Prikupljanje i pohranjivanje podataka iz Tesla Wall Connectora

Osmanović, Dorotea

Sveučilište Josipa Jurja Strossmayera u Osijeku

Fakultet primijenjene matematike i informatike

Sveučilišni prijediplomski studij Matematika i računarstvo

Dorotea Osmanović

Prikupljanje i pohranjivanje podataka iz Tesla Wall Connectora

Završni rad

Osijek, 2023

Sveučilište Josipa Jurja Strossmayera u Osijeku
Fakultet primijenjene matematike i informatike

Sveučilišni prijediplomski studij Matematika i računarstvo

Dorotea Osmanović

Prikupljanje i pohranjivanje podataka iz Tesla Wall Connectora

Završni rad

Mentor:
izv. prof. dr. sc. Danijel Grahovac

Osijek, 2023

Sažetak

Ovaj rad se bavi razvojem sustava za prikupljanje i pohranjivanje podataka s Tesla Wall Connector Gen 3 punjača. To je punjač za električna vozila koji može biti spojen na lokalnu WiFi mrežu. Nedokumentirani REST API omogućava dohvaćanje podataka o statusu punjača poput potrošnje energije, trajanja punjena, temperature i drugo. U ovom će radu biti pobliže opisano kako API, odnosno REST API funkcionira te kako ga koristiti. Također, bit će opisane sve tehnologije korištene u izradi sustava, počevši od Python modula (gdje će svaki biti objašnjen na temelju projekta), HTTP protokola te InfluxDB platforme koja će nam omogućiti vizualizaciju prikupljenih podataka. U konačnici detaljno će biti opisana instalacija samog sustava.

Ključne riječi

Tesla Wall Connector, REST API, Python, InfluxDB, data logger, HTTP protokol, Raspberry Pi

Collecting and storing data from the Tesla Wall Connector

Summary

This thesis deals with the development of a system for collecting and storing data from the Tesla Wall Connector Gen 3 charger. It is an electric vehicle charger that can be connected to a local WiFi network. An undocumented REST API allows retrieving data about the charger's status, such as energy consumption, charging duration, temperature, and more. This paper will provide a closer look at how the API, specifically the REST API, functions and how to use it. Additionally, it will describe all the technologies used in the system's development, starting with Python modules (each of which will be explained based on the project), the HTTP protocol, and the InfluxDB platform, which will enable us to visualize the collected data. Finally, the installation of the system itself will be detailed.

Keywords

Tesla Wall Connector, REST API, Python, InfluxDB, data logger, HTTP protocol, Raspberry Pi

Sadržaj

1 Uvod **1**

2 Tesla Wall Connector data logger **2**
2.1 Tehnologije . . . 3

3 Tehnički aspekti **5**
3.1 API . . . 5
3.1.1 Arhitektura API-ja . . . 6
3.1.2 API protokoli . . . 6
3.2 REST API . . . 7
3.2.1 Zašto i kako koristiti REST API? . . . 9
3.3 HTTP protokol . . . 9
3.3.1 HTTP metode . . . 10
3.4 Python moduli . . . 11
3.4.1 *Json* . . . 11
3.4.2 *Time* . . . 12
3.4.3 *Datetime* . . . 12
3.4.4 *Certifi* . . . 13
3.4.5 *Requests* . . . 14
3.4.6 *Influxdb_client* . . . 15

4 Implementacija **16**
4.1 Organizacija koda . . . 16
4.1.1 GitHub repozitorij . . . 17
4.2 twc.py skripta . . . 17
4.2.1 Autorizacija u bazu . . . 17
4.2.2 Prikupljanje podataka . . . 18
4.2.3 Adaptacija podataka . . . 21
4.2.4 Pohranjivanje podataka . . . 23
4.3 Automatsko pokretanje i konfiguracija . . . 24
4.3.1 run-at-startup.service . . . 24
4.3.2 start_twc_logger.sh . . . 25

5 Instalacija data loggera **27**
5.1 InfluxDB . . . 27
5.2 Raspberry Pi . . . 27
5.3 InfluxDB Cloud autorizacija . . . 29

1 | Uvod

U današnjem dinamičnom svijetu, tehnološke inovacije konstantno transformiraju način na koji percipiramo i koristimo svakodnevne objekte i usluge. Jedna od ključnih transformacija koja ne samo da je zahvatila, već i oblikuje budućnost prometa, jest sve veći broj električnih vozila koja se pojavljuju na cestama diljem svijeta. Vođeni ekološkom sviješću i potrebom za smanjenjem negativnog utjecaja na okoliš, sve više vozača okreće se električnim vozilima kao održivoj alternativi konvencionalnim vozilima koja koriste fosilna goriva. S porastom broja električnih vozila, sve veća potreba za infrastrukturom koja omogućava brzo i praktično punjenje postaje izuzetno važna. U tom kontekstu, kućni punjači poput Tesla Wall Connector Gen 3 punjača, ističu se kao tehnološko rješenje koje podržava rastuću potrebu za brzim i praktičnim punjenjem. Ovakvi punjači vlasnicima električnih vozila pružaju mogućnost jednostavnog punjenja kod kuće, osiguravajući im neovisnost i fleksibilnost pri svakodnevnim putovanjima.

S obzirom na rastuću cijenu energenata, trošak punjenja električnih vozila postaje bitan čimbenik za vozače. Kako bi vozači imali bolji uvid u stanje punjača tijekom punjenja te kako bi im se otvorila mogućnost analiziranja prikupljenih podataka poput trajanja punjenja, potrošene energije, cijene sesije itd., izrađen je projekt Tesla Wall Connector data logger čiji će dio s prikupljanjem i pohranjivanjem podataka biti pobliže opisan u ovome radu.

Budući da je Tesla Wall Connector Gen 3 kućni punjač za električna vozila punjač koji može biti spojen na lokalnu WIFI mrežu, pomoću nedokumentiranog REST API-ja omogućava se dohvaćanje podataka o statusu punjača. Projekt je osmišljen tako da nakon što se data logger instalira na Raspberry Pi računalo pokreće se Python skripta koja zajedno s drugim servisima omogućava dohvaćanje podataka s punjača te njihovo parsiranje i slanje u InfluxDB Cloud. U suštini, cilj ovoga projekta bilo je razviti sustav koji se relativno jednostavno instalira na mikro računalo u lokalnoj mreži, kontinuirano prikuplja podatke i omogućuje pristup trenutnim i povijesnim podacima u InfluxDB-u. Kako bismo bolje razumjeli projekt, u drugom će poglavlju biti dan kratki pregled projekta, uključujući pregled vizualizacije i tehnologija korištenih u radu. Treće poglavlje će detaljnije opisati API, REST API, HTTP protokol te sve korištene Python module, dok će četvrto poglavlje pružiti dublji uvid u implementaciju projekta i organizaciju koda. Na kraju, poglavlje 5 će dati detaljne upute za instalaciju data loggera.

2 | Tesla Wall Connector data logger

Kako bi se objasnile karakteristike i funkcionalnosti prikupljanja i pohranjivanja podataka iz Tesla wall connectora, ono mora biti opisano kao dio cijelog projekta.

Projekt se sastojao od dvije faze. U prvoj fazi trebalo je izraditi alat na Raspberry Pi računalu koji će u lokalnoj mreži prikupljati podatke kroz REST API. Zatim, na osnovu strukture prikupljenih podataka izraditi grafičko sučelje za prikaz podataka (dashboard), praćenje statistike podataka i drugo. U konačnici cilj ovog projekta bilo je razviti data logger za Tesla Wall Connector Gen 3 kućni punjač za električna vozila, tj. sustav koji se jednostavno može instalirati na mikro računalo poput Raspberry Pi, u lokalnoj mreži, prikupljati podatke i omogućiti pristup trenutnim i povijesnim podacima u InfluxDB-u. Vizualizacija podataka ostvarena je u InfluxDB-u u obliku dashboarda kao što je prikazano na slici 2.1.

Slika 2.1: Dashboard

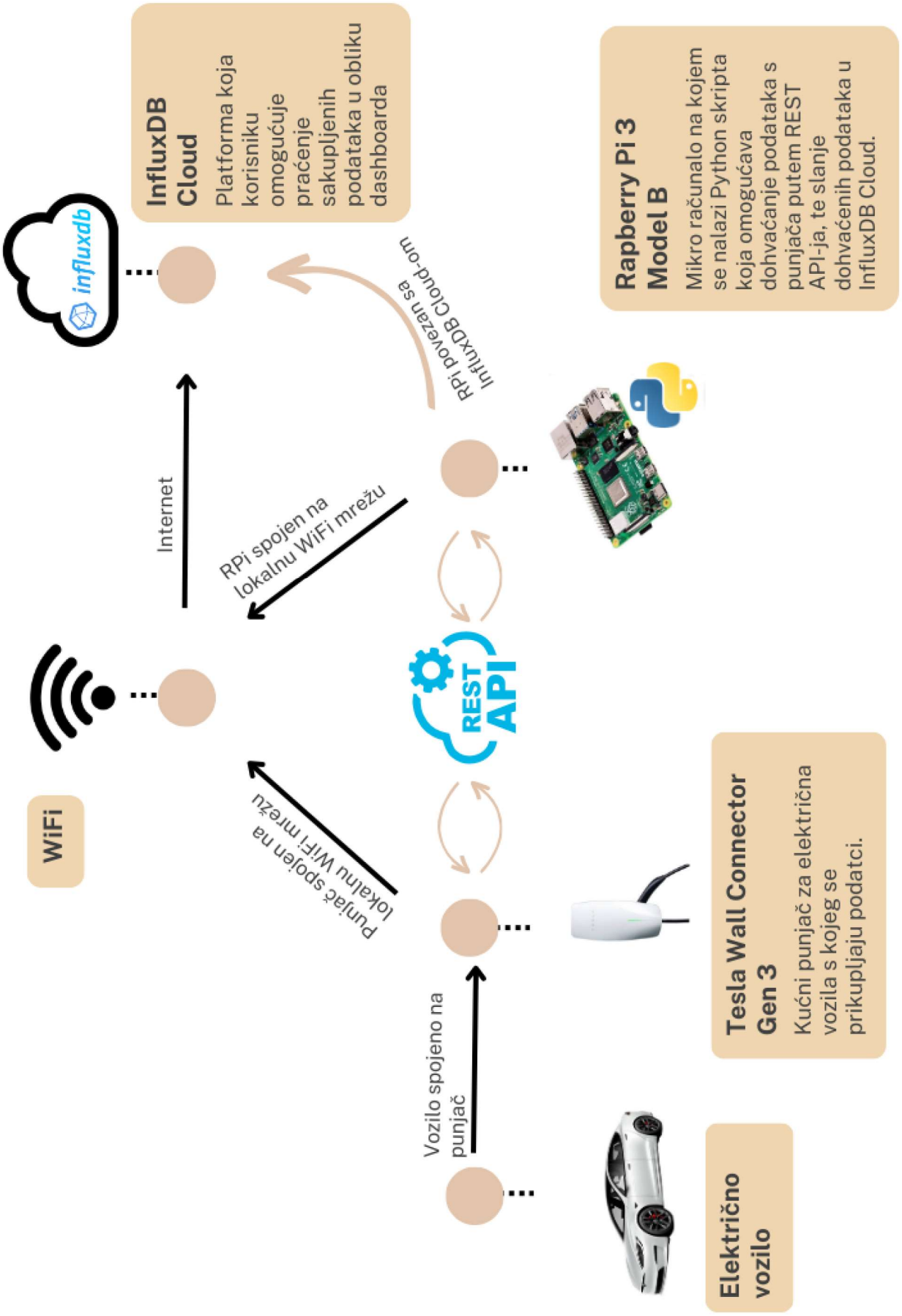
Dashboard se sastoji od 17 ćelija, a svaka ćelija je sagrađena na temelju nekog od prikupljenih podataka. Dakle, za podatke koji su ranije sakupljeni s punjača, u dashboardu se iscrtavaju grafovi kako bi bilo jednostavnije pratiti statistiku. Dashboard omogućuje biranje vremenskog intervala za koji želimo prikazati podatke te omogućuje odabir brzine osvježavanja.

