INTERNET STVARI IN KIBERFIZIČNI SISTEMI – PROTOTIPNA ZASNOVA

ANDREJ ŠKRABA
Univerza v Mariboru, Fakulteta za organizacijske vede, Kranj, Slovenija
andrej.skraba@um.si

Sinopsis Izveden je krajši pregled raziskav na področju interneta stvari in kiberfizičnih sistemov. Podrobneje je opisan pristop k razvoju prototipov interneta stvari in kiberfizičnih sistemov z modulom esp32. Izpostavljeni so ključni koncepti pri razvoju. Opredeljen je nabor orodij za prototipiranje ter različni pristopi k prototipiranju. Predstavljen je koncept samostojnega razvoja prototipnih rešitev ter koncept razvoja z uporabo obstoječe mrežne opreme. Obravnavan je prenos sporočil preko mehanizma zahteve in odgovora ter preko spletnega vtičnika. Opisan je pristop k razvoju uporabniškega vmesnika. Opredeljena je zasnova sistema za upravljanje več esp32 modulov. Izvedena je primerjava hitrosti prenosa sporočil preko mehanizma zahteve in odgovora ter spletnega vtičnika. Podana so izhodišča za nadaljnji razvoj.

Ključne besede:
internet stvari,
kiberfizični sistem,
organizacija,
informacijski
sistem,
prototipiranje

Univerzitetna založba
Univerze v Mariboru

DOI https://doi.org/10.18690/um.fov.6.2022.9
ISBN 978-961-286-637-2

INTERNET OF THINGS AND CYBER-PHYSICAL SYSTEMS – PROTOTYPE DESIGNA

ANDREJ ŠKRABA
University of Maribor, Faculty of Organizational Sciences, Kranj, Slovenia
andrej.skraba@um.si

Abstract An overview of research in the field of the Internet of Things and Cyber-physical systems has been conducted. The approach to the development of prototypes of the Internet of Things and Cyber-physical systems with the esp32 module is described in more detail. Key concepts in development are highlighted. A set of prototyping tools and different approaches to prototyping are defined. The concept of independent development of prototype solutions and the concept of development using existing network equipment are presented. The transmission of messages via the request and response mechanism and via the WebSocket is discussed. An approach to user interface development is described. The design of a system for managing several esp32 modules is defined. A comparison of the speed of message transmission via the request and response mechanism and the WebSocket is performed. Gidelines for further development are given.

Keywords: cyber-physical system, speech recognition, wheelchair, cloud information system, prototyping

https://doi.org/10.18690/um.fov.6.2022.9 DOI
978-961-286-637-2 ISBN

University of Maribor Press

1 Uvod

Področje interneta stvari in kiberfizičnih sistemov neustavljivo posega na najrazličnejša področja sodobnih informacijskih sistemov, kot npr. spremljanje biomedicinskih podatkov (Kofjač et al., 2018; Škraba et al., 2017, 2016; Škraba, Kolozvari, et al., 2019), spremljanje razvoja prototipnih avtonomnih vozil (Skraba et al., 2016) in pripomočkov za gibalno ovirane osebe (Koložvari, 2020; Koložvari et al., 2019; Skraba et al., 2019, 2014, 2015; Škraba et al., 2015). Tu velja omeniti možnost enostavnega prototipiranja tovrstnih naprav s pomočjo dostopnih mikrokontrolerjev, kot npr. Arduino (Škraba et al., 2016; Stojanović et al., 2020), ARM razvojnih plošč oz. računalnikov (Skraba et al., 2016; Škraba et al., 2015, 2018; Škraba, Stanovov, et al., 2019), modulov ESP8266 (Škraba et al., 2017, 2016) ESP32 (Škraba, Kolozvari, et al., 2019). Pri tem velja omeniti, da so stroški razvoja inovativnih rešitev na področju interneta stvari in kiberfizičnih sistemov zaradi dostopnosti strojne opreme lahko relativno nizki. S tem je omogočeno delo s strojno opremo, ki je neposredno povezana z internetom in njegovimi uporabniki tudi v klasičnih študijskih programih, ki pokrivajo področje informacijskih sistemov. Tako je za sodoben študij informatike pomembno, da programska koda, ki jo napišemo, tudi zaznava okolje in hkrati omogoča interakcijo s fizičnim okoljem. Tu torej želimo npr. premakniti določen objekt ali spremeniti njegovo zgradbo. Pomembna lastnost naprav interneta stvari in kiberfizičnih sistemov je medsebojna povezanost. Razvoj brezžičnih omrežij in mobilne telefonije je v zadnjem času omogočil razvoj cenovno dostopnih modulov ESP8266 in ESP32, s pomočjo katerih enostavno izvedemo priključitev poljubne strojne opreme na omrežje. Tu lahko uporabimo tudi brezžično povezavo modrega zoba (»BlueTooth«), vendar pa je predvsem pomembna povezava preko omrežja WiFi. Pomembni tehnologiji za prenos podatkov sta tudi LoraWAN (Adelantado et al., 2017; Ali et al., 2019) kakor tudi 5G (Li et al., 2018). S pomočjo omenjenih tehnologij lahko najrazličnejša analogna elektronska vezja in druge naprave priključimo na omrežje. Sicer je to le en del celotne zgodbe. Naprave namreč lahko pridobijo popolnoma nove tehnične lastnosti z možnostjo povezave z internetom in njegovimi uporabniki. Prav tako je moč razviti sisteme, ki vključujejo več uporabnikov, npr. spremljanje biomedicinskih podatkov večje skupine bolnikov (Škraba et al., 2017; Škraba, Kolozvari, et al., 2019), kar je zlasti pomembno v času pandemije Covid-19 (Stojanović et al., 2020).

Tako lahko identificiramo več področij, na katera pomembno vplivajo internet stvari in kiberfizični sistemi:

- razvoj strojne opreme,
- razvoj omrežnih aplikacij,
- integracija novih tehnologij,
- izkoriščanje oblačnih informacijskih sistemov,
- izobraževanje.

Področje izobraževanje, ki je sicer navedeno kot zadnje, ima poseben pomen. Platforma Arduino se je namreč razvila za potrebe izobraževanja na »Interaction Design Institute« v Italiji (Severance, 2014), podobno kot ARM platforma Raspberry Pi (Severance, 2013), torej tudi za potrebe izobraževanja.

Kiberfizični sistem lahko definiramo kot mehanizem, ki je upravljan in nadziran s pomočjo računalniških algoritmov v tesni povezavi z internetom in njegovimi uporabniki. Pod kibernetski razumemo tudi prisotnost v navideznem, virtualnem prostoru. Tako se tu razume, da za nek kiberfizični sistem obstaja dvojnik ali dvojček v virtualnem okolju. Idealno bi bil takšen dvojček realiziran kot simulacijski model v navidezni resničnosti, kjer uporabnik ne bi mogel ločiti med realnim sistemom in digitalnim dvojčkom. Tehnične rešitve gredo vsekakor v to smer, vendar pa bo potreben še nadaljnji razvoj.

Internet stvari opredelimo kot podmnožico kiberfizičnih sistemov, saj so kiberfizični sistemi povezani z okoljem in lahko z njimi vplivamo na realno okolje. Pri internetu stvari predpostavimo predvsem spremljanje parametrov okolja in ne same interakcije z realnim svetom v smislu spremembe pozicije fizičnih objektov ali njihove zgradbe.

