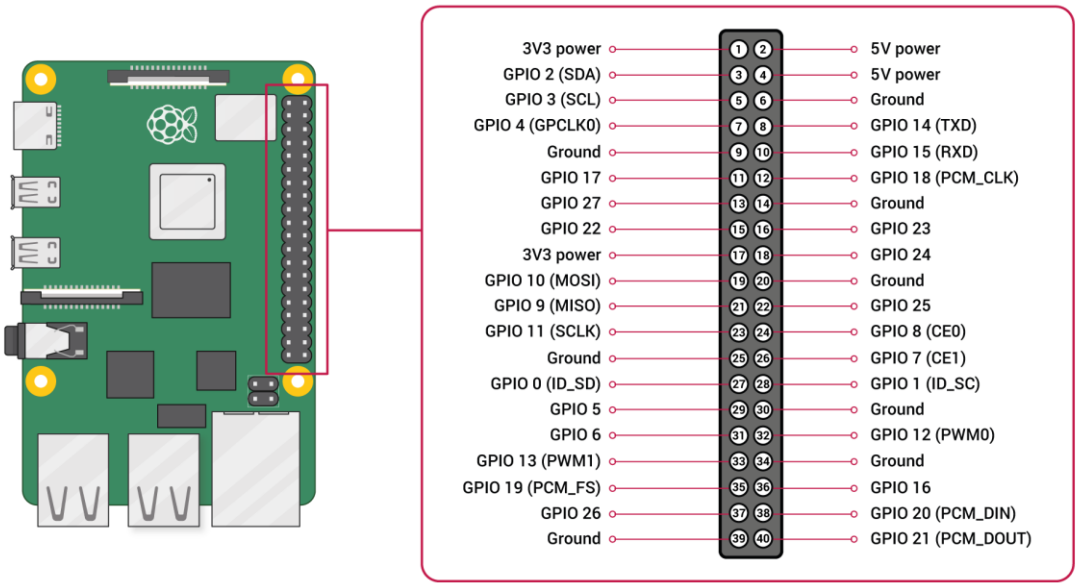
Za početak potrebno je importa-ti GPIO knjižnicu. Ona se koristi za kontroliranje GPIO pinova na uređaju.

GPIO.setmode funkcija određuje način na koji se referenciraju pinovi na uređaju. Postoje dva različita načina:

- GPIO numeriranje – koristi se i BCM naziv (Broadcom SOC Channel), kod ove metode brojevi pinova odgovaraju tzv. Broadcom sustavu na čipu koji se koristi kod Raspberry PI. Ova metoda temelji se na internim vezama, što ju čini konzistentnom preko različitih Raspberry Pi modela, čak i u slučajevima gdje se raspored pinova mijenja

- Fizičko numeriranje – metoda koja se odnosi na fizičku lokaciju pinova na ploči. Numeracija se radi uzastopno, od 1 do 40, počevši od gornjeg lijevog pina. Metoda je dosta izravna i jednostavna, međutim gubi se konzistencija između različitih modela uređaja.

U ovom slučaju postavljena ja na prvi način, odnosno na BCM, upravo radi konzistencije i izravne korelacije s SoC dokumentacijom, koja je korisna za programiranje niske razine i otklanjanje pogrešaka. Iduća slika pokazuje numeriranje ovom metodom.



Slika 17: GPIO pinovi numeriranje
Izvor: https://www.raspberrypi.com/documentation/computers/images/GPIO-Pinout-Diagram-2.png?hash=df7d7847c57a1ca6d5b2617695de6d46

Nakon toga definiramo GPIO pin 21 kao pin koji je spojen na senzor kretanja i definiramo funkciju koja prima parametar, i provjerava trenutnu ulaznu vrijednost navedenog GPIO pina. Ako je pin HIGH, otkriveno je kretanje i uvjet je istinit.

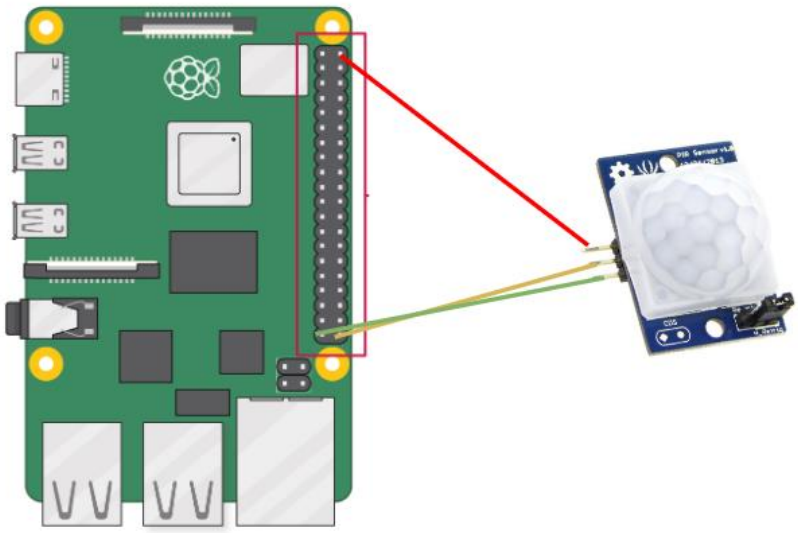
GPIO.add_event_detect(motion_pin, GPIO.BOTH, callback=motion_detection)

Definiramo event na pinu koji smo odredili na početku programa.

GPIO.BOTH argument specificira da se detekcija mora aktivirati, odnosno da se funkcija *motion_detection* mora pozivati u trenutcima kada je kretnja detektirana (LOW to HIGH) ili kada kretnja nije detektirana (HIGH to LOW).

Na kraju specificiramo da beskonačnu petlju s kojom konstantno provjeravamo ima li kretnje ili ne. korisnik želi zaustaviti program, može pritisnuti *Ctrl+C*, što pokreće iznimku *KeyboardInterrupt*, ispisuje poruku o zatvaranju i čisti GPIO postavke.

Na idućoj slici nalazi se shema spajanja Raspberry Pi uređaja s senzorom. Senzor se napaja preko 5 volti (pin 2), uzemljenje je spojeno na pin 39, a GPIO pin 21 se koristi za signal.



Slika 18: Shema spajanja senzora pokreta
Izvor: vlasitita izrada

Na idućoj slici nalazi se programski kod za konfiguriranje ultrazvučnog senzora.

```
import RPi.GPIO as GPIO
import time

TRIG = 23
ECHO = 24

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)
GPIO.output(TRIG, False)

def get_distance():
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)

    while GPIO.input(ECHO) == 0:
        pulse_start = time.time()
    while GPIO.input(ECHO) == 1:
        pulse_end = time.time()

    pulse_duration = pulse_end - pulse_start
    distance = pulse_duration * 17150
    distance = round(distance, 2)
    return distance

try:
    while True:
        distance = get_distance()
        print("Distance:", distance, "cm")
        time.sleep(1)
except KeyboardInterrupt:
    print("Quit")
finally:
    GPIO.cleanup()
```