2.1 Tehnologije

Projekt je napravljen koristeći platformu InfluxDB, programski jezik Python te sljedeće Python module koji će detaljnije biti opisani u poglavlju 3.4 :

- **json** – rad s JSON podacima
- **requests** – slanje HTTP zahtjeva
- **certifi** – SSL certifikat
- **influxdb_client** – spajanje Python skripte na InfluxDB
- **time** – rad s vremenom
- **datetime** – rad s datumima, pretvaranje sekundi u sate, minute i sekunde

Python skripta s popratim kodovima, pomoću koje se prikupljaju podaci s Tesla Wall Connector Gen 3 punjača, nalazi se na Raspberry Pi-u 3 Model B. Kako bi bilo moguće prikupljati podatke, Tesla Wall Connector mora biti spojen na istu lokalnu WiFi mrežu kao i Raspberry Pi budući da se dohvaćanje podataka vrši pomoću zahtjeva kroz REST API gdje se zatim podaci isporučuju putem HTTP-a u JSON formatu. Dakle, RPi i punjač razmjenjuje podatke putem REST API-ja što se jasnije vidi na slici 2.2.



Slika 2.2: Arhitektura projekta.

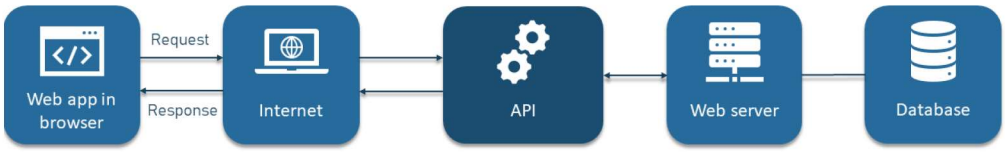
3 | Tehnički aspekti

U izradi projekta uključeno je više tehnologija. Budući da su se za dohvaćanje podataka najviše koristili API i REST API oni će za početak biti pobliže opisani, a nakon toga i svi korišteni Python moduli. Radi razumijevanja rada pobliže će biti opisan i HTTP protokol.

3.1 API

API (engl. *application programming interface*) je kratica za "sučelje za programiranje aplikacija". On predstavlja skup pravila i protokola koji omogućuju komunikaciju između različitih softverskih komponenti ili aplikacija. Omogućava različitim programima da razmjenjuju podatke, funkcionalnosti ili usluge bez potrebe da znaju detalje o unutarnjoj implementaciji jedan drugoga [24]. To čini API izuzetno važnim alatom za razvoj i integraciju softverskih sustava.

Aplikacija koja treba pristupiti informacijama ili funkcionalnostima iz druge aplikacije poziva njeno API sučelje, specificirajući zahtjeve o tome kako podaci/funkcionalnosti moraju biti pruženi. Druga aplikacija vraća podatke/funkcionalnosti koje je zatražila prva aplikacija. Sučelje preko kojeg ove dvije aplikacije komuniciraju je ono što API specificira [22].



Slika 3.1: Kako radi API? [24]

Princip rada API-ja obično se opisuje kroz *zahtjev-odgovor* (engl. *request-response*) komunikaciju između klijenta i poslužitelja što je i prikazano na slici 3.1. Poslužitelj je zadužen za backend logiku i za operacije s bazama podataka, dok je klijent svaka frontend aplikacija s kojom korisnik komunicira. U suštini, API radi kao srednji sloj između klijenta i poslužitelja(servera) tako što omogućava slanje zahtjeva za podacima i odgovora [24].

3.1.1 Arhitektura API-ja

API se sastoji od dviju komponenti:

- **tehnička specifikacija** - svojevrsna dokumentacija koja opisuje kako se podaci razmjenjuju između različitih softverskih rješenja ili aplikacija. To uključuje detalje o tome kako se postavljaju zahtjevi za obradu podataka (kao što su HTTP zahtjevi) i kako se podaci dostavljaju između rješenja, uključujući vrste podataka koje se mogu razmjenjivati i protokole koji se koriste za komunikaciju. Ova specifikacija pomaže razvojnim timovima razumjeti kako se koristi API i kako komunicirati s njim.

- **programsko sučelje** - predstavlja stvarni set programskih instrukcija i metoda koje su napisane prema tehničkoj specifikaciji. Ono omogućuje različitim aplikacijama da komuniciraju i razmjenjuju podatke ili funkcionalnosti.

Ukratko arhitektura API-ja sačinjena je od tehničke specifikacije koja definira kako API funkcionira i kako se podaci razmjenjuju, dok programska sučelja implementiraju tu specifikaciju, omogućavajući aplikacijama da stvarno koriste API za komunikaciju i razmjenu podataka [24].

3.1.2 API protokoli

Razvojni inženjeri često koriste određene nizove zahtjeva i tehnika kako bi integrirali sučelja za programiranje aplikacija u svoj softver. Da bi se upravljalo tim zahtjevima, postoje specifikacije nazvane **protokoli API-ja**. Ovi protokoli služe kao smjernice za inženjere. Oni definiraju standarde za korištenje API-ja kao što su vrste podataka, ispravne naredbe koje treba koristiti i mnoge druge aspekte povezane s razvojem aplikacija koje koriste API. Drugim riječima, protokoli API-ja su pravila koja olakšavaju komunikaciju i integraciju između različitih softverskih komponenata, čime se osigurava da se različite aplikacije mogu međusobno razumjeti i raditi zajedno na konzistentan način. Neki od najpoznatijih API protokola su:

- **SOAP** (eng. *Simple object access protocol*) - protokol za razmjenu strukturiranih informacija u decentraliziranoj okolini. Koristi XML (eng. *extensible markup language*) za pokretanje API komunikacije. XML je fleksibilan tekstualni oblik koji se široko koristi za razmjenu i pohranu podataka preko interneta i drugih mreža. On definira set pravila za enkodiranje dokumenata u formatu koji mogu razumjeti i čovjek i stroj.

  SOAP se prvi put pojavio u lipnju 1998. godine, stoga je najstariji protokol koji je još uvijek u upotrebi. Popularan je zbog fleksibilnosti svojeg prijenosnog kanala, ali strogo ovisi o XML-u zbog čega zahtijeva rigidna pravila [23].

- **gRPC** (eng. *Google remote procedure call*) - razvijen od strane Google-a i pušten za korištenje javnosti 2015. godine. Ovaj protokol specificira interakciju između aplikacija na temelju arhitekture klijent-poslužitelj. Prijenosni sloj gRPC-a prvenstveno se oslanja na HTTP. Glavna značajka gRPC-a je davanje

mogućnosti programeru da specificira svoje funkcije koje omogućuju fleksibilnu komunikaciju između usluga. On također nudi dodatne značajke poput vremenskog ograničenja, provjere autentičnosti i kontrole protoka [24].

3.2 REST API

REST API (eng. *Representational State Transfer Application Programming Interface*) predstavlja skup arhitektonskih ograničenja, a ne standard ili protokol. API programeri mogu implementirati REST na razne načine. REST je prvi put definirao računalni znanstvenik Dr. Roy Field 2000. godine u svojoj doktorskoj disertaciji [3]. REST pruža relativno visoku razinu fleksibilnosti i slobode za programere što je dovelo do toga da se danas REST API koristi kao uobičajena metoda za povezivanje komponenti i aplikacija u arhitekturi mikroservisa.

Web API-ji koji su u skladu s REST arhitektonskim ograničenjima nazivaju se RESTful API-ji. Postoji 6 ograničenja/pravila koja RESTful API mora zadovoljavati [25]:

1. **uniformno sučelje** (eng. *Uniform interface*) - bez obzira odakle zahtjev dolazi, svi API zahtjevi za isti izvor trebaju izgledati jednako. REST bi trebao osigurati da isti dio podatka, poput e-pošte korisnika, pripada samo jednom jedinstvenom izvoru.

2. **odvajanje klijenta od poslužitelja** (eng. *Client-server decoupling*) - klijentske i poslužiteljske aplikacije moraju biti potpuno neovisne jedna o drugoj. Jedina informacija koju klijentska aplikacija treba znati je URI (eng. *Uniform Resource Identifier*) traženog resursa. Ne može komunicirati s aplikacijom poslužitelja ni na koji drugi način. S druge strane, poslužiteljska aplikacija ne bi trebala modificirati klijentsku aplikaciju osim proslijediti joj tražene podatke putem HTTP-a.

3. **bez statusa** (eng. *Statelessness*) - REST API-ji su bez statusa, pa svaki zahtjev mora sadržavati sve informacije potrebne za njegovu obradu. Poslužiteljskim aplikacijama nije dopušteno pohranjivati podatke koji se odnose na zahtjev klijenta.