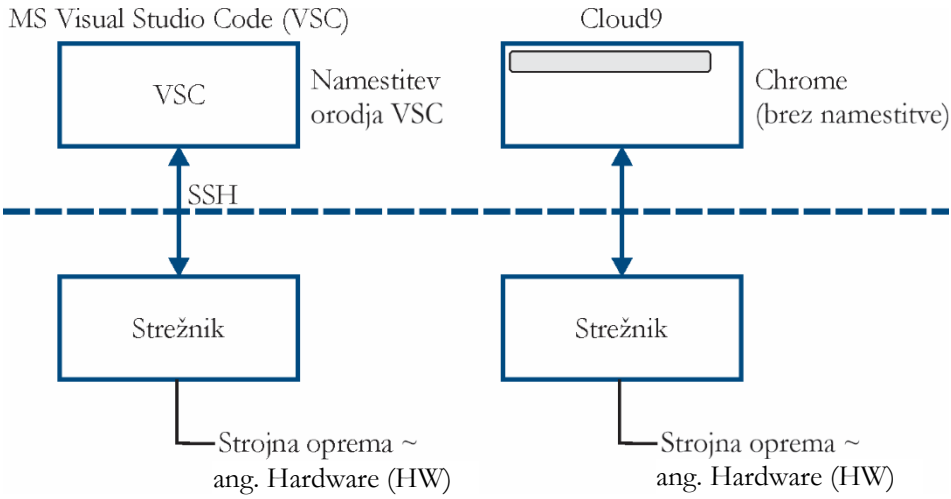
Zaradi splošne razširjenosti modula esp32 bomo v nadaljevanju opisali pristop k razvoju sistemov z modulom esp32. Obravnavali bomo prenos sporočil med klientom in modulom preko mehanizma zahteve in odgovora ter prenosa preko spletnega vtičnika.

2 Orodja za prototipni razvoj

Nabor orodij za prototipni razvoj na področju interneta stvari in kiberfizičnih sistemov je relativno velik. V naboru orodij omenimo dvoje urejevalnikov kode. Na prvem mestu omenimo orodje Cloud9, ki je bilo do nedavna prosto dostopno (Skraba et al., 2016; Škraba et al., 2018; Škraba, Stanovov, et al., 2019; Soare et al.,

2021). Pri orodju Cloud9 omenimo pomembno funkcionalnost, tj. možnost, da urejevalnik kode poženemo neposredno iz strojne opreme. Npr.: če imamo kot strojno opremo uporabljen Raspberry Pi z nameščenim operacijskim sistemom Linux, lahko do urejevalnika kode dostopimo preko brskalnika. Omenjena funkcionalnost je pomembna pri razvoju programske opreme, kjer imamo lahko več strojnih komponent. Z dostopom do vsake komponente v brskalniku lahko tako poenostavimo razvoj programske opreme.

V zadnjem času je na popularnosti pridobilo integrirano razvojno orodje Microsoft Visual Studio Code (Del Sole, 2021). Tu velja omeniti, da je možno orodje Visual Studio Code prav tako izvajati na strežniku, npr. ARM računalniku Raspberry Pi, in dostopati do integriranega razvojnega okolja preko brskalnika. Slika 1 prikazuje dostop do strojne opreme s pomočjo integriranih razvojnih okolij (IDE) MS Visual Studio Code ali Cloud9.



Slika 1: Dostop do strojne opreme s pomočjo integriranih razvojnih okolij (IDE) MS Visual Studio Code ali Cloud9

Vir: lasten.

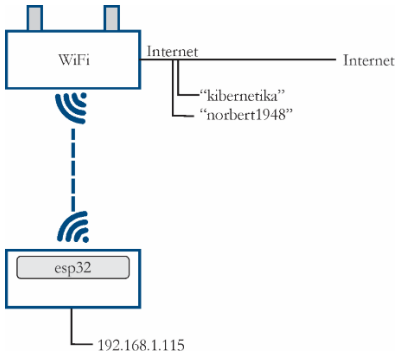
V obeh primerih je moč integrirano razvojno okolje namestiti na strojni opremi, npr. na ARM računalniku Raspberry Pi, ter dostopati preko spletnega brskalnika. Pri okolju Visual Studio Code pa je možno interakcijo s strojno opremo izvesti tudi s pomočjo protokola SSH. Z vidika uporabnika je bolj preprosta namestitev na sami strojni opremi, vendar pa je to povezano z zmogljivostjo strojne opreme. Namestitev

na modulu esp32 npr. ni mogoča na Raspberry Pi pa omenjena sistema brez težav lahko namestimo. Pri tem je pomembna tudi možnost razhroščevanja aplikacij, napisanih v JavaScrip na strani strežnika in tudi klienta. To je prav tako pomembno pri razvoju aplikacij interneta stvari in kiberfizičnih sistemov.

Visual Studio Code nam omogoča enostavno delo s strojno opremo s pomočjo vtičnika platforme platformio za kolaborativno delo na področju vgrajenih sistemov (Platformio.org, 2022).

2 Priključitev modula esp32

V nadaljevanju bomo opisali priključitev modula esp32 na brezžično omrežje. Pri tem potrebujemo brezžični WiFi usmerjevalnik (angl. »router«). Med modulom esp32 in brezžičnim usmerjevalnikom moramo vzpostaviti povezavo, ki nam omogoča interakcijo s strojno opremo, tj. s tipali in z aktuatorji. Običajno je brezžični WiFi usmerjevalnik priključen na internet, kar pa je odvisno od konfiguracije. Za priključitev bomo potrebovali identifikator oz. SSID (»Service Set Identifier«) brezžičnega usmerjevalnika, v našem primeru je SSID »kibernetika« ter geslo, ki je v našem primeru »norbert1948«. Slika 2 prikazuje priklop modula esp32 na brezžično WiFi omrežje. Po priklopu module esp32 pridobi statični IP naslov, npr. 192.168.1.115. IP naslov je lahko določen s pomočjo protokola DHCP (»Dynamic Host Configuration Protocol«). Pri tem je lahko IP naslov ob vsakem priklopu drugačen. Lahko pa s pomočjo nastavitev na usmerjevalniku fiksiramo IP naslov glede na unikatni identifikator MAC naslova (»Medium Access Control«) esp32 modula.



Slika 2: Priklop modula esp32 na brezžično WiFi omrežje
Vir: lasten.

Na ta način je lahko modul priključen v interno omrežje ali pa povezan z internetom. Za napredne funkcionalnosti moramo imeti na voljo povezavo z internetom.

Slika 3 prikazuje kodo za priklop modula ESP32 na WiFi omrežje. Uvodoma vključimo knjižnice za Arduino in WiFi. Nato sledi priklop na WiFi omrežje. V našem primeru je SSID omrežja »kibernetika«. Nato vnesemo geslo. Koda na modulu je sicer razdeljena na dva dela. V funkciji `setup()` zapišemo kodo, ki se sicer izvede le enkrat, tj. ob vklopu modula. V funkciji `loop()` pa zapišemo kodo, ki se neprestano izvaja. Tu je običajno zapisan kontrolni algoritem našega sistema. Za komunikacijo preko serijskih vrat določimo hitrost prenosa, ki je v našem primeru 115200 bits/s. Izpis je uporaben pri razhroščevanju kode in kontroli pravilnega delovanja.

```
#include <Arduino.h>
#include "WiFi.h"
const char* ssid = "kibernetika";
const char* password = "norbert1948";

void setup() {
  // tu zapišemo kodo za nastavitve, koda bo izvedena le enkrat:
  Serial.begin(115200);
  WiFi.begin(ssid, password); // WiFi omrežje poženemo

  while(WiFi.status() != WL_CONNECTED){
    delay(500);
    Serial.println("Povezovanje z Wifi omrežjem...");
  }

  Serial.println("Povezava z WiFi omrežjem je vzpostavljena.");
  Serial.print("IP naslov esp32 modula je: ");
  Serial.println(WiFi.localIP());

  // del, ki služi kot indikator, da je koda prenešena na modul
  pinMode(2, OUTPUT); // nožica št. 2 bo delovala kot izhod
  digitalWrite(2, HIGH); // na nožici 2 zapišemo visoko vrednost
  delay(750); // zakasnitev izvedbe v milisekundah
  digitalWrite(2, LOW); // na nožici 2 zapišemo nizko vrednost
```

```
  delay(750);
}

void loop() {
  // tu zapišemo kodo, ki se ponavljajoče izvaja:
}
```

Slika 3: Koda za priklop modula ESP32 na WiFi omrežje
Vir: lasten.