Slika 19: Programski kod ultrazvučnog senzora
Izvor: vlasitita izrada

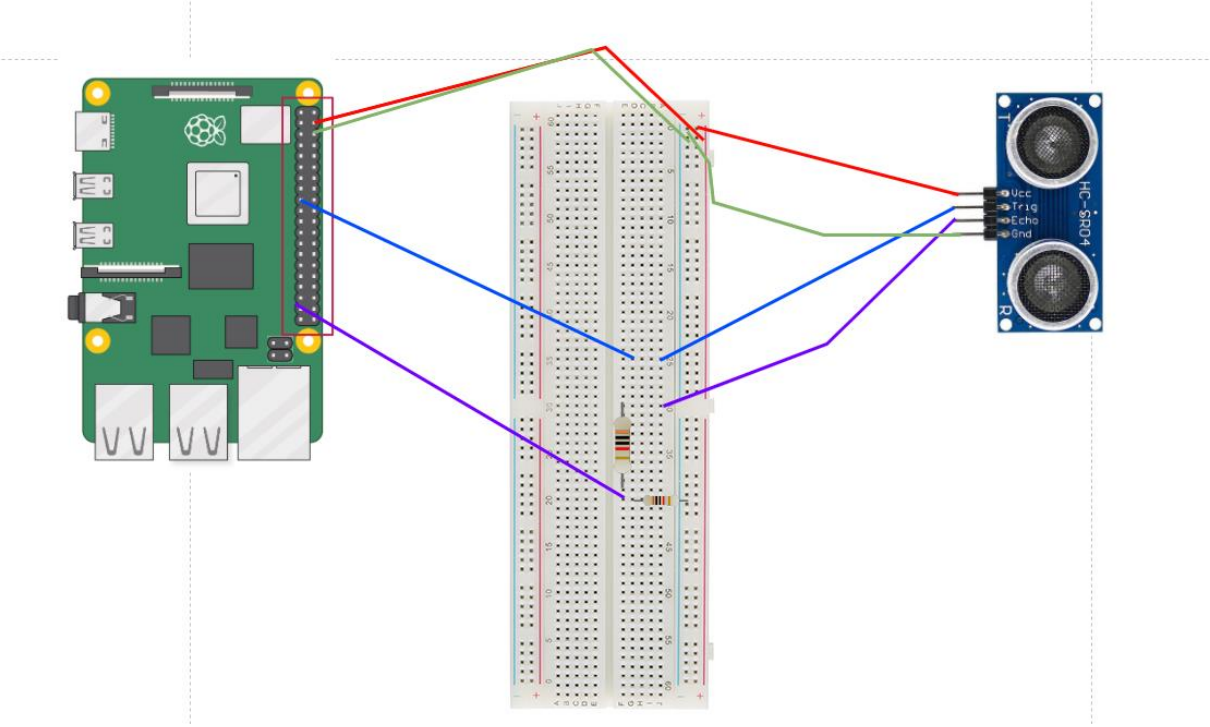
U početku import-amo potrebne knjižnice i postavljamo dvije konstante, pin za okidanje (eng. *Trigger pin*) na GPIO pin 23, i pin za jeku (eng. *Echo pin*) na GPIO pin 24. Nakon toga postavljamo numeraciju GPIO pinova na BCM način rada kao i u prethodnom primjeru konfiguriranja detekcije pokreta. Uz to, postavljamo TRIG kao izlazni, i ECHO kao ulazni pin. Odnosno, GPIO.OUT omogućuje Raspberry-ju da mjenja njegovo stanje (HIGH-LOW) i pri

tome šalje signale prema ostalim komponentama (uključivanje/isključivanje LED svijetla, slanje signala s ultrazvučnog senzora). GPIO.IN omogućuje Raspberry-ju da „čita“ odnosno prima signale od vanjskih komponenti, i time zapravo mjeri vrijeme potrebno da se signal vrati do ultrazvučnog senzora.

Konfiguracijom In i OUT pinova omogućili smo Raspberry Pi-ju da pošalje ultrazvučni impuls i izmjeri vrijeme koje je potrebno da se odjek vrati, što se zatim koristit za izračunavanje udaljenosti do objekta. ***GPIO.output(TRIG, False)*** postavlja *trigger* pin na LOW, odnosno da je inicijalno isključen (False).

Get_distance funkcija postavlja TRIG pin na HIGH stanje, odnosno inicira proces slanja ultrazvučnog signala. Nakon toga pauziramo program na 10 mikrosekundi, što je taman vremena da ultrazvučni senzor pošalje signal prema van, prije nego li vratimo stanje na LOW, što zapravo signalizira završetak slanja ultrazvučnog signala.

Idući korak je provjera stanje ECHO pina. Kada je on u LOW stanju, postavljamo početak mjerenja vremena s *time.time()* metodom. U trenutku kada se on postavi na HIGH stanje, znamo da se signal vratio nazad do ultrazvučnog senzora i taj trenutak označujemo kao kraj mjerenja vremena. Nakon toga zapravo je potrebno samo izračunati vrijeme potrebno do trenutka kada je signal poslan iz ultrazvučnog senzora, do trenutka kada smo ga ponovno primili. Vrijeme je potrebno parsirati radi lakšeg pregleda. Brzina zvuka u zraku je otprilike 343 metra u sekundi, što je 34300 centimetara u sekundi. Budući da puls putuje do objekta i natrag, dijelimo s 2, što rezultira 17150. Množenjem trajanja s 17150 dobivamo udaljenost u centimetrima. Na kraju kroz beskonačnu petlju mjerimo udaljenost, i definiramo kraj programa na *user* akciju, nakon čega se pinovi postavljaju na *default* stanja.



Slika 20: Shema spajanja ultrazvučnog senzora
Izvor: vlasitita izrada

Na slici iznad nalazi se shema spajanja ultrazvučnog senzora s uređajem. Za spajanje koristit ćemo *Breadboard*. Razlog su dva otpornika koja su potrebna za pravilno korištenje senzora. Otpornici su potrebni jer ECHO pin, koji je izlazni pin emitira signal od 5 volti kada detektira dovoljno puls. GPIO pinovi su dizajnirani za rad na 3.3 volti, a primjena napona viših od 3,3 V može oštetiti pinove. Iz tih razloga potrebno smanjiti signal s 5 V na 3,3 V pomoću razdjelnika napona napravljenog od otpornika.

Razdjelnik napona je jednostavan sklop koji koristi dva otpornika u seriji za smanjenje napona. Za izradu razdjelnika koristimo iduću formulu za kalkulaciju:

$$V_{out} = V_{in} \times \frac{R2}{(R1 + R2)}$$

Gdje su:

- V_{in} – ulazni napon (5 V)
- V_{out} – izlazni napon (3,3 V)
- R1 – otpornik spojen na ulazni napon

- R2 – otpornik spojen na izlazni napon

R1 nam predstavlja otpornik od 1kΩ, dok je R2 otpornik od 2kΩ. Ako te vrijednosti uvrstimo u jednadžbu dobijemo:

$$V_{out} = 5\text{V} \times \frac{2000\Omega}{(1000\Omega + 2000\Omega)}$$

$$V_{out} \approx 3{,}33\text{V}$$

Ta vrijednost je zadovoljna i nemamo rizika s korištenjem ultrazvučnog senzora s Raspberry Pi-jem u toj konfiguraciji.

Vcc, odnosno napajanje senzora je spojeno na pin 4 na mikrokontroleru. Uzemljenje je spojeno na pin 6. Trig pin, odnosno onaj koji šalje puls spojen je na GPIO 23, a Echo pin koji prima signal na GPIO pin 24.

```
import time
import RPi.GPIO as GPIO
from hx711 import HX711
import statistics

GPIO.setmode(GPIO.BCM)
hx = HX711(dout_pin=6, pd_sck_pin=5)

def calculate_weight(hx):
    try:
        weight = hx.get_weight_mean()
        calculated_weight = weight * 1
        print("Weight:", calculated_weight)
    except statistics.StatisticsError:
        print("Not enough data points for calculation")
        return None

try:
    while True:
        calculate_weight(hx)
        time.sleep(0.1)
except KeyboardInterrupt:
        print("Quit")
finally:
    hx.power_down()
    GPIO.cleanup()
```

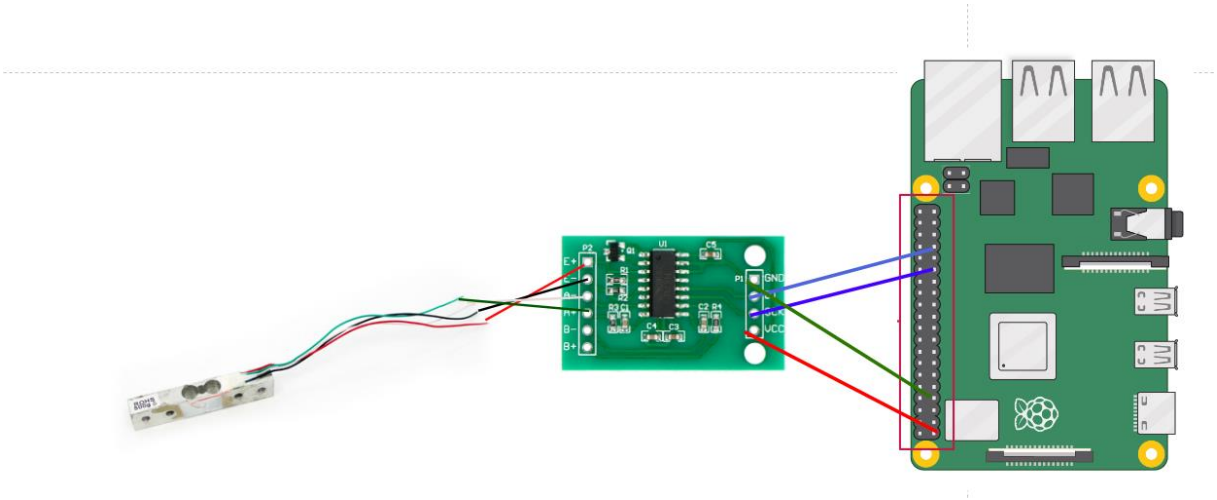
Slika 21:Programski kod senzora težine
Izvor: vlastita izrada