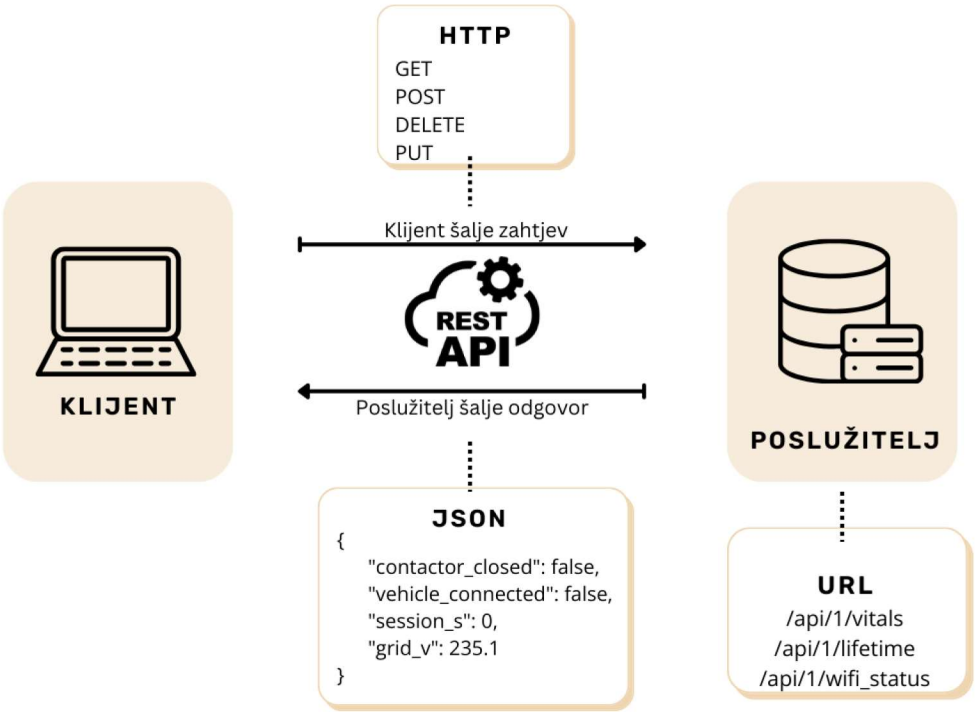
4. **mogućnost predmemoriranja** (eng. *Cacheability*) - kada je moguće, resursi bi trebali biti predmemorirani na strani klijenta ili poslužitelja, a odgovori poslužitelja moraju sadržavati informaciju o tome je li predmemoriranje dopušteno za isporučeni resurs.

5. **slojevita arhitektura sustava** (eng. *Layered system architecture*) - u REST API-ju, pozivi i odgovori prolaze kroz različite slojeve. Ne treba se pretpostaviti da se aplikacije klijenta i poslužitelja povezuju izravno jedna s drugom. U REST komunikacijskoj petlji može postojati više različitih posrednika, stoga REST API-ji moraju biti dizajnirani tako da ni klijent, ni poslužitelj ne mogu odrediti komuniciraju li s krajnjom aplikacijom ili s posrednikom.

6. **kod na zahtjev** (eng. *Code on demand*) - ovo pravilo je opcionalno. REST API-ji obično šalju statičke resurse, ali u određenim situacijama odgovori mogu sadržavati i izvršni kod. U tim slučajevima, kod bi se trebao izvoditi samo na zahtjev.

Kod REST API-ja stanje resursa u određenom trenutku naziva se "reprezentacija resursa" (eng. *resource representation*). Ona podrazumijeva način na koji se informacije o resursu prenose putem web API-ja. Kada klijent šalje zahtjev za određenim resursom putem API-ja, poslužitelj odgovara tako da šalje reprezentaciju tog resursa. Ona može biti u formatu koji je najprikladniji za klijenta ili za zahtjeve koje klijent postavlja. Formati koji se često koriste su HTML, XLT, Python, PHP ili običan tekst [26]. Ipak najviše se koristi JSON (eng. *JavaScript Object Notation*) jer je jednostavan za čitanje i interpretaciju kako za ljude tako i za računalne programe. Osim toga, JSON je neovisan o programskom jeziku te se stoga može koristiti s različitim programskim jezicima bez potrebe za složenim pretvorbama podataka. Više o JSON-u u poglavlju 3.4.1.

Kako bi izveli standardne funkcije s bazama podataka, poput stvaranja, čitanja, ažuriranja i brisanja zapisa, REST API-ji komuniciraju putem HTTP protokola koji će biti pobliže opisan u poglavlju 3.3. Sve HTTP metode se mogu koristiti u API pozivima. Štoviše, dobro dizajniran REST API sličan je web stranici koja radi u web pregledniku s ugrađenom HTTP funkcionalnosti [25]. Funkcionalnost samog REST API-ja objašnjena je na slici 3.2.



Slika 3.2: Funkcioniranje REST API-ja.

3.2.1 Zašto i kako koristiti REST API?

REST danas ima široku primjenu. Za svoje aplikacije koriste ga neke od najpopularnijih web tvrtki, uključujući Facebook, YouTube, Twitter i Google. Za razliku od npr. SOAP protokola, REST je odličan sistem za web aplikacije. Glavne prednosti REST-a su:

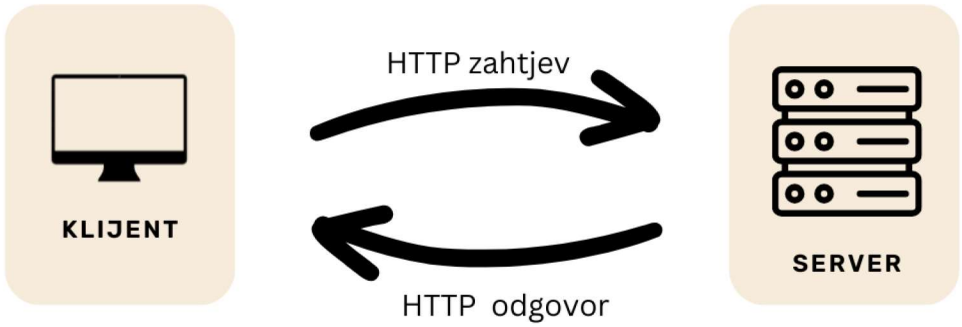
- **fleksibilnost** - mogu obraditi mnogo vrsti zahtjeva i poslati mnogo podataka u mnogo različitih formata.
- **skalabilnost** - dizajnirani su za komunikaciju između bilo koja dva računalna programa, bez obzira na njihovu veličinu ili sposobnosti. Dakle, kako web aplikacija raste i dodaje više resursa, njezin REST API se može brzo prilagoditi i obraditi sve veću količinu i broj zahtjeva.
- **uključivanje postojećih tehnologija** - da bi se zatražio resurs putem REST API-ja potrebno je samo dati URL.

Web aplikacije s javno dostupnim API-ima imat će dokumentaciju dostupnu u odjeljku "programeri" na svojim službenim web stranicama. Ondje je moguće pronaći upute o tome kako pristupiti i koristiti API. Ukoliko je korišten API sagrađen prema REST principima to će biti naznačeno. Mnogi API-ji zahtijevaju API ključ za korištenje. To je jedinstveni niz znakova koji programer koristi za autorizaciju pristupa u API. On se često šalje sa zahtjevima klijenta kao identifikacija klijenta poslužitelju. API ključeve treba držati privatnima, budući da ih je jednostavno zloupotrijebiti [18].

3.3 HTTP protokol

HTTP(eng. *Hypertext Transfer Protocol*) je protokol koji se koristi za razmjenu informacija putem interneta. Osigurava prijenos informacija s jednog mjesta na drugo. Temelj je World Wide Weba i koristi se za učitavanje web stranica pomoću hipertekstualnih veza. Fleksibilan je te može prenositi različite tipove podataka (tekst, slike, zvuk) između umreženih uređaja, a leži na aplikacijskom sloju (iznad TPC protokola, eng. *Transmition control protocol*). Temeljen je na modelu klijent-poslužitelj, tako da tipičan tok informacija putem HTTP-a uključuje klijentsko računalo koje šalje zahtjev poslužitelju, koji zatim šalje poruku odgovora kao što je prikazano na slici 3.3. On je proširiv protokol, što znači da će se njime moći prenositi i svi budući tipovi podataka.

HTTP zahtjev je način na koji internetske komunikacijske platforme kao što su web preglednici traže informacije potrebne za učitavanje web stranice. Svaki HTTP zahtjev upućen putem interneta sa sobom nosi niz kodiranih podataka koji nose različite vrste informacija. Tipični HTTP zahtjev se sastoji od: HTTP verzije, URL-a, HTTP metode, zaglavlja HTTP zahtjeva i opcionalno od HTTP tijela [27].



Slika 3.3: HTTP protokol

3.3.1 HTTP metode

HTTP metode, koje se ponekad nazivaju i HTTP glagoli, su radnje koje klijent (npr. web preglednik) izvršava nad resursom na web poslužitelju. Svaka HTTP metoda ima specifičnu svrhu i definira što klijent želi učiniti s resursom na poslužitelju. Neke od najčešćih metoda su:

- **GET** - koristi se za dohvat informacija s web poslužitelja. Klijent šalje zahtjev za određenim resursom i poslužitelj šalje taj resurs natrag klijentu.
- **POST** - koristi se za slanje podataka na web poslužitelj radi obrade. Na primjer, kada korisnik ispunjava obrazac na web stranici i pritisne "Pošalji," preglednik šalje POST zahtjev s podacima na poslužitelj, gdje se ti podaci obrađuju i pohranjuju. POST se često koristi za stvaranje novih resursa.
- **PUT** - koristi se za ažuriranje postojećeg resursa ili stvaranje novog resursa ako ne postoji. Klijent šalje cijeli resurs s ažuriranim podacima na poslužitelj.
- **DELETE** - koristi se za brisanje određenog resursa na poslužitelju. Klijent šalje zahtjev za brisanje, a poslužitelj uklanja resurs. Brisanje resursa putem DELETE metode obično nema povratnu informaciju i trajno briše resurs.
- **HEAD** - slična metodi GET, ali ne šalje stvaran sadržaj resursa, već se koristi za dobivanje informacije o resursu, kao što su veličina ili datum posljednjeg ažuriranja.

Ove su metode osnovni građevni blokovi HTTP-a i omogućuju raznovrsne akcije nad web resursima. Korištenje odgovarajuće metode ovisi o tome što se želi postići s resursom na web poslužitelju [8].