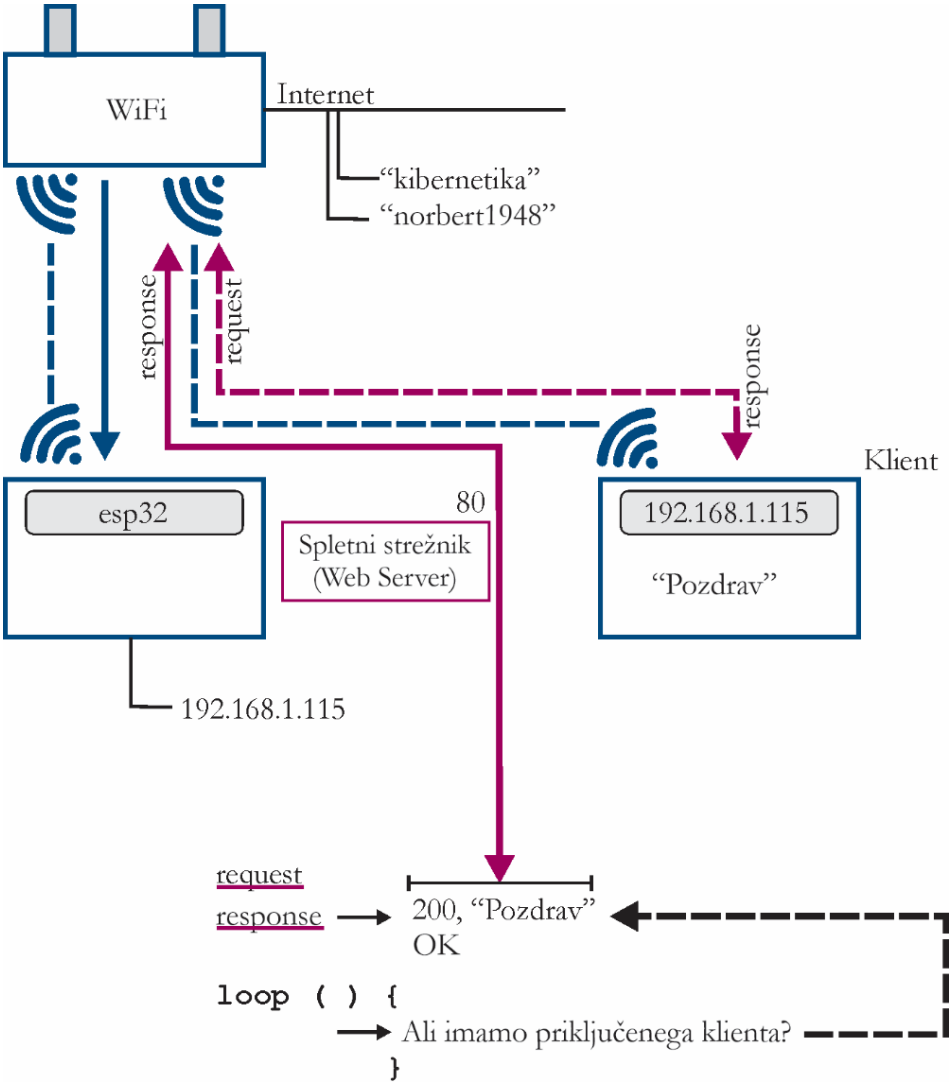
Ob zapisani kodi moramo tudi ustrezno definirati razvojno okolje Visual Studio Code/PlatformIO. Pri tem uporabimo platformo espresiff32 s programskim svežnjem Arduino. Hitrost komunikacije preko serijskega vmesnika, ki ga uporabimo za razhroščevanje, nastavimo na 115200 bit/s.

```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino
monitor_speed = 115200
```

Slika 4: Nastavitve razvojnega okolja Visual Studio Code/PlatfromIO
Vir: lasten.

Na modul ESP32 lahko dodamo spletni strežnik (slika 5), ki nam bo ob zahtevi klienta posredoval določen odgovor. Na sliki 5 je prikazan računalnik kot klient, ki je povezan z našim brezžičnim omrežjem (WiFi). V brskalniku želimo vpisati spletni naslov ESP32 modula, tj. IP naslov, npr. http://192.168.1.115. Potem ko uporabnik pritisne tipko enter, se zahteva oz. »request« posreduje na modul esp32. Spletni strežnik na modulu esp32 bo na omrežju ves čas »poslušal«, ali je določen klient priključen na modul. V primeru, da je klient prisoten in izda zahtevo, se le-ta posreduje modulu esp32. Na modulu esp32 bomo dodali kodo, ki bo ob zahtevi oz. »request« posredovala klientu odgovor, tj. »response«, ki bo v našem primeru le krajše besedilo. Tu bomo posredovali html kodo 200 za uspešen odgovor na zahtevo (OK), hkrati pa pošljemo klientu besedilo, v našem primeru »Pozdrav!«, ki se posreduje nazaj kot odgovor na zahtevo (»response«). Odgovor tako pride do klienta in v brskalniku klienta se izpiše pozdrav skupaj s html kodo 200 za uspešen odziv na zahtevo. Z dodano kodo za spletni strežnik na modulu esp32 bomo poslušali na

standardnih html vratih 80, ali je klient priključen. Preverjanje izvajamo v zanki loop(). V primeru, da je klient priključen in je prejeta zahteva s strani klienta, strežnik vrne odgovor s kodo 200 ter besedilom »Pozdrav!«. Modul esp32 lahko torej deluje kot spletni strežnik z izvedbo ključne funkcionalnosti mehanizma zahteve in odgovora oz. »request/response«.



Slika 5: Posredovanje odgovora klientu ob posredovani zahtevi – mehanizem zahteva/odgovor oz. »request/response«

Vir: lasten.

Za izvedbo funkcionalnosti spletnega strežnika na modulu esp32 moramo vključiti knjižnico WebServer.h (slika 6).

```
#include "WebServer.h"
```

Slika 6: Vključitev kode za funkcionalnost spletnega strežnika
Vir: lasten.

Spletni strežnik poženemo na vratih 80, privzeta vrata za posredovanje spletne strani. Tu ustvarimo WebServer z argumentom 80, ki prestavlja številko vrat:

```
WebServer server(80);
```

Slika 7: Ustvarimo spletni strežnik WebServer, ki posluša na vratih 80
Vir: lasten.

V nadaljevanju zapišemo funkcijo, ki opredeli, kaj naj se dogodi, če uporabnik vpiše korenski (»root«) naslov našega strežnika na esp32 modulu (slika 8). Npr.: če uporabnik vpiše v brskalnik naslov http: 192.168.1.115, bo funkcija prek strežnika posredovala klientu html kodo 200 ter sporočilo »Pozdravljen svet iz esp32!«

```
void handle_root(){
  server.send(200, "text/html", "Pozdravljen svet iz esp32!");
}
```

Slika 8: Funkcija handle_root(), ki opredeli odziv strežnika
Vir: lasten.

Strežnik tako klientu pošlje kot odziv na zahtevo kodo 200 ter sporočilo s pozdravom. Pri tem gre, kot omenjeno, za funkcionalnost zahteve in odziva.

Nato moramo dodati kodo (slika 9), ki opredeli, kaj se dogodi, ko strežnik prejme zahtevo ("/"). Tu je izvedena zahteva po korenskem imeniku. Ko vpišemo IP naslov v brskalnik, pokličemo funkcijo handle_root, ki vrne kratko sporočilo klientu (). Z ukazom server.begin() izvedemo zagon strežnika. Ta del kode je sicer zapisan v delu setup().

```
server.on("/", handle_root);
  server.begin(); // zagon strežnika
```

Slika 9: Klic funkcije handle_root ob prejemu zahteve "/"
Vir: lasten.

Po zagonu strežnika običajno še dodamo zakasnitev, npr. delaj(500). Tako z izvedbo kode počakamo na zagon strežnika.

V funkciji loop() zapišemo kodo za spremljanje prisotnosti klientov (slika 10). Tu »poslušamo« na vratih 80 in prestrezamo zahteve klientov (»request«). Za večjo stabilnost delovanja dodamo zakasnitev delay(4).

```
  server.handleClient();
  delay(4);
```

Slika 10: Spremljanje prisotnosti klientov
Vir: lasten.