Ovog puta, uz standardne library-je, potrebno je i importati HX711 klasu iz hx711 knjižnice. HX711 je zapravo 24-bitni analogno-digitalni konverter dizajniran za skaliranje težine. Uz to, importamo i statistički modul, obzirom da postoji mogućnost statističke pogreške prilikom mjerenja težine radi očitavanja podataka s senzora. To je potrebno jer je senzor težine zapravo spojen na HX711 čip, koji služi kao pojačivač signala prema Raspberry Pi uređaju jer su incijalni signali koje senzor šalje preslabi.

hx = HX711(dout_pin=6, pd_sck_pin=5) kreira instancu HX711 klase i konfigurira pin 6 kao data output pin, i pin 5 kao tzv. clock signal koji služi za dohvaćanje podataka i isključivanje HX711 čipa kada se on ne koristi.

Funkcija *calculate_weight* prima hx objekt kao argument, što predstvalja instancu HX711 klase. Funkcija služi za očitavanje mase i *handling* potencijalnih greški. *Get_weight_mean* metoda hx objekta služi za očitavanje više očitanja s HX711 čipa, i kalkulira prosjek (eng. *mean*) težina. Taj princip minimizira smetnje i poboljšava točnost mjerenja.

statistics.StatisticsError se okida ako ne postoji dovoljno podataka za kalkuliranje prosjeka težine. Kako se program ne bi raspao, ovo služio kao *error handling* da se ispiše kako nema dovoljno podataka za kalkulacije, a program nastavlja dalje s novim računanjem.



Slika 22: Shema spajanja senzora težine
Izvor: vlasitita izrada

Senzor težine spajamo na HX711 čip koji služi kao pojačalo signala koji izlazi iz senzora. Uobičajeni kod boja je:

- Crvena žica: E+
- Crna žica: E-
- Bijela žica: A-
- Zelena žica: A+

VCC je napajanje i njega spajamo na pin 1 na mikrokontroleru. Uzemljenje spajamo na pin 9. SCK za slanje taktnih signala za sinkronizaciju prijenosa podataka spajamo na GPIO pin 6, a DT, koji šalje digitalne podatke o težini s HX711 do Raspberry-ja spajamo na GPIO pin 5.

7.2.1. Notifikacije i obavijesti

Obavijesti su jedna od značajki ovog projekta. Ideja je slati obavijesti kada se kanta napuni do određenog kapaciteta koji je zadan. To se radi na temelju izračuna ultrazvučnog senzora, koji vraća udaljenost u centimetrima između poklopca kante i njenog sadržaja. Notifikacije dolaze u sklopu slanja SMS poruke na mobilni uređaj, kao i slanja email poruke.

Za slanje SMS notifikacija koristiti ćemo Twillio. Twillio je komunikacijska platforma za skaliranje i upravljanje komunikacijskim rješenjima. Pruža set API-ja koji omogućuju aplikacijama da primaju i upućuju pozive, SMS poruke i razne komunikacijske funkcije. Glavne značajke Twillio platforme su:

- Slanje i primanje SMS poruka globalno, s mogućnosti dvosmjerne komunikacije i SMS notificiranja
- Mogućnost glasovnih poziva (slanje i primanje), snimanje poziva i pretvaranje teksta u govor
- Programibilna kontakt centar platforma s mogućnošću prilagodbe rješenje
- Slanje i primanje WhatsApp poruka, s mogućnosti slanja multimedijskih poruka i dijeljenja lokacijskih podataka
- Podrška za grupne video pozive, dijeljenje i snimanje zaslona
- Slanje e-pošte, s predlošcima i mogućnosti praćenja i analitike

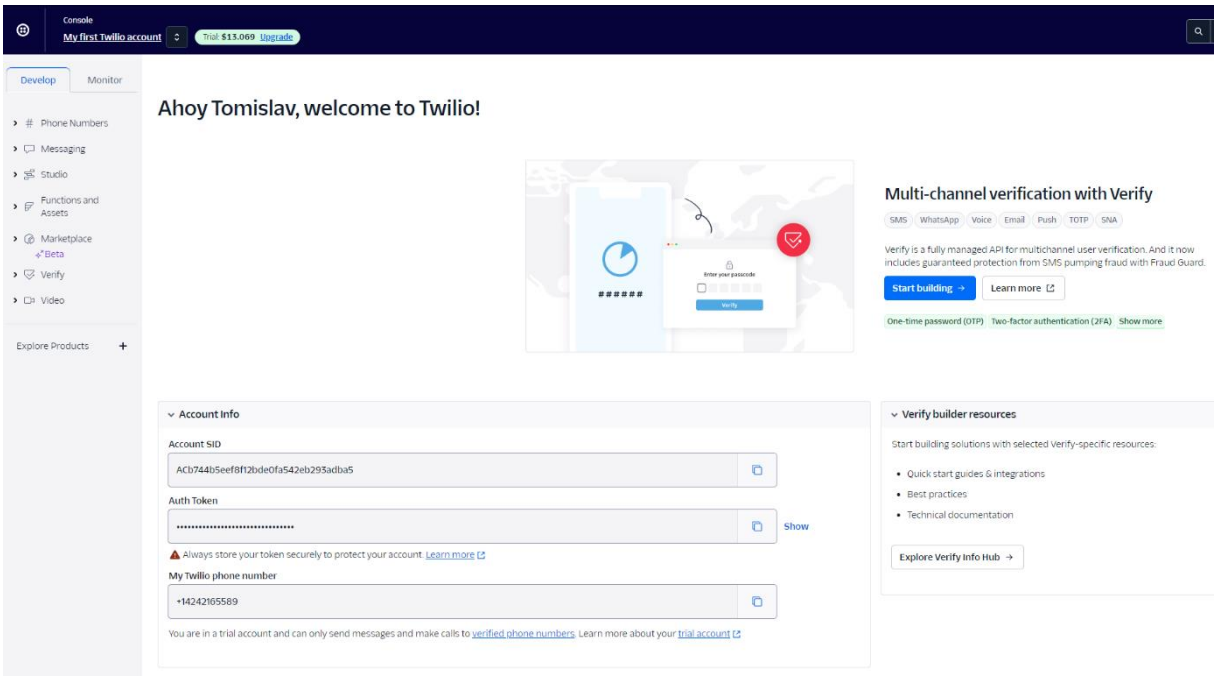
```
def send_email_notification():
    sender_email = 'rpiberry4@gmail.com'
    reciever_email = 'rpiberry4@gmail.com'
    email_password = 'trpczprlvrqequhh'
    subject = 'Trash bin capacity'
    body = "Your trash bin is almost full!"
    email_message = f"Subject: {subject}\n\n{body}"
    server = smtplib.SMTP("smtp.gmail.com", 587)

    try:
        server.starttls()
        server.login(sender_email, email_password)
        server.sendmail(sender_email, reciever_email, email_message)
        print("Email sent to: " + reciever_email)
    except:
        print("Email not sent.")
    finally:
```

```
        server.quit()

def send_sms_notification(weight, distance):
    message = f"Your bin is almost full!. Weight: {weight} g, Empty
space: {distance} cm"
    try:
        message = client.messages.create(
    body= message,
    from_=keys.twilio_number,
    to=keys.target_number
    )
        print("SMS sent to:", keys.target_number)
    except Exception as e:
        print("SMS not sent. Error:", str(e))