3.4 Python moduli

Python moduli su datoteke koje sadrže Python kod. Oni mogu biti sačinjeni od funkcija, klasa, varijabli i drugih Python definicija. Omogućuju organizaciju koda, ponovnu upotrebu funkcionalnosti i bolju upravljivost projekta. U nastavku će biti opisani moduli koji su korišteni u izgradnji ovog projekta.

3.4.1 *Json*

Json je Python modul koji omogućuje rad s JSON (eng. *JavaScript Object Notation*) formatom podataka. **JSON** je tekstualni format za razmjenu podataka između različitih računalnih sustava. Jednostavan je za čitanje i pisanje te je razumljiv ljudima i strojevima. Ima sintaksu vrlo sličnu Pythonu. Koristi se za slanje podataka između web preglednika i web servera, kao i za pohranu konfiguracijskih podataka i razmjenu podataka između različitih programskih jezika. JSON podaci često su organizirani kao kolekcije parova "ključ-vrijednost" (eng. key-value) što omogućuje strukturiranje podataka u obliku objekata s atributima. On je danas standardni format za komunikaciju između različitih računalnih sustava zbog svoje jednostavnosti i rasprostranjenosti. JSON podaci često se izražavaju kao ugniježđeni rječnici, liste i skalarne vrijednosti kao što su tekstovi, brojevi, Booleovi i Null. Nazvan je JSON jer blisko oponaša sintaksu koja se koristi u JavaScript objektima. JSON igra važnu ulogu u Python programiranju jer omogućuje učinkovitu serijalizaciju i deserijalizaciju podataka. Omogućuje Python programima da bez napora komuniciraju s web servisima, razmjenjuju podatke i pohranjuju strukturirane informacije.

Kako bi radili s JSON podacima, može se koristiti Pythonov modul *json*. To je ugrađeni Python modul koji pruža snažan skup podataka i klasa koje čine rad s JSON podacima jednostavnim. Programeri ga mogu koristiti za kodiranje Python objekta u JSON nizove i dekodiranje JSON nizova natrag u Python objekte [6].

U ovom radu modul *json* je korišten za deserijalizaciju (pretvaranje) JSON podataka u Python objekte. Korištena je funkcija *json.loads*() kao što se može vidjeti u primjeru koda 3.1. Nakon izvršenja ove funkcije, objava, objava_liftime i objava_wifi postaju Python objekti koji sadrže podatke iz JSON-a.

```
1 objava = json.loads(requests.get(api_url, timeout=5).content.decode
    ('UTF-8'))
2 objava_liftime = json.loads(requests.get(api_liftime_url, timeout
    =5).content.decode('UTF-8'))
3 objava_wifi = json.loads(requests.get(api_wifi_url, timeout=5).
    content.decode('UTF-8'))
```

Kod 3.1: Modul *json*

3.4.2 *Time*

Modul ***time*** u Pythonu je ugrađeni modul koji omogućuje rukovanje sa zadacima povezanim s vremenom. Takvi zadaci uključuju očitavanje trenutnog vremena, vrijeme formatiranja, omogućuju "spavanje" programa na određeni broj sekundi i drugo. *Time* modul dolazi s Pythonovim standardnim uslužnim modulom, tako da nema potrebe da ga se instalira izvana, već se jednostavno uvede pomoću naredbe *import time*.

Neke od funkcija koje modul *time* podržava su:

- **time.time() function** - vraća broj sekundi koje su prošle od epohe. Epoha je točka kada je vrijeme počelo. Za većinu sistema ona je 1.siječnja.1970 u 00:00:00.
- **time.ctime() function** - uzima sekunde prošle od epohe kao argument i vraća niz koji predstavlja lokalno vrijeme.
- **time.sleep() function** - obustavlja/odgađa izvršenje trenutnog koda na određeni broj sekundi. Kod 3.2 daje nam primjer korištenja funkcije time.sleep() u ovom projektu. Dakle, funkcija sleep(30) nam omogućuje da Python skripta svakih 30 sekundi dohvaća podatke o statusu punjača. Isto tako, u ovom primjeru funkcija sleep(5) osigurava da se svakih 5 sekundi ispisuje poruka "No connection to InfluxDB!" sve dok se veza sa InfluxDB-om ne uspostavi.

```
1     while (True):
2         try:
3             write_api.write(bucket=bucket, org= str(org_s),
    record=point)
4             print("Data sent")
5             break
6         except:
7             print("No connection to InfluxDB!")
8             time.sleep(5)
9
10    time.sleep(30) # separate points
```

Kod 3.2: *time.sleep()* funkcija

- **time.localtime() function** - uzima broj sekundi preteklih od epohe kao argument i vraća *struct_time* (strukturirani objekt koji se sastoji od 9 elemenata) u lokalnom vremenu.
- **time.perf_counter() function** - koristi se za mjerenje vremena izvođenja programa s visokom preciznošću (na razini mikro i nano sekundi) [14] [16].

3.4.3 *Datetime*

Datum i vrijeme u Pythonu nisu tipovi podataka, ali se može uvesti modul *datetime* kako bi se omogućio rad s njima. Kao i modul *time*, modul *datetime* je već

ugrađen u Python te ga ne treba instalirati izvana. Modul *datetime* nudi klase za rad s datumima i vremenom. *Date* i *datetime* su objekti u Pythonu, tako da kada se manipulira s njima, zapravo se manipulira s objektima, a ne stringom ili vremenskom oznakom.
Datetime modul kategoriziran je u 6 glavnih klasa:

- **date** - predstavlja datum (godine, mjesece i dane) bez vremena.
- **time** - predstavlja vrijeme (sate, minute, sekunde i mikrosekunde) bez datuma.
- **datetime** - predstavlja jednu točku u vremenu, uključujući i datum i vrijeme.
- **timedelta** - predstavlja trajanje, može biti korištena za izvođenje metrike sa datetime objektima.
- **tzinfo** - pruža informacije o vremenskoj zoni [13].

U ovom je radu bila korištena klasa **timedelta** koja predstavlja jednu od najlakših metoda za manipulaciju s datumima. Primjer koda 3.3 pokazuje kako je modul *datetime* s klasom timedelta korišten u projektu u svrhu pretvorbe dobivenih sekundi u format sati:minute:sekunde tj. 00:00:00.

```
1 #time in hours, minutes and seconds
2 session_s = objava['session_s']
3 import datetime
4 def sec_to_hours(sec):
5     return str(datetime.timedelta(seconds=sec))
6 session = sec_to_hours(session_s)
```

Kod 3.3: Modul *datetime*

3.4.4 *Certifi*

Modul *certifi* se koristi za upravljanje listom pouzdanih SSL (eng. *Secure Socket Layer*) certifikata u Python okruženju. SSL certifikati su ključni za sigurnu komunikaciju između Python aplikacija i web servera putem HTTPS (eng. *HyperText Transfer Protocol Secure*) protokola.
Ovaj modul sadrži kolekciju pouzdanih SSL certifikata koja se redovito ažurira kako bi sadržavala najnovije certifikate. Korištenje modula *certifi* olakšava programerima i developerima rad s SSL certifikatima, jer ne moraju ručno upravljati listom pouzdanih CA (eng. *Certificate Authority*) certifikata. Umjesto toga, mogu koristiti *certifi* kako bi automatski prepoznali i provjerili certifikate tijekom SSL komunikacije.

```
1 #InfluxDB authorization to the database
2 token = str(token_s)
3 org = str(org_s)
4 url = str(url_s)
```

```
client = influxdb_client.InfluxDBClient(url=url, token=token, org=
    org, ssl_ca_cert=certifi.where())
```

Kod 3.4: Modul *certifi*

U ovom radu (kod 3.4) *certifi* se koristi kako bi se osigurala sigurna komunikacija između Python aplikacije i InfluxDB baze podataka putem HTTPS protokola. Konkretno, *certifi* se koristi za postavljanje SSL certifikata tijekom uspostave veze s InfluxDB serverom.
ssl_ca_cert je dodatni parametar koji se koristi za postavljanje SSL CA certifikata. Ovdje se koristi funkcija *certifi.where*() kako bi se dobila putanja do datoteke koja sadrži listu pouzdanih CA certifikata iz modula certifi.
SSL CA certifikati koriste se za provjeru autentičnosti InfluxDB servera tijekom SSL komunikacije. Ako server ima SSL certifikat koji nije izdan od strane pouzdanog CA, veza će biti odbijena. Ukratko, korištenjem *certifi.where*() za postavljanje parametra ssl_ca_cert, osigurava se da InfluxDBClient provjerava SSL certifikate tijekom komunikacije s InfluxDB serverom kako bi se osigurala sigurna veza [11].

3.4.5 *Requests*

Python *requests* modul je jedna od najpopularnijih Python biblioteka. Ona omogućuje izradu HTTP zahtjeva prema web serverima. Pojednostavljuje rad s HTTP protokolom i omogućuje programerima da lako šalju podatke, izrađuju zahtjeve i obrađuju odgovore. Ovaj modul je potrebno instalirati prije upotrebe. Modul *requests* podržava različite HTTP metode, uključujući get, delete, head, patch, post, put i request [15].