Od spletnega strežnika pričakujemo, da nam kot odgovor poda ne le krajše sporočilo, temveč tudi spletno stran z ustrezno vsebino, tj. html stran. Namesto krajšega sporočila bomo tako ustvarili novo spremenljivko tipa string, v kateri bomo shranili vsebino spletne strani. Alternativna rešitev bi bila, da bi datoteko shranili na modul esp32 in nato posredovali vsebino s pomočjo SPIFFS (SPI Flash File System). Vsebino spletne strani najprej pripravimo v obliki samostojne html datoteke, ki jo nato prenesemo v obliko string spremenljivke. Slika 11 prikazuje vsebino spletne strani v spremenljivki HTML tipa String. Na koncu vsake vrstice dodamo poševnico, le-ta pove, da se definicija spremenljivke HTML nadaljuje. Če želimo spletno stran lepše izpisati na strani klienta, pa dodamo še kombinacijo \n za novo vrstico oz. »newline«.

```
String HTML = "<!DOCTYPE html>\n\
<html>\n\
    <head>\n\
        <meta charset='UTF-8'>\n\
    </head>\n\
    <body>\n\
        <h1>Pozdravljen svet na križišču tehnologij!</h1>\n\
```

```
    </body>\n\
</html>";
```

Slika 11: Vsebina spletne strani v spremenljivki HTML tipa string
Vir: lasten.

Pri posredovanju odgovora klientu glede na poslano zahtevo uporabimo funkcijo server.send(), podobno kot v predhodni izvedbi, vendar pa tokrat kot tretji argument ne zapišemo znakovnega niza neposredno, temveč posredujemo spremenljivko HTML (slika 12).

```
server.send(200, "text/html", HTML);
```

Slika 12: Kot tretji argument podamo spremenljivko HTML z vsebino spletne strani
Vir: lasten.

4 Od spletne strani do interakcije s strojno opremo

Standardno delovanje spletnega strežnika omogoča izvedbo mehanizma zahteve in odgovora. Glede na do sedaj predstavljeno realizacijo bi želeli izvesti interakcijo s strojno opremo. Modul esp32 ima več digitalnih vhodov in izhodov kakor tudi analognih vhodov in izhodov (PWM).

V nadaljevanju bi želeli, da se ob zahtevi: http://192.168.1.115/1 LED dioda na esp32 modulu prižge in ob zahtevi http://192.168.1.115/0 le-ta ugasne. Torej, če za naslovom vpišemo 1, sledi vklop, če zapišemo 0, sledi izklop.

Uvodoma bomo pripravili tri spremenljivke tipa string, HTML0, HTML1 in HTML3. Slika 13 prikazuje spremenljivko HTML0, kjer je pripravljeno sporočilo, ki ga posredujemo klientu, če le-ta vpiše http://192.168.1.115/0. Tako klientu sporočimo, da je bil prejet ukaz za izklop LED diode. Ime spremenljivke je skladno s sporočilom.

```
String HTML0 = "<!DOCTYPE html>\n\
<html>\n\
    <head>\n\
        <meta charset='UTF-8'>\n\
    </head>\n\
```

```
    <body>\n\
        <h2>Prejet ukaz za izklop LED diode.</h2>\n\
    </body>\n\
</html>";
```

Slika 13: Spremenljivka HTML0 z obvestilom o prejemu ukaza za izklop
Vir: lasten.

Slika 14 prikazuje spremenljivko HTML1, kjer je pripravljeno sporočilo, ki ga posredujemo klientu, če le-ta vpiše http://192.168.1.115/1. Ker je na koncu zapisana »1«, želimo LED diodo vklopiti. Tako klientu sporočimo, da je bil prejet ukaz za vklop LED diode. Ime spremenljivke je skladno s sporočilom.

```
String HTML1 = "<!DOCTYPE html>\n\
<html>\n\
    <head>\n\
        <meta charset='UTF-8'>\n\
    </head>\n\
    <body>\n\
        <h2>Prejet ukaz za vklop LED diode.</h2>\n\
    </body>\n\
</html>";
```

Slika 14: Spremenljivka HTML1 z obvestilom o prejemu ukaza za vklop
Vir: lasten.

Kot prednastavljeni odziv, torej, če uporabnik v brskalnik vpiše http://192.168.1.115/, bo strežnik odgovoril s sporočilom, naj uporabnik vnese ustrezno zahtevo v naslovno vrstico brskalnika (slika 15).

```
String HTML3 = "<!DOCTYPE html>\n\
<html>\n\
    <head>\n\
        <meta charset='UTF-8'>\n\
    </head>\n\
    <body>\n\
        <h2>HTTP strežnik je zagnan. V brskalnik vpišete
http://192.168.1.115/1 ali http://192.168.1.115/0 </h2>\n\
```

```
    </body>\n\
</html>";
```

Slika 15: Spremenljivka HTML3 s sporočilom o delovanju
Vir: lasten.

V globalnem prostoru moramo definirati spremenljivko LED1status tipa bool, ki bo ustrezala statusu LED diode (slika 16).

```
bool LED1status = LOW;
```

Slika 16: Spremenljivka LED1status za spremljanje statusa LED diode
Vir: lasten.

V primeru, da uporabnik v spletni brskalnik vnese http://192.168.1.115/, spletni strežnik vrne spremenljivko HTML3 z obvestilom o delovanju sistema (slika 17).

```
void handle_root(){
  server.send(200, "text/html", HTML3);

}
```

Slika 17: Strežnik vrne spremenljivko HTML3
Vir: lasten.

Funkcija, ki jo prikazuje slika 17, se sicer kliče ob zaznani zahtevi klienta, torej, ko v brskalnik vpišemo http://192.168.1.115 (slika 18).

```
server.on("/", handle_root);
```

Slika 18: Ob dostopu klienta do strežnika kličemo funkcijo handle_root
Vir: lasten.

Pri tem (slika 17) strežnik le vrne niz, ki je zapisan v spremenljivki HTML3. Druga funkcionalnost tu ni bila izvedena za razliko od naslednjih dveh odsekov kode.

Slika 19 prikazuje funkcijo handle_led1off(). Globalni spremenljivki LED1status določimo vrednost LOW, izvedemo izpis v serijski monitor ter klientu pošljemo sporočilo, da je bila zahteva sprejeta, kar je zapisano v spremenljivki HTML0.

```
void handle_led1off(){
  LED1status = LOW;
  Serial.println("GPIO2 Statuts: OFF");
  server.send(200, "text/html", HTML0);

}
```

Slika 19: Klientu sporočimo, da je zahteva za izklop sprejeta, globalni spremenljivki LED1status določimo vrednost LOW

Vir: lasten.

Funkcija, ki jo prikazuje slika 19, se sicer kliče ob zaznani zahtevi klienta za izklop LED diode, torej, ko v brskalnik za IP naslovom vpišemo ničlo, tj. http://192.168.1.115/0 (slika 20).

```
server.on("/0", handle_led1off);
```

Slika 20: Ob zahtevi klienta za izklop, tj. /0 kličemo funkcijo handle_led1off()

Vir: lasten.

Slika 21 prikazuje funkcijo handle_led1on(). Globalni spremenljivki LED1status določimo vrednost HIGH, izvedemo izpis v serijski monitor ter klientu pošljemo sporočilo, da je bila zahteva za vklop sprejeta, kar je zapisano v spremenljivki HTML1.

```
void handle_led1on(){
  LED1status = HIGH;
  Serial.println("GPIO2 Statuts: ON");
  server.send(200, "text/html", HTML1);
}
```

Slika 21: Klientu sporočimo, da je zahteva za vklop sprejeta, globalni spremenljivki LED1status določimo vrednost HIGH

Vir: lasten.

Funkcija, ki jo prikazuje slika 21 se sicer kliče ob zaznani zahtevi klienta za vklop LED diode, torej, ko v brskalnik za IP naslovom vpišemo enko, tj. http://192.168.1.115/1 (slika 22).

```
server.on("/1", handle_led1on);
```

Slika 22: Ob zahtevi klienta za vklop, tj. /1 kličemo funkcijo handle_led1on()
Vir: lasten.