```

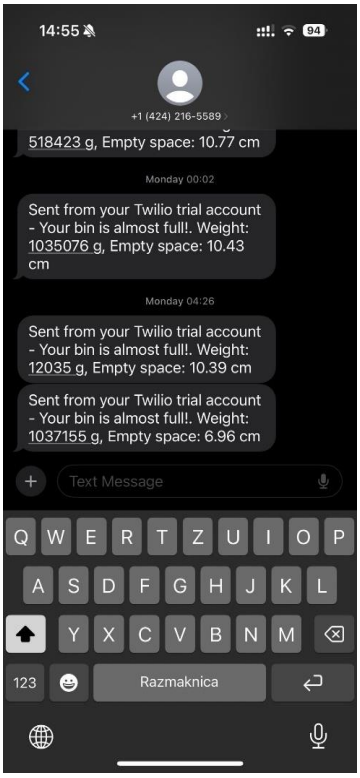
Slika 23: Programski kod slanja notifikacija
Izvor: vlastita izrada

Za početak potrebno se prijaviti i kreirati račun na Twillio web stranici (https://www.twilio.com/en-us). Nakon izrade račun imamo dostupnih 15$ za kupovinu broja s kojega ćemo primati SMS poruke. Cijene brojeva kreću se od 1.5$ do 15$+. U sklopu ovog projekta kupljen je američki broj.

Prvi korak je kreiranje *keys.py* datoteku. Unutra se nalaze parametri potrebni za slanje SMS poruka. *Account_number* ili account SID je jedinstveni identifikator Twillio računa. On određuje s kojim računom radimo, odnosno prilikom API zahtjeva prema Twillio-u naznačujemo da zahtjev dolazi s našeg računa. *Autetntication_token* je tajni ključ našeg računa za provjeru autentičnosti API zahtjeva kako bismo autorizirali zahtjeve da zapravo dolaze od nas. Ovaj se parametar koristi za sigurnost i da se onemogući neautorizirani pristup našem računu. Idući parametri su *twillio_number*, odnosno broj s kojega se SMS poruke šalju, i *target_number* što predstavlja broj na kojega primamo poruku.



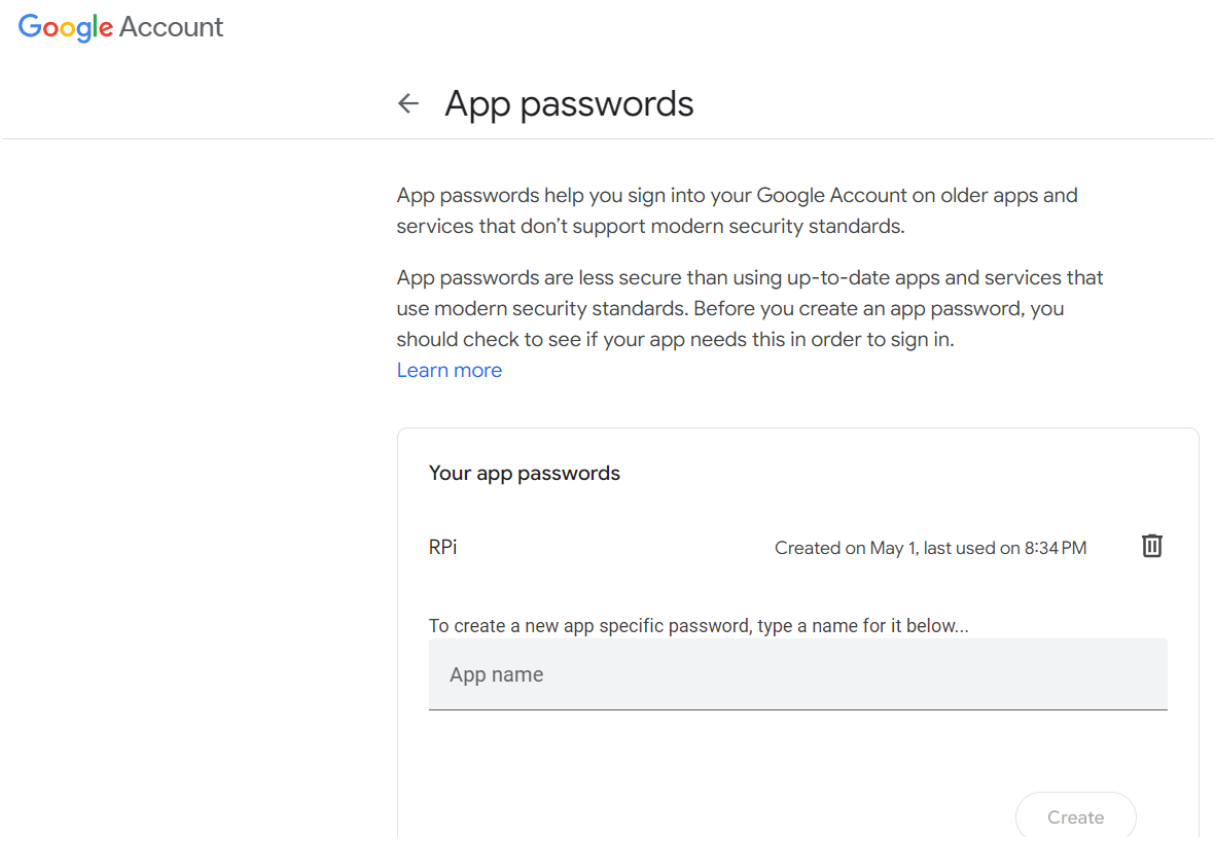
Slika 24: Twillio račun,
Izvor: https://www.twilio.com/en-us



Slika 25: Primjer SMS obavijesti
Izvor: vlastita izrada

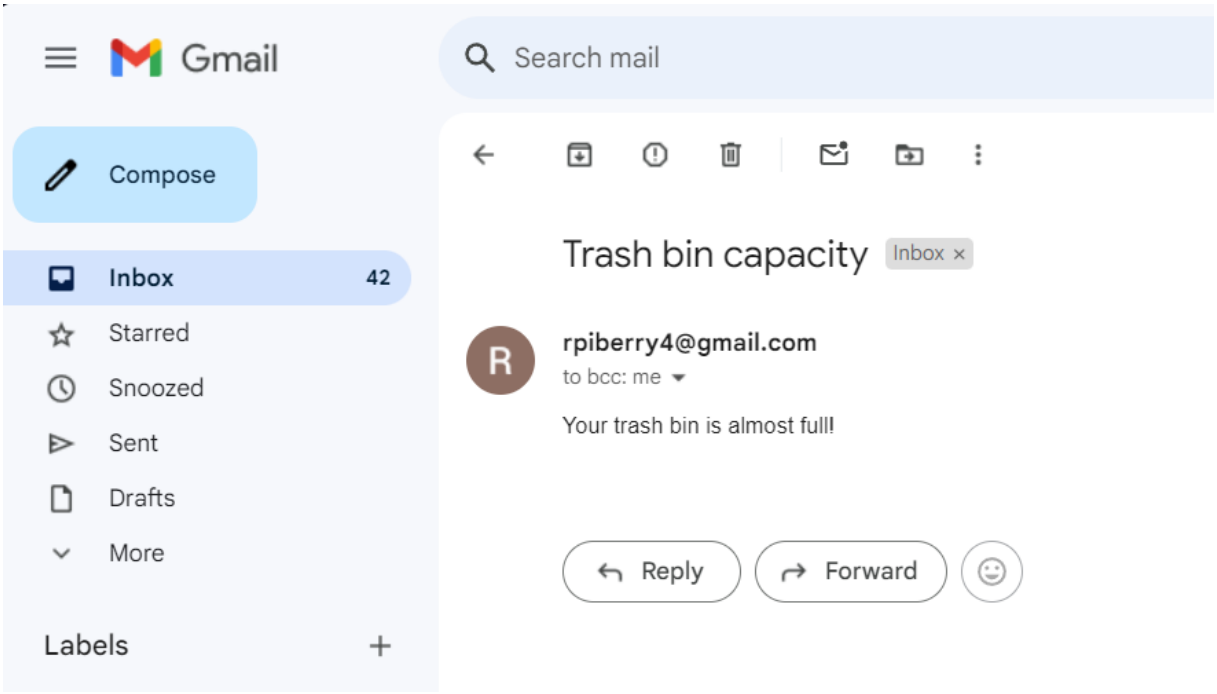
Za slanje E-mail notifikacija koristit ćemo *smtplib* knjižnicu. Ona se koristi za slanje mailova preko *Simple Mail Transfer Protocol*, ili SMTP protokola. Omogućuje funkcionalnost slanja mailova preko Python skripti povezujući ju s SMPT serverom. Može se koristiti u kombinaciji s email knjižnicom za kompoziciju kompleksnijih poruka s HTML sadržajem ili privitcima. Podržava različite načine autentikacije, što omogućuje interakciju s različitima SMTP serverima, uključujući one koji koriste SSL/TLS za sigurnu komunikaciju.