Kod 3.5 daje nam primjer kako je modul *requests* korišten u ovom radu. On je dio koda koji osigurava dohvaćanje podataka svakih 5 sekundi s određenih web servisa. Ovdje se koristi *requests.get*() funkcija za izradu HTTP GET zahtjeva prema tri različita URL-a (api_url, api_liftime_url, api_wifi_url). Svaki od ovih zahtjeva preuzima podatke s odgovarajućeg web servisa.

```
while (True):
    write_api = client.write_api(write_options=SYNCHRONOUS)

    while (True):
        try:
            objava = json.loads(requests.get(api_url, timeout=5).
    content.decode('UTF-8'))
            objava_liftime = json.loads(requests.get(
    api_liftime_url, timeout=5).content.decode('UTF-8'))
            objava_wifi = json.loads(requests.get(api_wifi_url,
    timeout=5).content.decode('UTF-8'))
            print("Data fetched")
            break
        except:
            print("No connection to TWC!")
            time.sleep(5)
```

Kod 3.5: Modul *requests*

3.4.6 *Influxdb_client*

Python modul *influxdb_client* je biblioteka koja omogućuje komunikaciju s InfluxDB bazom podataka iz Python aplikacija. InfluxDB je popularna baza podataka optimizirana za pohranu i upit podataka vremenskih serija, često se koristi za praćenje senzorskih podataka, vremenskih podataka i druge vrste vremenskih serija. Osnovna komponenta biblioteke je ***InfluxDBClient*** klasa, koja omogućuje stvaranje veze s InfluxDB bazom podataka. Ova klasa omogućuje postavljanje parametara veze poput URL-a InfluxDB servera, autentikacijskih tokena i drugih opcija. Kod 3.6 je primjer korištenja *InfluxDBClient* komponente u ovom radu.

```
#InfluxDB authorization to the database
token = str(token_s)
org = str(org_s)
url = str(url_s)
client = influxdb_client.InfluxDBClient(url=url, token=token, org=
    org, ssl_ca_cert=certifi.where())
```

Kod 3.6: *InfluxDBClient* klasa

Point je klasa koja predstavlja pojedinačnu točku vremenskih serija podataka koje želimo zapisati u InfluxDB bazu. Može postaviti različite atribute kao što su mjerenje, polja i vremenska oznaka. U kodu 3.7 iz rada, klasa *Point* je služila kao objekt koji predstavlja vremenski trenutak ili događaj s određenim atributima i vrijednostima. Nakon što se stvorio *Point* objekt on se koristi za upisivanje podataka u InfluxDB bazu. Pohranjuje se kao pojedinačna točka vremenske serije u bazi podataka.

```
point = (
        Point("objava")
        #vitals
        .field("session", session)
        .field("contactor_closed_str", contactor_closed_str)
        .field("vehicle_connected", vozilo_spojeno)
```

Kod 3.7: *Point* klasa

Iz *influxdb_client* biblioteke učitavamo i **SYNCHRONOUS** koji se u radu koristi u drugom retku koda 3.5. To je postavka koja se koristi pri stvaranju Write API-ja za sinkrono pisanje podataka u bazu. To znači da će zahtjevi za pisanje biti izvršeni odmah i blokiraju izvođenje programa dok se ne dovrši upisivanje. ***write_api*** u navedenom kodu predstavlja Write API objekt iz *influxdb_client* biblioteke. Write API je sučelje koje omogućuje pisanje podataka u InfluxDB bazu podataka. U ovom slučaju, *write_api* stvara se kako bi se omogućilo sinkrono pisanje podataka u bazu [12].

4 | Implementacija

Projekt je osmišljen tako da nakon što se provede instalacija data loggera na Raspberry Pi skripta za prikupljanje i pohranjivanje podataka će se sama pokretati pri svakom uključivanju RPi-a. Potrebno je da pri prvom uključivanju korisnik unese IP adresu punjača kako bi skripta mogla dohvatiti podatke s punjača. Također, kako bi se podatci koje je skripta dohvatila mogli vidjeti, oni se parsiraju i šalju u InfluxDB Cloud. Svaki korisnik InfluxDB-a ima vlastiti token, org i url s kojim se može autorizirati i koje je potrebno unijeti pri pokretanju skripte.

4.1 Organizacija koda

Ovaj projekt podijeljen je u 8 datoteka:

- **twc.py** – glavna datoteka koja prikuplja i šalje podatke
- **start_twc_logger.sh** – datoteka koja omogućuje pokretanje novog terminala na Raspberry Pi u kojem se zatim pokreće twc.py
- **run-at-startup.service** - servis koji omogućuje da se start_twc_logger.sh pokreće pri svakom pokretanju RPi-ja
- **ip_address.txt** – sadrži IP adresu punjača
- **token.txt** – sadrži token za autorizaciju na InfluxDB Cloud
- **org.txt** - sadrži org (naziv za organizaciju) za autorizaciju na InfluxDB Cloud
- **url.txt** - sadrži url za autorizaciju na InfluxDB Cloud
- **tesla_wall_connector_gen_3.json** – dashboard u kojem se nalaze grafički prikazi dohvaćenih podataka u vremenu u JSON formatu, može se umetnuti u InfluxDB

Datoteke twc.py, start_twc_logger.sh, ip_address.txt, token.txt, org.txt i url.txt dio su mape naziva twc koja je stavljena na Raspberry Pi koji će pomoću ovih datoteka prikupljati podatke preko REST API-ja i kontinuirano ih slati u InfluxDB Cloud u obliku bucket-a naziva "baza". Svi podaci spremljeni u bucket naziva "baza" bit će prikazani u dashboardu u InfluxDB-u što će nam omogućiti json datoteka tesla_wall_connector_gen_3.json.

4.1.1 GitHub repozitorij

Izvorni kod sa sveukupnom implementacijom može se pronaći u GitHub repozitoriju:
https://github.com/DoraTea/Tesla-Wall-Connector-data-logger.git

4.2 twc.py skripta

twc.py je glavna skripta koja se nalazi na Raspberry Pi uređaju i omogućava dohvaćanje podatka s Tesla Wall Connectora te slanje dohvaćenih podataka u InfluxDB Cloud. Ona također parsira te podatke kako bi bili u željenom obliku. Kako bi ova skripta mogla vršiti sve navedeno, za njenu implementaciju potrebni su Python moduli koji se pomoću naredbe `import` učitavaju na početku twc.py skripte (Kod 4.1). Svi korišteni moduli su detaljno opisani u poglavlju 3.4.

```
import json
import requests
import certifi

import influxdb_client, time
from influxdb_client import InfluxDBClient, Point
from influxdb_client.client.write_api import SYNCHRONOUS
```

Kod 4.1: Učitavanje potrebnih Python modula.

4.2.1 Autorizacija u bazu

InfluxDB Cloud je brza platforma koja pruža skalabilnu i upravljivu bazu podataka za pohranu, upravljanje i analizu vremenskih serija podataka. Ova usluga omogućava organizacijama da efikasno prikupljaju, čuvaju i analiziraju podatke koji se mijenjaju tijekom vremena, kao što su senzorski podaci i metrike. Omogućava korisnicima jednostavno skaliranje resursa prema potrebama i pristup raznim alatima za vizualizaciju i analizu podataka. Osim toga, pruža sigurnosne mehanizme i integracije kako bi se osiguralo da se podaci mogu koristiti na pouzdan i siguran način [4].

Budući da su za autorizaciju u InfluxDB Cloud potrebni token, org i url, twc.py skripta je osmišljena tako da pri njenom prvom pokretanju korisnik treba upisati svoj token, url i org, koji će zatim trajno biti upisani u tekstualnim datotekama token.txt, org.txt i url.txt. Ukoliko bude potrebe za njihovom promjenom, to se može učiniti prema uputama u poglavlju 5. Na primjeru koda 4.2 prikazano je kako se vrši unos i spremanje tokena. Org i url se unose na analogan način.

```
#Token input
token_text = open("token.txt", "r")
token_s = token_text.read().replace('\n', '').replace(' ','')

```

```
5 if (token_s == ""):
6     token_i = input("Enter the Token:")
7
8     token_text= open("token.txt", "w")
9     t_write = token_text.write(token_i)
10    token_text.close()
11
12    token_text2 = open('token.txt', 'r')
13    token_s = token_text2.read().replace('\n', '').replace(' ','')
14    print("Token: " + token_s)
15    token_text2.close()
16 else:
17    print("Token: " + token_s)
18    token_text.close()
```

Kod 4.2: Unos tokena.

Dakle, u početku je napravljena tekst datoteka token.txt koja se zatim u twc.py skripti čita kako bi se provjerilo je li već unesen neki token, ako se utvrdi da nema upisanog tokena, ispisuje se poruka "`Enter the Token:`" te potom korisnik upisuje token koji se zatim sprema u token.txt datoteku kako bi se pri drugim pokretanjima skripte mogao samo pročitati, a ne ponovno upisivati. Na analogan način se provodi upis org-a i url-a.

Nakon što Python skripta ima pristup podacima potrebnim za autorizaciju na InfluxDB Cloud ona će biti izvršena pomoću *InfluxDBClient* klase iz modula *influxdb_client* pomoću kojeg se skripta spaja na InfluxDB. Sama funkcionalnost modula opisana je u poglavlju 3.4.6.

```
1 #InfluxDB authorization to the database
2 token = str(token_s)
3 org = str(org_s)
4 url = str(url_s)
5 client = influxdb_client.InfluxDBClient(url=url, token=token, org=
    org, ssl_ca_cert=certifi.where())
```

Kod 4.3: Autorizacija u bazu.

4.2.2 Prikupljanje podataka

Kako bi bilo omogućeno prikupljanje podataka s punjača, twc.py skripta se povezuje s punjačem preko njegove IP adrese koju korisnik mora unijeti pri pokretanju skripte.

```
1 #IP address input
2 text = open("ip_address.txt", "r")
3 ip = text.read().replace('\n', '').replace(' ','')
4
5 if (ip == ""):
6     ip_adr = input("Enter the IP address:")
7
```

```
8       text= open("ip_address.txt", "w")
9       write = text.write(ip_adr)
10      text.close()
11
12      text2 = open('ip_address.txt', 'r')
13      ip = text2.read().replace('\n', '').replace(' ','')
14      print("IP address: " + ip )
15      text2.close()
16  else:
17      print("IP address: " + ip )
18      text.close()
```

Kod 4.4: Unos IP adrese.

Kod 4.4 osigurava da postoji IP adresa u datoteci ip_address.txt i ispisuje ju. U početku koda, otvara se već postojeća datoteka ip_address.txt u načinu čitanja (eng. *read mode*) i čita se sadržaj datoteke. Nakon toga, uklanjanju se svi znakovi za novi red i praznine iz pročitanog sadržaja, kako bi se dobio samo čisti tekst IP adrese. Zatim se provjerava je li IP adresa unesena. Ako je, ona se ispisuje korisniku. Ako nije, traži se od korisnika da unese IP adresu i ona se zatim upisuje u datoteku ip_address.txt. Ponovno se otvara ta datoteka kako bi se pročitala nova IP adresa i ispisuje se korisniku.

Nakon toga IP adresa se prosljeđuje u linkove pomoću kojih se potom dohvaćaju podaci putem nedokumentiranog REST API-ja kao što se može vidjeti u kodu 4.5. S obzirom na to da je API nedokumentiran, poznati su samo neki API pozivi [19]. U danom kodu korišteni su API pozivi:

- **vitals** - za podatke o statusu punjača u sesiji.
- **lifetime** - za podatke o statusu punjača od početka njegovog korištenja.
- **wifi_status** - za podatke vezane uz wifi status punjača.

```
1 #vitals
2 api_url = 'http://' + str(ip) + '/api/1/vitals'
3 print(api_url)
4 #lifetime
5 api_liftime_url = 'http://' + str(ip) + '/api/1/lifetime'
6 print(api_liftime_url)
7 #wifi_status
8 api_wifi_url = 'http://' + str(ip) + '/api/1/wifi_status'
9 print(api_wifi_url)
```

Kod 4.5: REST API pozivi.