V funkciji loop(), ki se neprekinjeno ponavlja, izvedemo preverjanje vrednosti globalne spremenljivke, ki se ob zahtevi /0 ali /1 spremeni. Glede na vrednost globalne spremenljivke LED1status s pomočjo funkcije digitalWrite na pin 2, določen s spremenljivko LED1pin, zapišemo visoko (HIGH) ali nizko (LOW) vrednost (slika 23).

```
void loop() {
// tu zapišemo kodo, ki se ponavljajoče izvaja:
  server.handleClient(); // spremljamo zahteve klientov

  if(LED1status){
    digitalWrite(LED1pin, HIGH);
  }
  else{
    digitalWrite(LED1pin, LOW);
  }

  delay(4); // za večjo stabilnost delovanja
}
```

Slika 23: Glede na vrednost globalne spremenljivke LED1status prižgemo ali ugasnemo LED diodo na pinu 2 (LED1pin)
Vir: lasten.

Osnovni primer interakcije s strojno opremo preko mehanizma zahteve in odgovora torej zajema naslednje funkcionalnosti:

- priklop strojne opreme na brezžično WiFi omrežje,
- vzpostavitev spletnega strežnika na esp32 modulu,

- prestrezanje zahtev klientov,
- izvedbo ustrezne funkcionalnosti ob različnih zahtevah klientov,
- interakcijo s strojno opremo glede na zahtevo klienta,
- odziv strojne opreme glede na spremembo vrednosti globalnih spremenljivk, katerih vrednosti se spreminjajo glede na zahteve klientov.

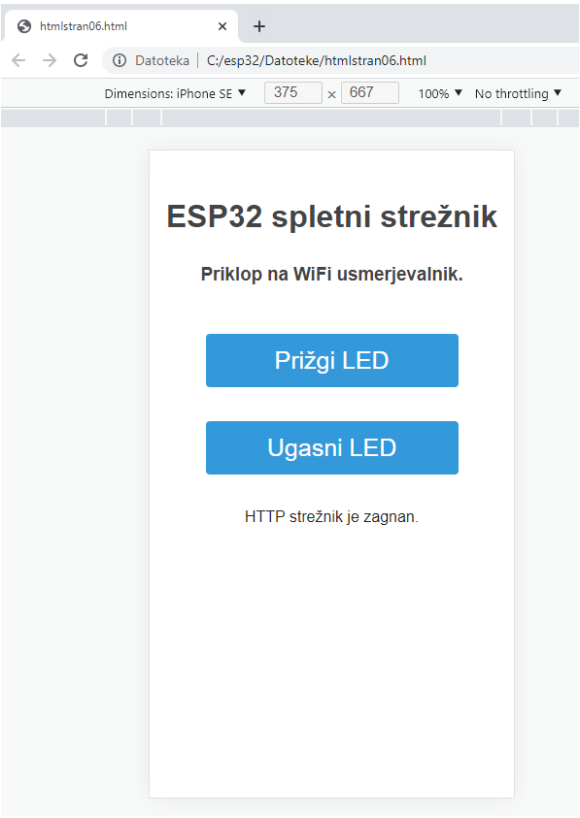
Z navedenimi točkami opišemo kompleksnost razvoja sistemov interneta stvari in kiberfizičnih sistemov. Tehnologija spleta je tako nadgrajena s strojno opremo. Na strojni opremi običajno delamo z jezikom C++, čeprav v zadnjem času srečamo tudi realizacije s pomočjo programskega jezika Python in JavaScript. Enostavnost programskih jezikov je pri tem precej pomembna, saj tako lahko pomembno zmanjšamo vprašanje kompleksnosti pri razvoju tovrstnih sistemov.

6 Razvoj uporabniškega vmesnika

Upravljanje strojne opreme neposredno z vnosom ustrezne zahteve na strani klienta z vidika enostavnosti uporabe in uporabniškega vmesnika ni ustrezno. Tako želimo realizirati vmesnik za interakcijo z modulom esp32, ki ga prikazuje slika 24. Spletni vmesnik je namenoma prikazan v Chrome (Google, 2018) brskalniku z odprtim razvijalskim vmesnikom. Razvijalec lahko izbere različne naprave, na katerih je izveden prikaz, v našem primeru je za prikaz izbrana naprava iPhoneSE. Vmesnik najprej razvijemo lokalno, nato pa ga prenesemo na modul. V našem primeru je bilo izvedeno oblikovanje s pomočjo CSS.

Spletni vmesnik (slika 24) vsebuje dva gumba za vklop in izklop LED diode. Hkrati je izpisano sporočilo, ki nam ga pošlje modul esp32 ob prejeti zahtevi s strani klienta. Ob pritisku na posamezni gumb se pošlje ustrezna zahteva do strežnika, ki je na esp32 modulu.

V spremenljivke HTML0, HTML1 ter HTML3 zapišemo vsebino spletnih strani, ki naj jih strežnik vrne ob zahtevah '/0', '/1' ter '/'.



Slika 24: Spletni vmesnik za interakcijo s strojno opremo
Vir: lasten, zajem zaslona.

Ključna dopolnitev kode se izvede na spletni strani klienta, kjer na posamezni gumb dodamo kodo za izvedbo ustrezne zahteve, tj. href='/1' ter href='/0' (slika 25).

```
<body>
  <h1>ESP32 spletni strežnik</h1>
  <h3>Priklop na WiFi usmerjevalnik.</h3>
  <a class='gumb' href='/1'>Prižgi LED</a>
  <a class='gumb' href='/0'>Ugasni LED</a>
  <p>HTTP strežnik je zagnan.</p>
</body>
```

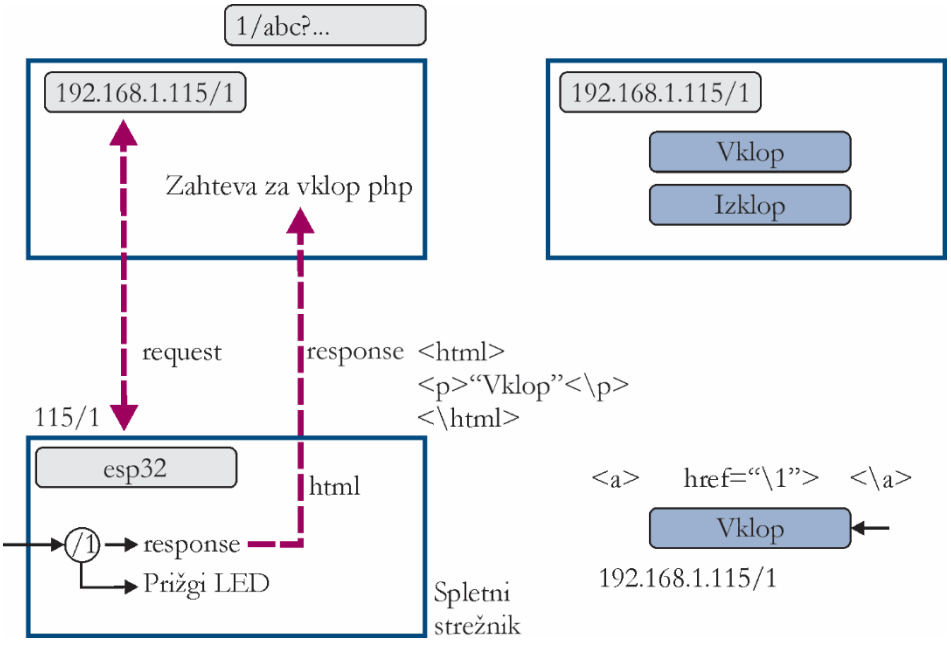
Slika 25: Izvedba zahteve ob pritisku na gumb v kodi html strani
Vir: lasten.