Za implementaciju definirali smo mail s kojega šaljemo poruku i email na koji šaljemo poruku (u ovom primjeru to je ista email adresa). Isto tako definirali smo naslov i sadržaj, odnosno tijelo poruke. U *email_password* varijablu pospremili smo sigurnosni ključ, pošto se nekada login odrađivao preko lozinke za prijavu, sada je potreban sigurnosni ključ. Do sigurnosnog ključa dolazimo kroz postavke Google računa, u *App passwords* sekciji. Potrebno je omogućiti dvofaktorsku autentikaciju, i definirati novi uređaj za kojega dobijemo generiranu lozinku.



Slika 26: Definiranje lozinke za slanje e-mail poruka
Izvor: https://www.google.com/gmail/about/

U server smo spremili SMTP objekt koji se povezuju s Gmail SMTP poslužiteljem na portu 587. U *try* bloku definiramo sigurnosnu šifriranu TLS vezu, i radimo login na e-mail adresu pošiljatelja koristeći definiranu lozinku i samu adresu. U idućim koracima definiramo i šaljemo samu email poruku, i ovisno o uspjehu vraćamo van tekstualnu poruku.



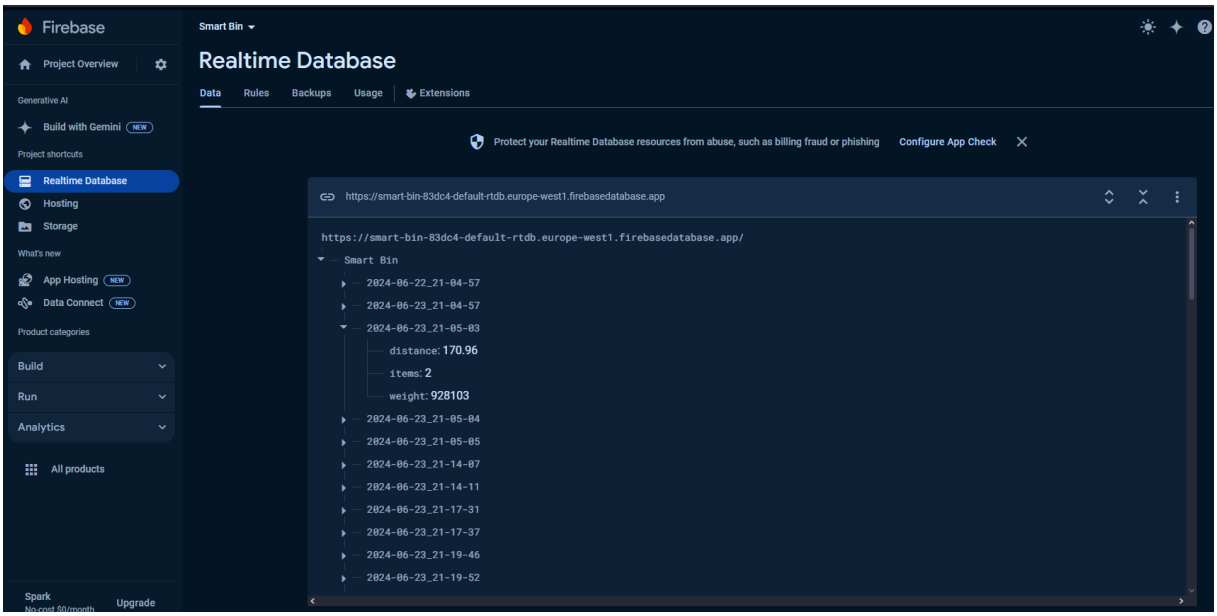
Slika 27: E-mail notifikacija
Izvor: https://www.google.com/gmail/about/

7.3. Logiranje u bazu podataka

Podatke koje senzor šalje prema frontend web aplikaciji spremati ćemo u bazu u svrhu prikaza povijesnih podataka i kao mogućnost analize podataka. Primjerice, iz tih podataka možemo pretpostaviti kakav otpad se u kanti nalazi. Recimo u ljetnim mjesecima možemo očekivati u kontinentalnim dijelovima manje otpada radi godišnjih odmora, dok u proljetnim mjesecima možemo očekivati više glomaznijeg otpada pod pretpostavkom da se ljudi rješavaju stvari koje su se akumulirale u zimskim mjesecima. Isto tako, proljetno vrijeme je uglavom vrijeme kada ljudi rade preinake i preuređivanje objekata, što gotovo uvijek rezultira otpadom. Uz to, upravo su ljetni proljetni mjeseci vrijeme uređivanja vrtova, voćnjaka, košnje itd.

Za implementaciju ove funkcionalnosti koristiti ćemo Firebase. To je platforma koje djeluje kao skup *backend* usluga u oblaku za razvoj aplikacija, web i mobilnih uglavnom. Razvijena je od strane Googla a pruža širok raspon raznih alata i servisa prigodnim za razvoj aplikacija. Oni su dizajnirani da budu skalabilni, učinkoviti i da omoguće razvojnim inženjerima da upravljanje *backend-om* bez da održavaju i upravljaju poslužiteljima. Neke od ključnih značajki Firebase:

- **Real-Time Database** – odnosno baza podataka u stvarnom vremenu predstavlja NoSQL bazu dizajniranu da bude skalabilna i fleksibilna. Glavna ideja ove baze je sinkronizacija podataka u stvarnome vremenu, odnosno sinkronizira podatke na svim klijentima u stvarnome vremenu a čak i pruža neke offline mogućnosti
- **Autentikacija** – Firebase autentikacija omogućuje cjelovito rješenje autentikacije i identiteta. Podržava autentifikaciju mail-om, lozinkom, telefonskim brojem i pružateljima identiteta kao što su Facebook, Twitter, Google i sl.
- **Analiza** – Firebase nudi besplatno rješenje za mjerenje i analizu koja pruža dublje uvide u korištenje same aplikacije
- **Hosting** – Podržava hosting statičnih datoteka kao što su primjerice HTML/CSS/JS datoteke koje isporučuje preko *Content Delivery Network-a* (CDN)
- **Funkcije u oblaku (cloud)** – omogućuju developerima pokretanje *backend* koda u odgovorima na događaje koji su pokrenuti ili preko HTTPS *requesta* ili nekih značajka Firebase-a



Slika 28: Logiranje podataka u bazu
Izvor: https://firebase.google.com

Na slici iznad nalazi se Firebase baza podataka u realnom vremenu. Svaki puta kad se detektira pokret, senzor zove funkcije koje kalkuliraju parametre i prikazuje ih na web aplikaciji. U isto vrijeme se ti isti podaci spremaju u bazu podataka. Spremaju se po *timestamp-u*, a unutra se nalazi struktura parametara koje su senzori u tom trenutku prikupili. Na taj način možemo na *frontend* strani prikazivati povijesne podatke, odnosno imamo mogućnost pokazati zadnja mjerenja od svakog dana. Na taj način možemo pratiti koliko se kanta punila, i koji je postotak popunjenosti svakoga dana. Upravo je Firebase odličan za ovakve stvari, radi svoje jednostavnosti i značajki koje nudi njegova *Realtime Database* funkcionalnost.

```
def fetch_latest_data_each_day():
    ref = db.reference('Smart Bin')
    all_data = ref.get()
    latest_entries = {}

    if all_data:
        for key, value in all_data.items():
            date_part = key.split('_')[0]
            if date_part not in latest_entries or key >
latest_entries[date_part]['timestamp']:
                timestamp = datetime.strptime(key, '%Y-%m-%d_%H-%M-%S')
                formatted_timestamp = timestamp.strftime('%Y-%m-%d at
%H:%M:%S')
                latest_entries[date_part] = {
                    'timestamp': formatted_timestamp,
                    'distance': value.get('distance', 0),
                    'items': value.get('items', 0),
                    'weight': value.get('weight', 0)
                }
    return latest_entries
```