Budući da REST API podržava GET zahtjev, koji omogućava preuzimanje podataka s poslužitelja iz određenog izvora, skripta twc.py koristi modul *requests* za izvršavanje ovog zadatka (Kod 4.6). Točnije, koristi se metoda *requests.get*(*url*) za dohvaćanje sadržaja s navedenog URL-a [7].

```
while (True):
    write_api = client.write_api(write_options=SYNCHRONOUS)

    while (True):
        try:
            objava = json.loads(requests.get(api_url, timeout=5).content.decode('UTF-8'))
            objava_liftime = json.loads(requests.get(api_liftime_url, timeout=5).content.decode('UTF-8'))
            objava_wifi = json.loads(requests.get(api_wifi_url, timeout=5).content.decode('UTF-8'))
            print("Data fetched")
            break
        except:
            print("No connection to TWC!")
            time.sleep(5)
```

Kod 4.6: Dohvaćanje podataka

Dobiveni sadržaj se zatim dekodira kako bi bio čitljiv, što se vrši pomoću metode *content.decode('UTF-8')*. Kako bi podatci bili lakše obradivi, dobiveni JSON niz se pretvara u Python rječnik pomoću metode *json.loads()*. Primjer podataka u json formatu koje vraća punjač dan je na slici 4.1.


```
{
  "contactor_closed": false,
  "vehicle_connected": false,
  "session_s": 0,
  "grid_v": 230.1,
  "grid_hz": 49.928,
  "vehicle_current_a": 0.1,
  "currentA_a": 0,
  "currentB_a": 0,
  "currentC_a": 0,
  "currentN_a": 0,
  "voltageA_v": 0,
  "voltageB_v": 0,
  "voltageC_v": 0,
  "relay_coil_v": 11.8,
  "pcba_temp_c": 19.2,
  "handle_temp_c": 15.3,
  "mcu_temp_c": 25.1,
  "uptime_s": 831580,
  "input_thermopile_uv": -233,
  "prox_v": 0,
  "pilot_high_v": 11.9,
  "pilot_low_v": 11.9,
  "session_energy_wh": 22864.699,
  "config_status": 5,
  "evse_state": 1,
  "current_alerts": []
}
```


Slika 4.1: Primjer podataka koje vraća punjač.

Da bi se osigurao kontinuirani rad skripte za dohvaćanje podataka s punjača, čak i u slučaju gubitka veze s punjačem, koristimo Python-ove metode "`try`" i "`except`". One omogućuju da skripta nastavi pokušavati dohvatiti podatke sve dok se ponovno ne uspostavi veza s punjačem [17]. Ako skripta nije uspostavila vezu s punjačem, periodično će ispisivati poruku: "`No connection to TWC!`" svakih 5 sekundi dok se veza ne obnovi. Kada skripta uspješno dohvati podatke, ispisuje poruku: "`Data fetched`".

Nakon što su se ranije opisanim postupkom podatci dohvaćeni s punjača prebacili u formu Python rječnika (Slika 4.2), iz njega se dohvaćaju samo oni podatci koji su potrebni u svrhu izrade data loggera. Primjer dohvaćanja podataka dan je u kodu 4.7.

```
currentA_a = objava['currentA_a']
vehicle_current_a = objava['vehicle_current_a']
voltageA_v = objava['voltageA_v']
relay_coil_v = objava['relay_coil_v']
grid_v = objava['grid_v']
grid_hz = objava['grid_hz']
pcba_temp_c = objava['pcba_temp_c']
handle_temp_c = objava['handle_temp_c']
mcu_temp_c = objava['mcu_temp_c']
```

Kod 4.7: Primjer dohvaćanja podataka iz rječnika "objava".

{'contactor_closed': False, 'vehicle_connected': False, 'session_s': 0, 'grid_v': 230.1, 'grid_hz': 49.928, 'vehicle_current_a': 0.1, 'currentA_a': 0, 'currentB_a': 0, 'currentC_a': 0, 'currentN_a': 0, 'voltageA_v': 0, 'voltageB_v': 0, 'voltageC_v': 0, 'relay_coil_v': 11.8, 'pcba_temp_c': 19.2, 'handle_temp_c': 15.3, 'mcu_temp_c': 25.1, 'uptime_s': 831580, 'input_thermopile_uv': -233, 'prox_v': 0, 'pilot_high_v': 11.9, 'pilot_low_v': 11.9, 'session_energy_wh': 22864.699, 'config_status': 5, 'evse_state': 1, 'current_alerts': []}

Slika 4.2: Primjer dohvaćenih podataka prebačenih u formu Python rječnika.

4.2.3 Adaptacija podataka

Prije korištenja nekih podataka moraju se napraviti izmjene. Od dobivenih podataka napravljeni su novi podatci pomoću raznih jednostavnih funkcija. Podatci poput "`contactor_closed`", "`vehicle_connected`" i "`wifi_connected`" su iz tipa bool prebačeni u oblik stringa s obzirom na to da je za izgled dashboard-a bolje odgovarala pisana poruka, tj. string (Kod 4.8).

```
#contactor closed? returns string
contactor_closed = objava['contactor_closed']
def bool_str(bool_var):
    if(bool_var == True):
        return "True"
    else:
        return "False"
contactor_closed_str = bool_str(contactor_closed)

```

```
#vehicle connected?
vehicle_connected = objava['vehicle_connected']
def if_connected(bool_var):
    if(bool_var == True):
        return "Vehicle is connected."
    else:
        return "Vehicle is not connected."

vozilo_spojeno = if_connected(vehicle_connected)
```

Kod 4.8: Primjer prebacivanja izvornih podataka tipa bool u tip string

Podatke poput "session_s", "uptime_s" i "charging_time_s" punjač vraća u sekundama. Stoga je napravljena funkcija (Kod 4.9) koja te podatke koristeći modul *datetime* pretvara iz sekundi u format "sat:minuta:sekunda".

```
import datetime
def sec_to_hours(sec):
    return str(datetime.timedelta(seconds=sec))
session = sec_to_hours(session_s)

session_s = objava['session_s']
session = sec_to_hours(session_s)
```

Kod 4.9: Funkcija sec_to_hours i njeno korištenje na "session_s" podatku.

Također, modificiran je i podataka o količini energije u sesiji "session_energy_wh" te "energy_wh" koji je prebačen iz mjerne jedinice Wh u kWh.

```
#wh to kWh
session_energy_wh= objava['session_energy_wh']
def kWh(n):
  return float(n/1000)
session_energy_kwh = kWh(session_energy_wh)
```

Kod 4.10: Primjer jednostavne funkcije za prebacivanje iz mjerne jedinice Wh u kWh.

Iz podatka o potrošenoj energiji u sesiji napravljen je novi podatak za trošak potrošene električne energije u sesiji tj. "cijena" te je uračunata niža tarifa i pretvoren je trošak u eure, a analogno je napravljeno i za podatak o trošku potrošnje električne energije od početka korištenja punjača "total_electricity_cost". Podaci za snagu punjača u fazi A, B i C su također načinjeni od već dohvaćenih podataka.

```
def price(x):
    return float(((x/1000)*0.312)/7.52)
cijena = price(session_energy_wh)
```

Kod 4.11: Primjer jednostavne funkcije za računanje cijene energije.

4.2.4 Pohranjivanje podataka

Napravljen je bucket na InfluxDB Cloud-u pod nazivom "baza" gdje će se spremati podaci s punjača u vremenu. Bucket je konceptualni kontejner koji se koristi za organizaciju i pohranu podataka. Svaki bucket predstavlja određenu logičku jedinicu za pohranu podataka. Kada podaci stignu u InfluxDB Cloud, obično se pohranjuju u određeni bucket prema konfiguraciji [10].

Podaci o stanju punjača su organizirani i pohranjeni u measurement "objava," što predstavlja strukturu podataka unutar InfluxDB-a. Za učinkovito pisanje tih podataka u obliku "point-ova," korištena je klasa *Point()* iz *influxdb_client* modula. Nazivi podataka zajedno s njihovim odgovarajućim vrijednostima pohranjeni su kao "field-ovi" [1].

```
#liftime
.field("charging_time_h", charging_time_h)
.field("energy_kwh", energy_kwh)
.field("total_electricity_cost", total_electricity_cost)
.field("uptime_h", uptime_h)

#wifi_status
.field("wifi_signal_strength", wifi_signal_strength)
.field("wifi_connected_str", wifi_connected_str)
```

Kod 4.12: Primjer spremanja podataka u field željenog naziva.

twc.py skripta sinkronizirano šalje dohvaćene podatke o statusu punjača na InfluxDB Cloud svakih 30 sekundi pomoću metode *write*. Ova metoda za argumente prima bucket, org i point koji su prethodno definirani. Kako bi se osigurao kontinuitet rada skripte i reagiranje na prekide veze s InfluxDB-om, opet se koriste strukture "`try`" i "`except`" [17]. One omogućuju skripti da čeka ponovno uspostavljanje veze i tijekom tog vremena ispisuje poruku "`No connection to InfluxDB!`" Kada veza bude ponovno uspostavljena, skripta će ispisati poruku "`Data sent`".

```
while (True):
  try:
      write_api.write(bucket=bucket, org= str(org_s), record=point)
      print("Data sent")
break
except:
  print("No connection to InfluxDB!")
  time.sleep(5)

time.sleep(30) # separate point
```

Kod 4.13: Slanje podataka o statusu punjača na InfluxDB Cloud.

4.3 Automatsko pokretanje i konfiguracija

U ovom poglavlju detaljno će biti opisani postupci i alati koji omogućavaju automatsko pokretanje i konfiguraciju sustava na Raspberry Pi. To je ključni aspekt implementacije koji omogućuje pouzdano i efikasno upravljanje sustavom.

Kroz postavljanje Raspberry Pi uređaja i njegovo konfiguriranje za samostalno pokretanje, osigurava se da prikupljanje podataka nikada ne prestaje. Kada RPi bude uključen u struju, skripta za prikupljanje podataka automatski se pokreće. To znači da nema potrebe za ručnim pokretanjem ili prisutnošću korisnika. Sustav je samoodrživ. Osim toga, automatizacija ide korak dalje. Ako dođe do problema kao što je nestanak struje, RPi će se samostalno ponovno pokrenuti. To jamči kontinuirano prikupljanje podataka, čak i u naizgled nepredvidivim situacijama.