Delovanje opisanega primera prikazuje slika 26. Na levi strani je prikazan uporabniški vmesnik na mobilnem telefonu iPhone SE. V brskalniku Chrome je vpisan spletni naslov spletnega strežnika, ki je nameščen na modulu esp32, ki je prikazan na desni strani. Na modulu smo s pritiskom na gumb »Prižgi LED« prižgali modro LED diodo. Komunikacija med mobilnim telefonom in modulom esp32 poteka preko brezžičnega WiFi omrežja. Modul esp32 je sicer priključen na USB kabel le zaradi napajanja +5 V. V spodnjem delu uporabniškega vmesnika je izpisano sporočilo, ki se posreduje iz modula »Prejet ukaz za vklop LED diode«. Povratna informacija je pomembna za uporabnika. Za ustrezno povratno informacijo mora biti sicer zagotovljena povratna zanka, realizirana na strani strojne opreme s tipali in komunikacijo v realnem času. Izpisano sporočilo tako predstavlja le potrditev, da je strežnik prejel našo zahtevo, ne pa, ali je LED dioda trenutno dejansko prižgana ali ne. Informacijo o tem bi lahko pridobili s pomočjo tipala osvetlitve na LED diodi, ki bi nam omogočilo posredovanje informacije o tem, ali po prejemu ukaza za vklop dioda dejansko oddaja svetlobo ali ne.

Slika 26: Uporabniški vmesnik, prikazan na mobilnem telefonu (levo), ter modul esp32 (desno)
Vir: lasten.

Slika 27 prikazuje princip interakcije s strojno opremo. Na levi strani so prikazani mehanizem zahteve (»request«) in odzivi (»response«). Uporabnik v naslovno vrstico brskalnika vpiše zahtevo '/1', tj. http://192.168.1.115/1, ki se posreduje spletnemu strežniku na modulu esp32. V kodi, zapisani na esp32 modulu, prestrežemo zahtevo '/1'. Poleg odziva (»response«) izvedemo tudi vklop LED diode z ukazom

digitalWrite(2, HIGH). Tako se ob prižgani LED diodi klientu posreduje tudi odziv oz. odgovor (»response«) v obliki spletne strani, ki je v našem primeru zapisana v spremenljivki HTML1 tipa string. Na desni strani (slika 27) je prikazan uporabniški vmesnik. V tem primeru vnos naslova ni potreben, ustrezna zahteva se izvede ob pritisku na tipko »Prižgi LED«, kjer izvedemo ('href') oz. preusmeritev. Koda, ki je dodana gumbu, je tako: Prižgi LED.



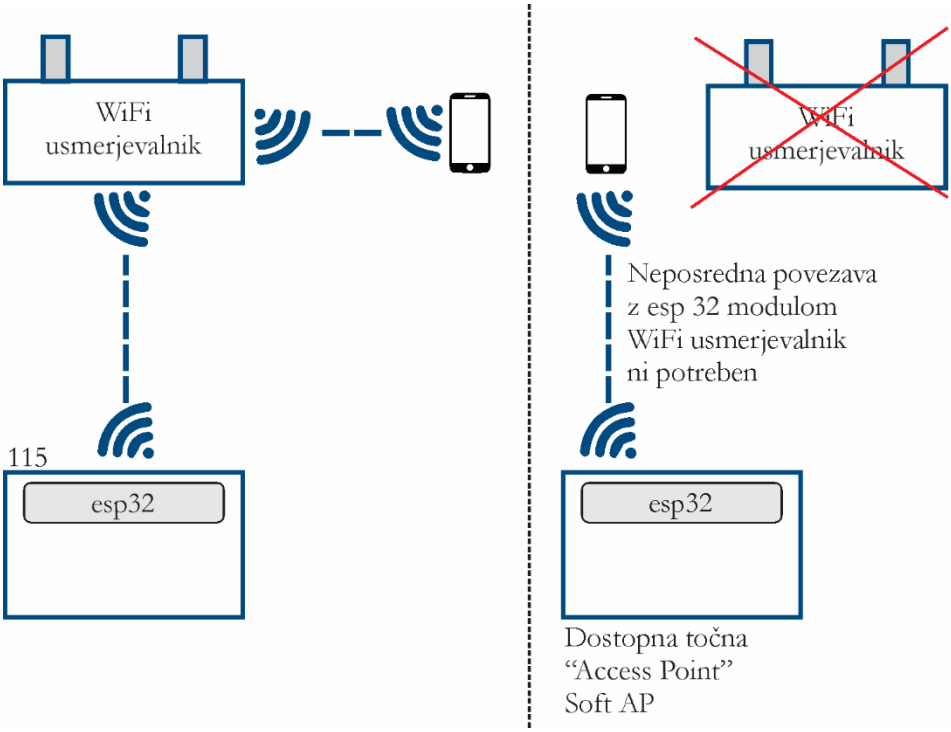
Slika 27: Posredovanje zahteve s strani klienta; levo – vnos zahteve '/1' v vrstico z naslovom, desno – posredovanje zahteve '/1' s pritiskom na gumb v uporabniškem vmesniku

Vir: lasten.

6 Modul esp32 kot dostopna točka

V predhodnem primeru smo modul esp32 povezali na WiFi usmerjevalnik. Preko usmerjevalnika smo z mobilnim telefonom dostopali do esp32 modula. Slika 28 v levem delu prikazuje princip priklopa, kjer potrebujemo WiFi usmerjevalnik.

Modul esp32 lahko deluje kot programska oz. mehka dostopna točka (»Soft Access Point – Soft AP«), kar je prikazano na desni strani (slika 28). Tu vidimo, da WiFi usmerjevalnika ne potrebujemo. Klient je preko mobilnega telefona neposredno povezan z modulom esp32. Na ta način imamo v sistemu komponento manj, izgubimo pa napredne funkcionalnosti usmerjevalnika ter dostop do interneta. Tako lahko enostavno realiziramo interakcijo uporabnika s strojno opremo s pomočjo mobilnega telefona. Najrazličnejše realizacije strojne opreme tako lahko neposredno upravljamo preko brezžične povezave z modulom esp32.



Slika 28: Modul esp32, priključen na WiFi usmerjevalnik (levo) in kot dostopna točka (desno)
Vir: lasten.

Za realizacijo moramo vključiti knjižnico za WiFi dostopno točko »Access Point AP« (slika 29). S pomočjo te knjižnice modul esp32 deluje kot dostopna točka, na katero se lahko brezžično priključimo, npr. s telefonom ali računalnikom.

```
#include <WiFiAP.h>
```

Slika 29: Vključitev knjižnice za dostopno točko »Access Point AP«
Vir: lasten.

V nadaljevanju moramo dostopno točko modula esp32 poimenovati. V našem primeru bo identifikator ssid oz. ime dostopne točke »esp32_1«. Določiti moramo tudi geslo za dostop (slika 30).

```
const char* ssid = "esp32_1";
const char* password = "12345678";
```

Slika 30: Poimenovanje dostopne točke modula esp32 in določitev gesla
Vir: lasten.

Programsko dostopno točko zaženemo znotraj funkcije setup(). Kot argument podamo identifikator točke ter geslo (slika 31).

```
WiFi.softAP(ssid, password);
```

Slika 31: Zagon programske dostopne točke
Vir: lasten.

"Soft Access Point AP" oz. "software enabled access point" nam tako omogoča, da npr. računalnik ali modul esp32, ki ni bil zasnovan kot dostopna točka, postane z ustreznim programjem brezžična dostopna točka. Prednastavljeni IP naslov, ki ga pridobi esp32 modul, je 192.168.4.1.