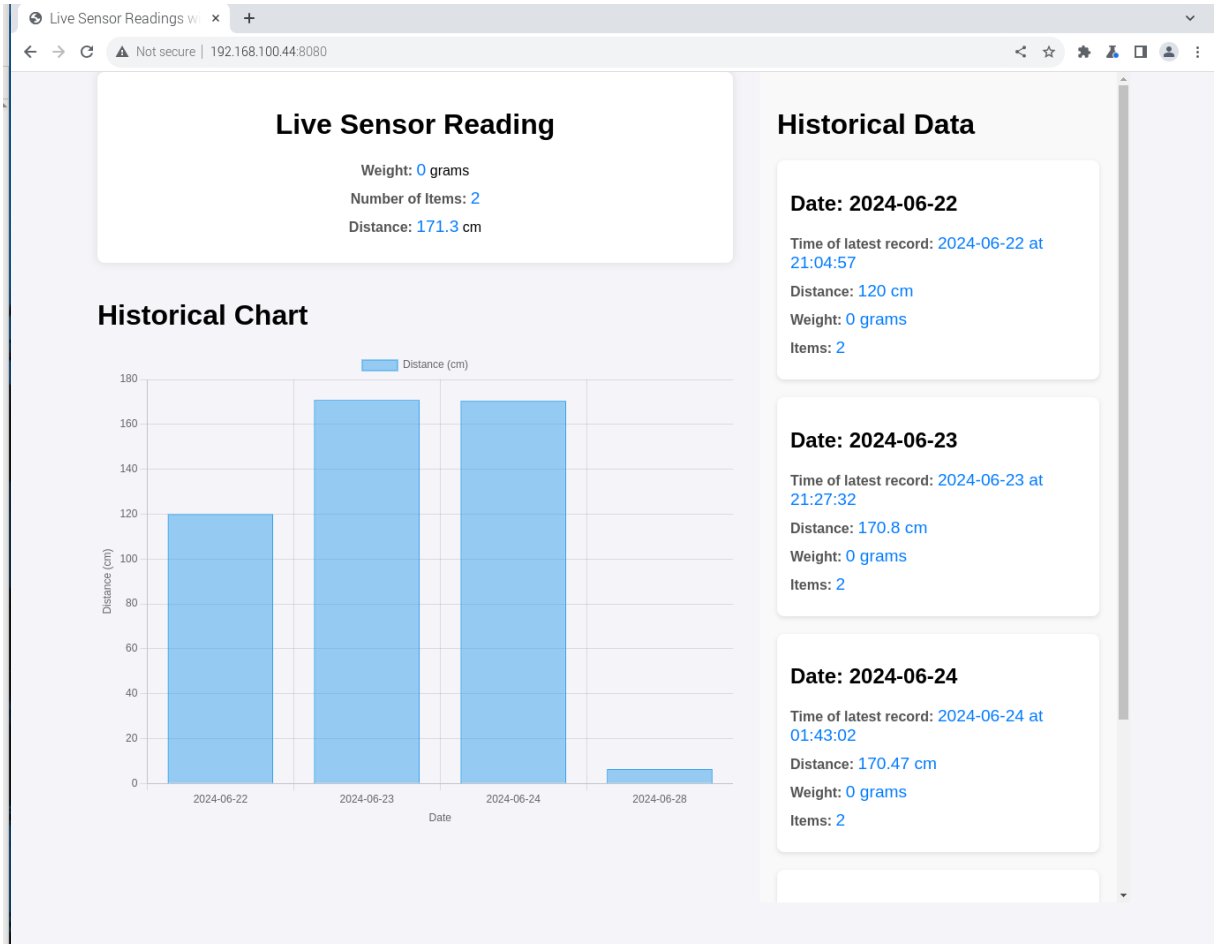
Slika 29: Dohvat povijesnih podataka
Izvor: vlastita izrada

Na slici iznad prikazana je funkcija za dohvat povijesnih podataka iz baze za svaki dan. Na početku stvaramo referencu do putanje Smart Bin baze, dohvaćamo sve podatke zapisane u 'Smart Bin' bazi i pospremamo podatke u *all_data* varijablu, i inicijaliziramo kontejner, odnosno prazan rječnik gdje ćemo spremiti podatke mjerenja za svaki dan.

Nakon toga ukoliko podaci dohvaćeni iz baze nisu prazni, iteriramo skroz sve vrijednosti i ključeve. Isto tako, obzirom na format zapisa timestamp-a u bazi, potrebno ga je parsirati u željeni format – YYYY-MM-DD iz formata YYYY-MM-DD_HH-MM-SS, koristeći '_' kao delimiter.

Latest_entries punimo tako što za svaki datumski dio vremenske oznake provjeravamo postoji li već u toj varijabli. Ako ne, to znači da nema zapisa za taj određeni datum. Isto tako, provjeravamo je li trenutni zapis, odnosno *timestamp* ili vremenska oznaka, veći od one koja je spremljena u *latest_entries*. Ako je trenutni ključ veći od spremljenog, znači da je to noviji zapis od onoga što trenutno imamo pohranjeno. Time smo osigurali da imamo pohranjene najnovije podatke svakoga dana.

7.4. Web aplikacija



Slika 30: Web aplikacija
Izvor: vlastita izrada

Web aplikacija služi nam za prikaz podataka prema korisniku. UI je podijeljen na tri dijela. Prvi je sekcija koja prikazuje mjerenja u stvarnome vremenu. Svaki puta kada senzor pokreta detektira pokret i izračunam potrebne parametre, isti se prikažu na ekranu. Ovaj prikaz služi za monitoriranje realnog stanje pametne kante. Odmah kraj sekcije s trenutnim mjerenjima, nalazi se sekcija s povijesnim prikazom podataka. Ta sekcija služi za prikaz zadnjih mjerenja ovisno o svakome danu. Funkcija prikuplja podatke o mjerenjima iz baze podataka, sortira podatke prema *timestampu*, i uzima zadnje vrijednosti mjerenja svakoga dana. Ti podaci mogu služiti za izrade izvještaja i praćenje punjenja/pražnjenja kante ovisno o svakome danu u tjednu ili mjesecu.

Ispod navedenih sekcija, nalazi se i prikaz kapaciteta punjenosti kante kroz grafikon. Grafikon je zamišljen da uzima zadnje vrijednosti mjerenja udaljenosti ultrazvučnog senzora svakoga dana. Upravo su mjerenja ultrazvučnog senzora najvrjedniji podatak jer predstavljaju sami

kapacitet kante za smeće, jer nam zapravo govore koliko ima slobodnoga prostora između poklopca i sadržaja kante.

Opisani prikazi podataka zapravo predstavljaju koncept ključnih informacija koje bi u realnom projektu bilo potrebno prikazivati korisnicima. Naravno, grafički prikazi bi trebali pokazivati informacije za svaki od parametara koji se mjere, i zahtijevali bi svoju sekciju. Povijesni podaci zahtijevali bi mogućnost filtriranja po datumu (dan/mjesec/godina) i mogućnost prikaza svih parametara ili samo odabirnih.