Budući da se svi podaci sa RPi uređaja šalju u InfluxDB Cloud, to osigurava da korisnik uopće ne mora biti u blizini RPi-ja već podatke može pratiti s bilo kojeg uređaja koji podržava InfluxDB registraciju.

4.3.1 run-at-startup.service

run-at-startup.service je systemd unit tj. konfiguracijska datoteka koja opisuje servis koji će se pokrenuti na Raspberry Pi uređaju. U ovom radu ona će nam omogućiti da se start_twc_logger.sh datoteka pokreće pri svakom pokretanju Raspberry Pi uređaja [5]. Više o start_twc_logger.sh datoteci u poglavlju 4.3.2.

```
1 [Unit]
2 Description=Run after all
3 After=graphical.target
4
5 [Service]
6 Environment=XAUTHORITY=/home/admin/.Xauthority
7 DISPLAY=:0
8 User=admin
9 Type=idle
10 RemainAfterExit=yes
11 ExecStart=/home/admin/twc/start_twc_logger.sh
12 User=admin
13 Environment=XAUTHORITY=/home/admin/.Xauthority
14
15 [Install]
16 WantedBy=graphical.target
```

Kod 4.14: Skripta run-at-startup.service

Objašnjenje koda 4.14 tj. run-at-startup.service datoteke po sekcijama [20]:

- [Unit]
  - `Description` - opis servisa, kratki tekst što servis radi, ne utječe na izvršavanje. U ovom slučaju to je "`Run after all`" tj. "pokreni nakon svega".

  - `After` - navodi koji sistemski cilj (eng. *target*) mora biti postignut prije nego što se servis pokrene. U ovom slučaju, servis će se pokrenuti nakon postizanja "`graphical.targeti`", što znači da će se pokrenuti nakon što se grafičko okruženje učita.

- [Service]
  - `Environment` - postavka okoline (eng. *environment variable*) koja se koristi u kontekstu ovog servisa. Ovdje se postavljaju `XAUTHORITY` i `DISPLAY` varijable koje su važne za grafičko okruženje.
  - `User` - definira pod kojim korisnikom će servis raditi. U ovom primjeru koristi se korisnik "`admin`".
  - `Type` - postavlja tip servisa. "`Idle`" znači da će se servis pokrenuti kada nema drugih važnijih zadatka za izvršavanje.
  - `RemainAfterExit` - ova opcija govori systemd-u da označi servis kao "`active`" čak i nakon što se izvršavanje završi.
  - `ExecStart` - ovdje se definira naredba koja će se izvršiti kada se pokrene servis. U ovom slučaju, poziva se skripta `/home/admin/twc/start_twc_logger.sh`.

- [Install]
  - `WantedBy` - označava koji sistemski cilj zahtijeva ovaj servis. U ovom slučaju, servis će biti pokrenut kada je postignut "`graphical.target`". Kada se postigne "`graphical.target`", to znači da je sustav spreman za prikazivanje grafičkog sučelja.

U suštini, skripta run-at-startup.service osigurava da se skripta start_twc_logger.sh automatski pokreće pri svakom pokretanju Raspberry Pi uređaja i izvršava navedene zadatke unutar grafičkog okruženja.

4.3.2 start_twc_logger.sh

Datoteka start_twc_logger.sh (Kod 4.15) je Bash skripta koja se koristi za pokretanje programa koji prikuplja i zapisuje podatke s Tesla Wall Connectora na Raspberry Pi uređaju, tj. za pokretanje twc.py skripte.

Bash (eng. *Bourne Again Shell*) je skripta napisana u Bash programskom jeziku. To je ljuska (eng. *shell*) za Unix i njemu slične operativne sustave, kao što su Linux i macOS. Bash omogućava korisnicima da izvršavaju naredbe i skripte za upravljanje operativnim sustavom, izvršavanje programa i automatizaciju zadataka. Bash skripte su tekstualne datoteke koje sadrže niz naredbi koje se izvršavaju slijedno, s mogućnošću dodavanja uvjeta, petlji, funkcija i drugih kontrolnih struktura kako bi se postigla željena funkcionalnost. Skripte se obično koriste za automatizaciju ponavljajućih zadataka, upravljanje sustavom i obradu podataka [21].

```
1 #!/bin/bash
2 sleep 5
```

```
3 DISPLAY=:0 lxterminal --command "cd /home/admin/twc/; python3 /home
    /admin/twc/twc.py"
```

Kod 4.15: Skripta start_twc_logger.sh

Objašnjenje start_twc_logger.sh skripte [2]:

- `#!/bin/bash` - ova linija govori operativnom sustavu da koristi Bash interpreter za izvršavanje skripte.
- `sleep 5` - čeka 5 sekundi prije nego što nastavi izvršavanje ostatka skripte. To može biti korisno kako bi se osiguralo da se svi potrebni resursi i usluge pravilno pokrenu prije nego što se izvrši sljedeći korak.
- `DISPLAY=:0` - postavlja X11 display na 0 (osnovni ekran).
- `lxterminal` - naredba pokreće LX Terminal emulator (Terminal emulator je program koji omogućava korisnicima interaktivno komuniciranje s operativnim sustavom putem naredbenog retka citelink25).
- `-command` - opcija koja omogućuje pokretanje naredbi unutar terminala.
- `cd /home/admin/twc/; python3 /home/admin/twc/twc.py` - to su naredbe koje će se izvršiti unutar terminala. Prvo se mijenja trenutni direktorij na `/home/admin/twc/`, a zatim se pokreće Python 3 skripta `/home/admin/twc/twc.py`.

U konačnici, ova skripta čeka 5 sekundi, postavlja odgovarajući X11 display i pokreće LX terminal s naredbama za promjenu direktorija i pokretanje Python skripte koja će prikupljati podatke s Tesla Wall Connectora.

5 | Instalacija data loggera

Predviđeno je da data logger radi na Raspberry Pi računalu koje će biti uključeno cijelo vrijeme te će dohvaćati podatke s Tesla wall connectora preko WiFi-ja i slati ih u InfluxDB gdje će podaci biti prikazani u Dashboardu. U početku je potrebno spojiti Tesla Wall Connector Gen 3 kućni punjač za električna vozila na lokalnu WiFi mrežu te saznati IP adresu punjača. Kako bi mogli vidjeti podatke koji se sakupljaju iz TWC data logger-a na Raspberry Pi računalu, podatci se direktno šalju u InfluxDB stoga je potrebno otvoriti korisnički račun na InfluxDB platformi.

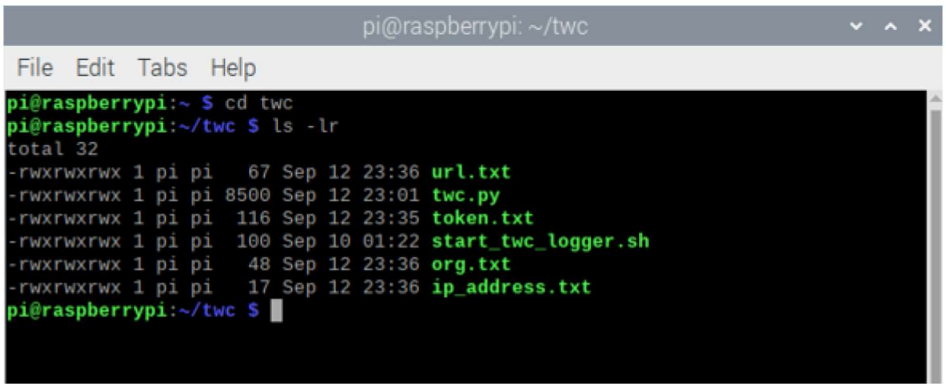
5.1 InfluxDB

InfluxDB korisnički profil može se napraviti na stranici https://cloud2.influxdata.com/signup ili treba instalirati InfluxDB preko Dockera kako bi bio dostupan lokalno što se može učiniti prema uputama na stranici: https://www.influxdata.com/blog/how-to-setup-influxdb-telegraf-and-grafana-on-docker-part-1/. Nakon što je otvoren profil potrebno je u InfluxDB-u napraviti "bucket" naziva "baza".
Nakon toga potrebno je kreirati "Dashboard" tako da se u lijevom izborniku u InfluxDB-u odabere "Dashboard" nakon čega je potrebno odabrati opciju "Import Dashboard" i dodati datoteku naziva "tesla_wall_connector_gen_3.json" čija je poveznica za preuzimanje s GitHub-a dana u poglavlju 4.1.1. Upravo u tom novom dashboardu (Slika 2.1) naziva "Tesla Wall Connector Gen 3" kasnije je moguće gledati podatke koji se dohvaćaju s punjača.

5.2 Raspberry Pi

Prije namještanja data loggera na Raspberry Pi, potrebno je provjeriti je li Raspberry Pi ažuriran te ima li najnoviju verziju Pythona. Procedura za instaliranje najnovije verzije Pythona dostupna je na linku: https://allurcode.com/install-latest-version-of-python-on-raspberry-pi/ . Nakon što je RPi ažuriran potrebno je otvoriti "File manager" te u otvoreni folder putanje `/home/username` (pri čemu je "*username*" naziv korisničkog računa koji se koristi na RPi-ju) dodati mapu naziva "*twc*" koja je dostupna na GitHub-u. (U ovim uputama koristit će se korisničko ime "*pi*" budući da je to ime zadano na svim RPi uređajima.)
Potrebno je otvoriti "Terminal" te unijeti naredbu `cd twc` kako bi ušli u twc folder. Potom se u terminal upiše `ls -lr` kako bi se provjerilo jesu li sve potrebne dato-

teke u folderu i jesu li omogućena sva dopuštenja za njih. Sve bi trebalo biti kao na slici 5.1.



Slika 5.1: Prikaz svih datoteka s potrebnim dopuštenjima.