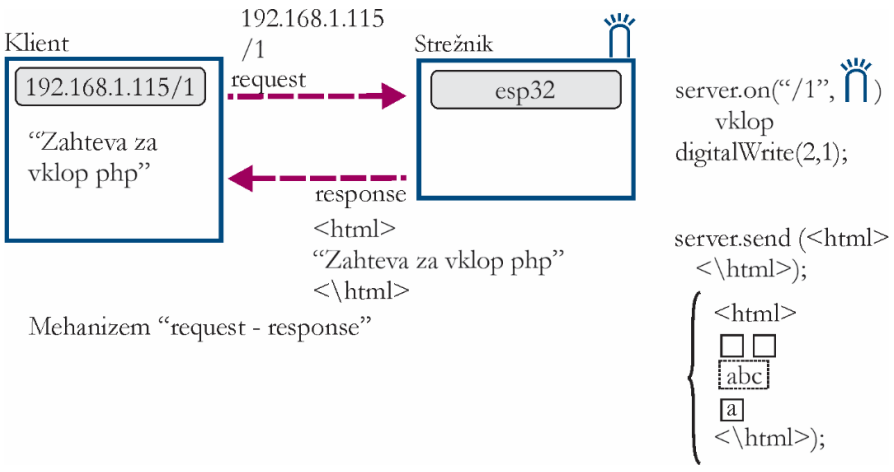
7 Komunikacija preko spletnega vtičnika WebSocket

Pri komunikaciji med klientom in strežnikom imamo na voljo več možnosti. V dosedanjih primerih smo uporabili možnost komunikacije po principu zahteve in odgovora. Tovrstna komunikacija ni stalna in tudi ne dvosmerna. Za potrebe stalne, dvosmerne komunikacije moramo uporabiti spletni vtičnik ali WebSocket. Z namenom, da bi pojasnili razliko, bomo v nadaljevanju nekoliko povzeli mehanizem delovanja celotnega sistema s pomočjo mehanizma zahteve in odgovora, tj. »request/response«, nato pa opisali še delovanje sistema s pomočjo komunikacije preko tehnologije WebSocket. Določen del opisa v prvem delu se nekoliko prekrije

s predhodno obravnavo, vendar pa želimo tu izvesti neposredno primerjavo med pristopoma.

V predhodnem delu smo torej že opisali interakcijo z modulom esp32 s pomočjo mehanizma zahteve in odgovora oz. »request/response«. Slika 32 prikazuje tovrstno interakcijo med klientom in strežnikom. Na klientu vnesemo naslov modula, npr. http://192.168.1.115/1. Na ta način prenesemo zahtevo '/1' na strežnik, v našem primeru je to modul esp32. Na modulu je nameščen spletni strežnik, ki nam omogoča izvedbo funkcionalnosti html strežnika. Na modul imamo priključeno strojno opremo, v našem primeru LED diodo. Na modulu esp32 programska koda neprestano preverja, ali je zahteva s strani klienta prejeta. V primeru, da strežnik prejme zahtevo za vklop '/1', izvedemo kodo za vklop LED diode, tj. digitalWrite(2, HIGH). Nato pa strežnik posreduje odgovor klientu 'Zahteva za vklop prejeta', kar se izpiše na uporabniškem vmesniku telefona oz. v spletnem brskalniku.

Na spletni strani, ki jo posredujemo kot odgovor, imamo lahko več gumbov, besedilo itd. Ob odgovoru ('server.send()') se posreduje celotna stran. Sicer smo spremenili le del spletne strani, npr. le izpisali obvestilo o prejeti zahtevi. Če bi bila spletna stran bolj kompleksna, bi ob spremembi npr. le ene črke posredovali celotno spletno stran, ki pa bi bila lahko precej obsežna. Tovrsten način interakcije med klientom in strežnikom je primeren le za manj zahtevne primere.

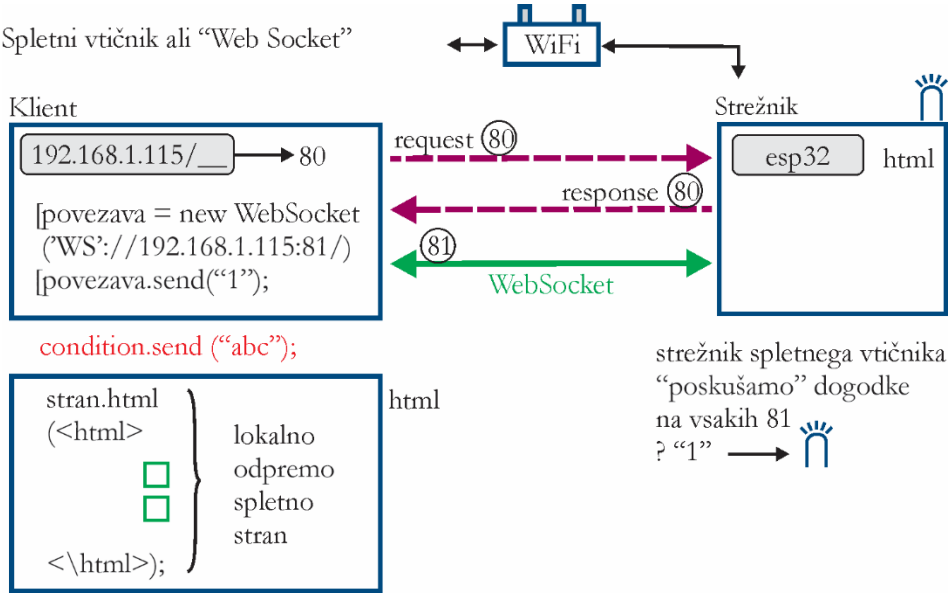


Slika 32: Interakcija z modulom preko mehanizma zahteve in odgovora oz. »request/response«

Vir: lasten.

Ena od možnosti, ki je primerna za bolj zahtevne realizacije, je komunikacija preko spletnega vtičnika ali »WebSocket«. Klient (slika 33) bo ob vzpostavitvi povezave, tj. ob vnosu naslova http://192.168.1.115, poslal zahtevo strežniku preko vrat 80. Kot odgovor bo strežnik na modulu esp32 posredoval spletno stran, ki bo vsebovala kodo za vzpostavitev povezave z modulom esp32 preko spletnega vtičnika št. 81 ('ws://192.168.1.115:81'). Hkrati se bo ob posredovanju spletne strani klientu na modulu esp32 vzpostavil strežnik, ki bo poslušal na vratih št. npr. 81. Na strani strežnika tako prestrezamo dogodke na vtičniku št. 81.

Med klientom in strežnikom se tako vzpostavi permanentna dvosmerna povezava preko spletnega vtičnika oz. »WebSocket«. Povezava je ves čas odprta in omogoča dvosmerno komunikacijo v realnem času, kar je primerno za realizacijo kontrolnih sistemov. Slika 33 prikazuje komunikacijo med klientom in strežnikom na modulu esp32 preko spletnega vtičnika »WebSocket« in interakcijo s strojno opremo.



Slika 33: Komunikacija med klientom in strežnikom na modulu esp32 preko spletnega vtičnika »WebSocket« in interakcija s strojno opremo

Vir: lasten.

Pri realizaciji moramo najprej vključiti knjižnico WebSocketsServer.h, ki omogoča dvosmerno stalno komunikacijo med klientom in strežnikom preko spletnega vtičnika (slika 34).

```
#include "WebSocketsServer.h"
```

Slika 34: Dodajanje knjižnice za dvosmerno stalno komunikacijo med klientom in strežnikom
Vir: lasten.

Na vratih 81 ustvarimo strežnik spletnih vtičnikov oz. 'WebSocketsServer', ki posluša na vratih 81 (slika 35).

```
WebSocketsServer webSocket = WebSocketsServer(81);
```

Slika 35: Ustvarimo strežnik, ki posluša na vratih 81
Vir: lasten.