Flask koristi Jinja-u kao *engine* za izradu predložaka, odnosno *templating engine*. Jinja omogućava korištenje Python izraza i logike direktno u HTML predložak. Ovakva integracija omogućuje Flask aplikacijama generiranje dinamičkog HTML sadržaja na temelju podataka koji dolaze iz pozadine Pythona. Time se pojednostavljuje sam proces izgradnje web stranica, poboljšava se održavanje koda i olakšava se prikaz i vizualizacija podataka u stvarnome vremena. Ovime se dodatno omogućuje stvaranje sofisticiranih i responzivnih web aplikacija.

```html
<!DOCTYPE html>
<html>
<head>
    <title>Live Sensor Readings with Historical Chart</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='styles.css') }}">
    <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.0.1/socket.io.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script type="text/javascript">
        document.addEventListener("DOMContentLoaded", function() {
            var socket = io();
            socket.on('update_sensor_data', function(data) {
                document.getElementById('weight').innerHTML = data.weight;
                document.getElementById('items').innerHTML = data.items;
                document.getElementById('distance').innerHTML = data.distance;
            });

            var historicalData = [];
            {% for date, info in latest_entries.items() %}
                historicalData.push({
                    date: "{{ date }}",
                    distance: {{ info.distance }}
                });
            {% endfor %}

            historicalData.sort((a, b) => new Date(a.date) - new Date(b.date));

            var labels = historicalData.map(entry => entry.date);
            var distances = historicalData.map(entry => entry.distance);

            var ctx = document.getElementById('myChart').getContext('2d');
            var myChart = new Chart(ctx, {···
        });
    </script>
</head>
<body>
    <div class="main-container">
        <div class="left-column">
            <div class="container" id="live-data">
                <h1>Live Sensor Reading</h1>
                <div class="data">
                    <strong>Weight:</strong> <span id="weight">{{ weight }}</span> grams
                </div>
                <div class="data">
                    <strong>Number of Items:</strong> <span id="items">{{ items }}</span>
                </div>
                <div class="data">
                    <strong>Distance:</strong> <span id="distance">{{ distance }}</span> cm
                </div>
            </div>
            <h1>Historical Chart</h1>
            <canvas id="myChart" width="400" height="300"></canvas>
        </div>
        <div class="right-column">
            <div class="historical-data">
                <h1>Historical Data</h1>
                {% for date, info in latest_entries.items() %}
                    <div class="historical-entry">
                        <h2>Date: {{ date }}</h2>
                        <div class="data">
                            <strong>Time of latest record:</strong> <span>{{ info["timestamp"] }}</span>
                        </div>
                        <div class="data">
                            <strong>Distance:</strong> <span>{{ info["distance"] }} cm</span>
                        </div>
                        <div class="data">
                            <strong>Weight:</strong> <span>{{ info["weight"] }} grams</span>
                        </div>
                        <div class="data">
                            <strong>Items:</strong> <span>{{ info["items"] }}</span>
                        </div>
                    </div>
                {% endfor %}
            </div>
        </div>
    </div>
</body>
</html>
```

Slika 31: Programski kod frontend djela

Izvor: vlastita izrada

7.5. Finalno rješenje

```
import RPi.GPIO as GPIO
import time
from datetime import datetime
import smtplib
import sys
sys.path.append('/home/tomi/.local/lib/python3.11/site-packages')
from hx711 import HX711
from twilio.rest import Client
import keys
from flask import Flask, render_template
from flask_socketio import SocketIO, emit
import threading
import firebase_admin
from firebase_admin import credentials, db

app = Flask(__name__)
socketio = SocketIO(app, async_mode='threading')

cred = credentials.Certificate("/home/tomi/web app/smart-bin-83dc4-firebase-adminsdk-ooiz2-d0235abe15.json")
firebase_admin.initialize_app(cred, {···
})

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
TRIG = 23
ECHO = 24
motion_pin = 21

sensor_data = {···
}

hx = HX711(dout_pin=6, pd_sck_pin=5)
client = Client(keys.account_sid, keys.auth_token)

def setup():···

def fetch_latest_data_each_day():···

def measure_distance():···

def measure_weight():···

def send_email_notification():···

def send_sms_notification(weight, distance):···

def motion_detection(motion_pin):···

def getSensorData():···

def start_sensor_data_collection():···

@app.route('/')
def index():
    latest_entries = fetch_latest_data_each_day()
    print("Latest entries data structure:", latest_entries)
    distance = sensor_data.get("distance", "No Data")
    weight = sensor_data.get("weight", "No Data")
    items = sensor_data.get("items", 0)
    return render_template('index.html', distance=distance, weight=weight, items=items, latest_entries=latest_entries)

start_sensor_data_collection()
socketio.run(app, host="0.0.0.0", port=8080)
```

Slika 32: Programski kod finalnog rješenja
Izvor: vlastita izrada

Iznad se nalazi slika završnog programskog rješenja ovoga projekta. Naravno, na početku radimo import svih potrebnih knjižnica. Idući korak je inicijalizacija Flask aplikacije i postavljanje SocketIO instance konfigurirane da koristi *threading* kao asinkroni mode. Taj parametar specificira i određuje kako SocketIO rukuje operacijama koje se dešavaju istovremeno, osiguravajući da poslužitelj istovremeno rukuje s višestrukim vezama klijenta i događajima. SocketIO omogućuje dvosmjernu komunikaciju u stvarnom vremenu između klijenta i poslužitelja. Za razliku od HTTP request-a koje pokreće klijent, SocketIO omogućuje serveru da da šalje podatke prema klijentu kada se oni ažuriranju. Ključan je dio ove aplikacije jer glavna funkcionalnost je zapravo prikazivanje trenutnih podataka na web aplikaciji, koji moraju biti prikazani čim budu dostupni. SocketIO koristi *event-driven* arhitekturu, gdje i klijent i server mogu emitirati i odgovarati na događaje. Ta fleksibilnost omogućuje kontrolu nad prijenosom podataka, omogućuje složene interakcije između različitih komponenti, optimizira mrežne resurse smanjenjem latencije i poboljšavao samo korisničko iskustvo ažuriranjem podataka u stvarnome vremenu. U ovoj konfiguraciji SocketIO će kreirati novu dretvu za svaki zahtjev. Dretve se izvode istodobno s glavnom dretvom Flask aplikacije.

Nakon toga postavljamo konfiguraciju baze podataka, *cred* varijabla predstavlja *credentialse* računa postavljenje u JSON konfiguracijskoj datoteci. Nju koristimo da se inicijalizira Firebase, skupa s putanjom do same baze podataka, što je drugi argument *initialize_app* funkcije.

U idućim koracima postavljamo varijable specifične za senzore na početne vrijednosti, postavljamo BCM kao mode za numeraciju GPIO pinova, kreiramo *sensor_data* rječnik za spremanje podataka mjerenja, kreiramo instancu HX711 klase koja nam je potrebna za senzor težine, i instanciramo Twillio klijent za slanje SMS poruka. Isto tako kreirali smo i setup funkciju u kojoj incijaliziramo GPIO pinove koje su potrebne za interakciju Raspberry-ja i senzora.

Nakon ovih početnih koraka, kada je sve što je potrebno inicijalizirano i definiramo, krećemo graditi funkcije za razna računanja i manipulaciju podataka:

- ***def fetch_latest_data_each_day()*** – dohvat podataka iz baze i parsiranje tako da se dobije zadnja vrijednost mjerenja za svaki pojedini dan
- ***def measure_distance()*** – funkcija za mjerenje udaljenosti što predstavlja kapacitet kante za smeće, odnosno vrijednost koja govori koliko ima praznog mjesta između poklopca kante i njenog sadržaja
- ***def measure_weight()*** - funkcija za mjerenje težine
- ***def send_email_notification()*** – funkcija za slanje email notifikacija
- ***def send_sms_notification(weight, distance)*** – funkcija za slanje SMS notifikacija
- ***def motion_detection(motion_pin)*** – funkcija za detekciju pokreta, međutim služi kao okidač za druge stvari

Funkcija za detekciju pokreta u sebi sadrži brojač koji zapravo predstavlja broj bačenih predmeta. Isto tako služi kao okidač za pozivanje funkcija za mjerenje udaljenosti i težine, te dohvaćanje njihovih vrijednosti koje se spremaju u prethodno definirano *sensor_data* strukturu. Isto tako pomoću *socketio.emit* šaljemo ažurirane vrijednosti mjerenja prema web aplikaciji što nam omogućuje promjene u stvarnome vremenu svaki puta kada su novi podaci dostupni. Uz to, podaci o mjerenjima se spremaju u bazu podataka, pod svoj *timestamp*. Ukoliko je zadovoljen postavljen uvjet o količini praznoga prostora između poklopca kante i njenog sadržaja, pozivaju se notifikacijske funkcije koje šalju email i SMS poruku.