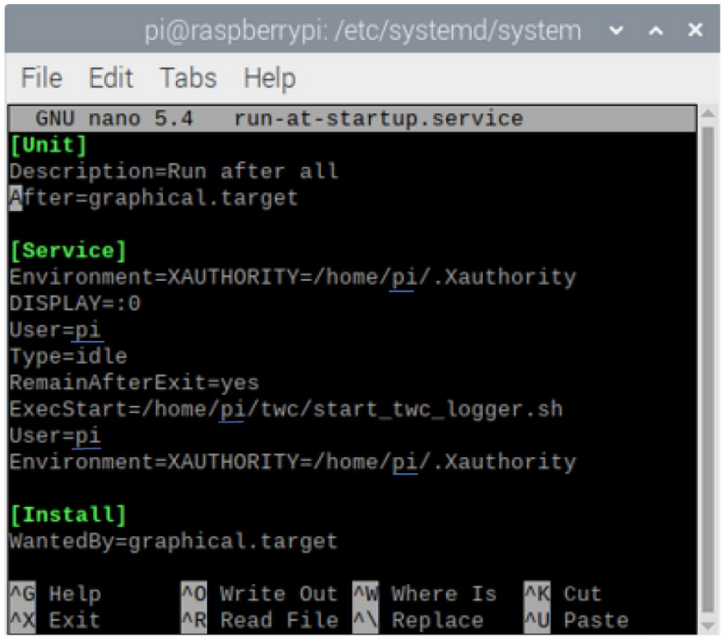
Potom je potrebno pokrenuti naredbu `cat start-twc-logger.sh` kako bi se provjerilo odgovara li putanja do twc datoteke. Ako se korisnikovo korisničko ime razlikuje od "*pi*", svugdje umjesto "*pi*" treba upisati željeno korisničko ime. Na slici 5.2 dan je primjer, tekst podcrtan plavom bojom treba zamijeniti željenim korisničkim imenom. Datoteka se može uređivati naredbom `nano start_twc_logger.sh` .

Slika 5.2: Prikaz start_twc_logger.sh datoteke s korisničkim imenom "*pi*".

Nakon toga treba upisati naredbu `cd/etc/systemd/system/` kako bi se ušlo u folder "system" u kojem se nalaze skripte koje pozivaju servise. U taj folder dodaje se novi servis naziva "run-at-startup" s naredbom `sudo touch run-at-startup.service`. Zatim, s naredbom `sudo nano run-at-startup.service` se otvara tekst editor te se upisuje tekst koji je dan na GitHub-u u folderu twc_data_logger u datoteci run-at-startup.service. Datoteku je opet potrebno prilagoditi vlastitom uređaju tako što svako pojavljivanje korisničkog imena "*pi*" treba zamijeniti željenim korisničkim imenom. Na slici 5.3 plavom bojom je označen tekst koji treba zamijeniti željenim korisničkim imenom.

U terminal se zatim upisuje naredba `sudo systemctl enable run-at-startup.service` kako bi se omogućilo da se servis izvršava svaki put. (Može se pokrenuti naredba `sudo systemctl start run-at-startup.service` kako bi se provjerilo hoće li se otvoriti novi terminal sa skriptom. Zatim naredba `sudo systemctl status run-at-startup.service` da bi se provjerio status servisa.) Na kraju, potrebno je još samo pokrenuti naredbu `sudo systemctl restart run-at-startup.service`.

Sada će se Python skripta za prikupljanje podataka sama pokretati pri svakom pokretanju Raspberry Pi računala te će se pokrenuti terminal u kojem će se ispisivati

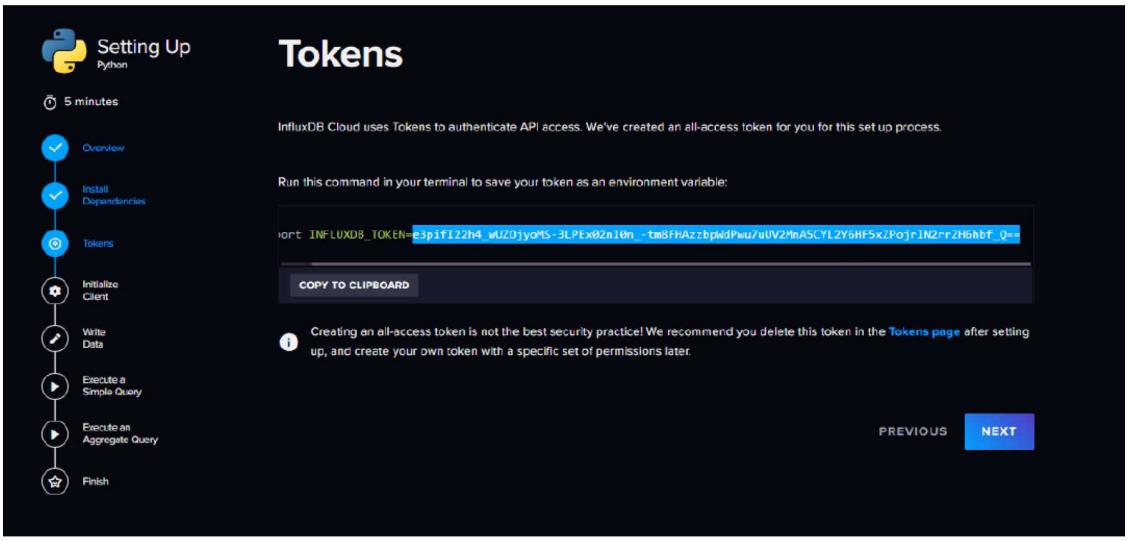


Slika 5.3: Primjer run-at-startup servisa s korisničkim imenom "*pi*"

poruka o stanju veze s punjačem. Pri prvom pokretanju skripte bit će potrebno upisati IP adresu punjača nakon čega će se ona trajno spremiti u datoteku naziva "ip_address.txt". Kako bi se podaci slali na željeni InfluxDB Cloud profil, pri prvom pokretanju je također potrebno unijeti token, url i org koj se mogu pronaći na InfluxDB profilu kao što je opisano u poglavlju 5.3. Uneseni token, url i org će se spremiti u zasebne datoteke "token.txt", "url.txt" i "org.txt" tako da ih nakon prvotnog unošenja više nije potrebno unositi. Ako kasnije ipak bude potrebe da se promijeni naprimjer prvotno unesena IP adresa to se može učiniti tako da se uđe u twc folder s naredbom `cd twc` te se nakon toga upiše `nano ip_address.txt` zbog čega će se otvoriti tekst editor u koji se potom upiše druga željena IP adresa i spreme se promjene. Isto se može primijeniti pri promijeni tokena, org-a i url-a.

5.3 InfluxDB Cloud autorizacija

Svaki InfluxDB korisnik ima vlastite podatke za autorizaciju (token, org, url) koji su potrebni kako bi se data logger povezao sa InfluxDB-om. Podaci se mogu pogledati u InfluxDB-u tako da se na početnoj stranici odabere Python. Nakon toga prikazat će se rubrika "Setting Up Python" u kojoj se treba u lijevom izborniku odabrati "Tokens". Plavo označeni dio sa slike 5.4 je dio koji se treba upisati pri prvom pokretanju skripte.
Zatim otvorimo "Initialize" s lijevog izbornika i tamo možemo vidjeti podatke "org" i "url" (Slika 5.5) koje je također potrebno upisati pri prvom pokretanju skripte, bez navodnika.



Slika 5.4: Primjer tokena.

```
org = "dorotea.osmanovic1@gmail.com"
url = "https://europe-west1-1.gcp.cloud2.influxdata.com"
```

Slika 5.5: Primjer org-a i url-a.

Literatura

[1] *API Reference, InfluxDBClient*, URL: https://influxdb-client.readthedocs.io/en/latest/api.html#

[2] *Bash Scripting – Working of Bash Scripting*, URL: https://www.geeksforgeeks.org/bash-scripting-working-of-bash-scripting/

[3] Dr. Roy Thomas Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, 2000, URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[4] *Get started with InfluxDB Cloud*, URL: https://ryanstutorials.net/bash-scripting-tutorial/bash-script.php

[5] *How To Run Long-running Scripts on a Raspberry Pi*, URL: https://www.tomshardware.com/how-to/run-long-running-scripts-raspberry-pi

[6] *How to Use the JSON Module in Python – A Beginner's Guide*, URL: https://www.freecodecamp.org/news/how-to-use-the-json-module-in-python/

[7] *HTTP Methods*, URL: https://restfulapi.net/http-methods/

[8] *Hypertext Transfer Protocol (HTTP)*, URL: https://www.extrahop.com/resources/protocols/http/

[9] *LXTerminal*, URL: https://www.computerhope.com/jargon/l/lxterminal.htm

[10] *Manage buckets*, URL: https://docs.influxdata.com/influxdb/v2/organizations/buckets/

[11] *Python certifi: How to Use SSL Certificate in Python*, URL: https://appdividend.com/2022/06/01/python-certifi/

[12] *Python client library*, influxdata, URL: https://docs.influxdata.com/influxdb/cloud/api-guide/client-libraries/python/

[13] *Python datetime module*, URL: https://www.geeksforgeeks.org/python-datetime-module/

[14] Python dokumentacija *time — Time access and conversions*, URL: https://docs.python.org/3/library/time.html

[15] *Python Requests Module*, URL: https://www.w3schools.com/python/module_requests.asp

[16] *Python time Module*, URL: https://www.programiz.com/python-programming/time#google_vignette

[17] *Python Try Except*, URL: https://www.w3schools.com/python/python_try_except.asp

[18] *REST APIs: How They Work and What You Need to Know*, URL: https://blog.hubspot.com/website/what-is-rest-api

[19] *Tesla Wall Connector Gen 3 RESTful*, URL: https://community.home-assistant.io/t/tesla-wall-connector-gen-3-restful/311670/9

[20] *The ultimate guide on using systemd to autostart scripts on the Raspberry Pi*, URL: https://www.thedigitalpictureframe.com/ultimate-guide-systemd-autostart-scripts-raspberry-pi/?utm_content=cmp-true

[21] *What is a Bash Script?*, URL: https://ryanstutorials.net/bash-scripting-tutorial/bash-script.php

[22] *What Is An API (Application Programming Interface)?*, URL: https://aws.amazon.com/what-is/api/

[23] *What Is an API (Application Programming Interface)? Meaning, Working, Types, Protocols, and Examples*, URL: https://www.spiceworks.com/tech/devops/articles/application-programming-interface/

[24] *What is an API: Definition, Types, Specifications, Documentation*, URL: https://www.altexsoft.com/blog/engineering/what-is-api-definition-types-specifications-documentation/

[25] *What is a REST API?*, URL: https://www.ibm.com/topics/rest-apis

[26] *What is a REST API?, Red Hat*, URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api

[27] *What is HTTP?*, URL: https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/