U ***get_sensor_data*** funckiji postavljamo *event_detection* na motin_pin (pin koji je povezan s senzorom za detekciju pokreta). Kada dođe do promjena stanja, okida se i poziva funkcija za detekciju pokreta. While petljom je osigurano kontinuirano prikupljanje podataka. Odgoda od 0,1 sekundi je postavljana kako bi se izbjegla pretjerana upotreba CPU-a dok se čekaju događaji povezani s kretanjem. Try-except blok osigurava da se program može prekinuti s CTRL+C kombinacijom tipki, što ujedno i čisti GPIO resurse.

Start_sensor_data_collection poziva *setup()* funkciju koja, kao što je već spomenuto incijalizira GPIO pinove. *Threading.Thread* stvara novi dretvu koja pokreću get_sensor_data funkciju i omogućuje da se proces prikupljanja podataka izvodi istovremeno s Flask aplikacijom osiguravajući da se monitoriranjem senzora ne blokira glavnu dretvu aplikacije. Nakon toga postavljamo vrijednost *sensor_thread* kao tzv.

daemon_thread. Time omogućujemo da se dretve automatski terminiraju kada se glavni proces zaustavi (Flask aplikacija). *Sensor_thread.start()* započinje kontinuirani proces prikupljanja podataka u pozadini.

Na kraju definiramo rutu web aplikacije. Kada klijent pristupi definiranom URL-u, Flask poziva *index()* funkciju za obradu zahtjeva. Logika funkcije je zapravo dohvaćanje najnovijih podataka mjerenja po danima, i dohvaćanje vrijednosti iz *sensor_data* strukture. Na kraju renderiramo HTML predložak s nazivom indeks.html i u njega prosljeđujemo dohvaćene vrijednosti kao varijable.

socketio.run() pokreće Flask aplikaciju s SocketIO podrškom, što omogućuje dvosmjernu komunikaciju u stvarnome vremenu između klijenta i poslužitelja. Ova postavka omogućuje klijentima primanje trenutnih ažuriranja o promjenama podataka senzora i interakciju s web aplikacijom u stvarnom vremenu

8. Zaključak

Ovaj rad demonstrira integraciju IoT tehnologija za praćenje i upravljanje podacima senzora u stvarnom vremenu, kao i povijeni pregled prikupljenih podataka. Ovaj koncept prikazuje kreiranje sustava koji može biti primijenjen u različitim scenarijima pametnih okruženja kao što su pametni spremnici za smeće, industrijsko praćenje ili kućno automatiziranje. Implementacija takvog sustava može biti jedan od koraka boljeg menadžmenta otpadom. Upravo je bitno spomenuti kako je rad s Raspberry PI-jem jako jednostavan, a cijena samog kontrolera i komponenti je dovoljno niska da bi jedan ovakav projekt bio i više nego prihvatljiv.

Izradom i implementacijom ovog koncepta stekao sam dublje razumijevanje važnosti primjene dretvi u programiranju. Ovakvim pristupom i implementacijom, omogućuje se paralelizam u smislu izvršavanje ključnih funkcija, poput neprekidnog prikupljanja podataka senzora i istovremenog osluškivanja događaja s GPIO pinova. Ovaj pristup ne samo da je optimizirao performanse aplikacije već je i omogućio nesmetano funkcioniranje više zadataka istovremeno, što je ključno u stvaranju efikasnog IoT sustava.

Kroz ovaj rad, naučio sam kako efektivno upravljati resursima, osigurati stabilnost i kontinuiranost rada aplikacije te optimizirati procese prikupljanja i obrade podataka. Vrijedna znanja koja će mi svakako pomoći u daljnjem razvoju i napretku.

9. Popis literature

[1] L. Williams, https://www.guru99.com/operating-system-tutorial.html, *What is Operating System? Explain Types of OS, Features and Examples*, dostupno 28.07.2023.

[2] G. L. Team, https://www.mygreatlearning.com/blog/what-is-operating-system/, *What is Operating System (OS)? Definition, Types, and Functions*, dostupno 28.07.2023.

[3] s.a., https://operacijskisustavi.weebly.com/podjela-os-a.html, *Podjela OS-a obzirom na tip koda,* dostupno 28.07.2023.

[4] s.a., https://www.javatpoint.com/types-of-operating-systems, *Types of Operating Systems (OS)I,* dostupno 28.07.2023.

[5] akash1295, https://www.geeksforgeeks.org/types-of-operating-systems/, *Types of Operating Systems*, dostupno 28.07.2023.

[6] s.a., https://gs.statcounter.com/os-market-share, O*perating System Market Share Worldwide*, dostupno 28.07.2023.

[7] pp_pankaj, https://www.geeksforgeeks.org/kernel-in-operating-system/, *Kernel in Operating System*, dostupno 28.07.2023.

[8] S. J. Bigelow, J. Lulka, https://www.techtarget.com/searchdatacenter/definition/kernel, *What is a kernel?*, dostupno 28.07.2023.

[9] A. S. Gillis, https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT, *Internet of Things (IoT)*, dostupno 28.07.2023.

[10] A. S. Gillis, B. Posey, S. Shea, https://www.techtarget.com/iotagenda/definition/IoT-device, *IoT devices (internet of things devices),* dostupno 28.07.2023.

[11] N. Duggal, https://www.simplilearn.com/iot-devices-article, *What Are IoT Devices? Definition, Types, and 5 Most Popular for 2023,* dostupno 28.07.2023.

[12] vikasG, arvindpdmn, https://devopedia.org/iot-operating-systems, *IoT Operating System*, dostupno 28.07.2023.

[13] Satyabrata_Jena, https://www.geeksforgeeks.org/architecture-of-internet-of-things-iot/, *Architecture of Internet of Things (IoT)*, dostupno 28.07.2023.

[14] shekharsaxena316, https://www.geeksforgeeks.org/5-layer-architecture-of-internet-of-things/?ref=rp, *5 Layer Architecture of Internet of Things*, dostupno 28.07.2023.

[15] K. Kimachia, https://www.enterprisenetworkingplanet.com/data-center/iot-operating-system/, *What is an IoT Operating System?*, dostupno 28.07.2023.

[16] L. Williams, https://www.guru99.com/preemptive-vs-non-preemptive-scheduling.html, Preemptive and Non-Preemptive Scheduling, dostupno 28.07.2023.

[17] S. Challouf, L. A. Saidane, L. Kriaa, https://www.researchgate.net/profile/Sabri-Challouf/publication/323711015_Comparison_of_IoT_Constrained_Devices_Operating_Systems_A_Survey/links/5feb6a01a6fdccdcb8167b06/Comparison-of-IoT-Constrained-Devices-Operating-Systems-A-Survey.pdf, *Comparison of IoT constrained devices operating systems : A Survey,* dostupno 28.07.2023.

[18] A. Pelaez, https://ubidots.com/blog/iot-operating-systems/, *9 IoT Operating Systems To Use in 2021 [List & Comparison],* dostupno 28.07.2023.

[19] FreeRTOS, https://www.freertos.org, dostupno 28.07.2023.

[20] RIOT, https://www.riot-os.org, dostupno 28.07.2023.

[21] D. E. Culler, https://www.fierceelectronics.com/iot-wireless/tinyos-operating-system-design-for-wireless-sensor-networks, *TinyOS: Operating System Design for Wireless Sensor Networks*, dostupno 28.07.2023.

[22] Mbed OS, https://os.mbed.com/mbed-os/, dostupno 28.07.2023.

[23] MicroPython, https://micropython.org, dostupno 28.07.2023.

[24] Raspberry Pi, https://www.raspberrypi.com, dostupno 28.07.2023.

[25] Tinkernut, https://www.youtube.com/watch?v=EKPobkb1N6o&t=1s, *Raspberry Pi – All you need to know*, dostupno 28.07.2023.

[26] LXDE, https://www.lxde.org, dostupno 29.06.2024.

10. Popis slika