V nadaljevanju moramo definirati funkcijo onWebSocketEvent, ki se izvede, kadarkoli dobimo sporočilo preko spletnega vtičnika preko 'WebSocket' oz. se dogodi dogodek - "event" na spletnem vtičniku. Funkcija onWebSocketEvent sprejme argumente: št. vtičnika, tip sporočila, kazalec na sporočilo ('payload') ter dolžino sporočila. Glede na tip sporočila s pomočjo stavka 'switch' izvedemo ustrezno funkcionalnost. Dogodki so lahko tipa npr. WStype_DISCONNECTED, WStype_CONNECTED, WStype_TEXT.

```
void
onWebSocketEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t length){
  switch (type)
  {
  case WStype_DISCONNECTED:
    {
    Serial.printf("[%u] Klient odklopljen!\n", num);
    }
    break;
  case WStype_CONNECTED:
    {
    IPAddress ip = webSocket.remoteIP(num);
```

```
    Serial.printf("[%u] Vpostavljena WebSocket povezava iz IP naslova ", num);
    Serial.println(ip.toString());
    }
    break;
  case WStype_TEXT:
  {
    String besedilo = String((char) payload[0]);

    if(besedilo == "1"){
      digitalWrite(LED1pin, 1); // zapis visoke vrednosti
    }
    if(besedilo == "0"){
      digitalWrite(LED1pin, 0); // zapis nizke vrednosti
    }
  }
  default:
    break;
  } // KONEC switch stavka
} // KONEC funkcije onWebSocketEvent
```

Slika 36: Ob dogodku na spletnem vtičniku ('WebSocket') določimo funkcionalnost glede na tip dogodka na spletnem vtičniku

Vir: lasten.

V primeru, da preko vtičnika posredujemo besedilo, izvedemo ustrezno kodo za vklop LED diode. Preneseni niz pretvorimo v spremenljivko besedila tipa string s pomočjo kode string besedilo = String((char) payload[0]). Če je besedilo "1", izvedemo vklop, če pa je besedilo enako "0", izvedemo izklop.

Slika 37 prikazuje spremenljivko HTML0 tipa string z vsebino spletne strani. Pomembna dopolnitev je vzpostavitev povezave preko spletnega vtičnika v delu zapisa JavaScript var povezava = new WebSocket('ws://192.168.1.115:81/'). Tu moramo upoštevati, da se sicer koda, ki se posreduje iz modula esp32, izvede v brskalniku klienta. Na ta način vzpostavimo stalno, dvosmerno povezavo med klientom in strežnikom oz. modulom esp32. Ob pritisku na gumb se sicer izvede ustrezna funkcija, npr. ob vklopu izvedemo funkcijo prižgi1(), ki preko povezave, vzpostavljene preko vtičnika 81, pošlje na strežnik oz. modul esp32 sporočilo '1', tj. povezava.send('1').

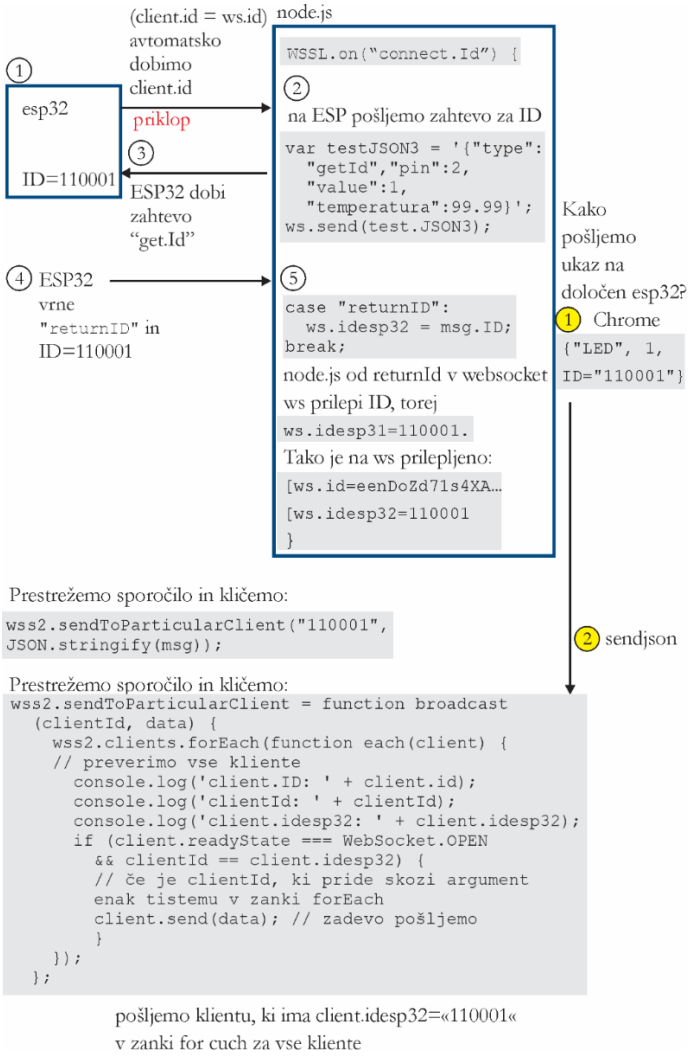
```
String HTML0 = "<!DOCTYPE html>\n\
<html>\n\
  <head>\n\
      <meta charset='UTF-8'>\n\
  </head>\n\
  <body>\n\
      <h2>Primer s spletnim vtičnikom WebSocket.</h2>\n\
      <button onclick='prižgi1()'>Prižgi LED</button>\n\
      <button onclick='ugasni1()'>Ugasni LED</button>\n\
      <script>\n\
          var povezava = new WebSocket('ws://192.168.1.115:81/');\n\
          function prižgi1(){\n\
              povezava.send('1');\n\
          }\n\
          function ugasni1(){\n\
              povezava.send('0');\n\
          }\n\
      </script>\n\
  </body>\n\
</html>";
```

Slika 37: Spremenljivka HTML0 tipa string s spletno stranjo
Vir: lasten.

8 Razvoj sistema za upravljanje več esp32 modulov

V primeru, da imamo več modulov esp32, ki jih želimo upravljati, moramo odgovoriti na več izzivov. Najprej moramo zagotoviti, da je razviti sistem dostopen za vse uporabnike. Tako moramo imeti ves čas prisoten spletni strežnik, ki mora delovati 24/7. Tovrstni strežnik je sicer lahko nameščen tudi v oblaku, mora pa biti dostopen kot klasični spletni strežnik. Slika 38 prikazuje korake pri delu z več moduli esp32 ob uporabi strežnika node.js. V prvem koraku (slika 38) moduel esp32 vzpostavi povezavo preko spletnega vtičnika. Klientu je avtomatsko določen identifikator, client.id. V delu na strežniku node.js prestrežemo povezavo z wss2.on("connection"). Takoj po vzpostavitvi povezave na modul esp32 pošljemo zahtevo za identifikator, v našem primeru je tip sporočila "getId". V tretjem koraku modul esp32 pridobi zahtevo "getId". V četrtem koraku modul esp32 vrne sporočilo "returId" z vrednostjo identifikatorja, v našem primeru je identifikator ID = 110001.

V petem koraku node.js strežnik na vtičnik ws prilepi identifikator ID, npr. ws.idesp32="110001". Tako imamo na vtičniku ws identifikator, npr. ws.id=eeuDoZdZ1S4Xf, ter identifikator esp32 modula, npr. ws.idesp32=110001. Na ta način smo združili identifikator vtičnika z identifikatorjem naprave. V primeru, da bi imeli več esp32 modulov, bi jim npr. lahko dodelili identifikatorje 110001, 110002, 110003 … Način identifikacije naprav je v domeni razvijalca. Pri tem je dobro, da upoštevamo standarde pri identifikaciji naprav.



Slika 38: Priklop modula esp32 preko vtičnika na node.js strežnik

Vir: lasten.

Na desni strani imamo prikazan mehanizem komunikacije klienta v Chrome s strežnikom node.js in nato z modulom esp32. Klient v prvem koraku pošlje ukaz na točno določen modul esp32 z identifikatorjem 110001 tako, da pošlje na node.js strežnik sporočilo z identifikatorjem ID="110001". Strežnik prestreže sporočilo in kliče funkcijo sendToParticularClient(). Kot argument podamo json sporočilo ter identifikator esp32 modula. V zanki preverjamo, ali je identifikator modula pravi. V primeru, da je logični pogoj izpolnjen, pošljemo sporočilo modulu esp32 z identifikatorjem 110001.

Poleg povezave preko mehanizma zahteve in odgovora ter povezave preko spletnega vtičnika, kjer je prvi klicatelj klient, omenimo tudi možnost, kjer je prvi klicatelj modul esp32. Prav tako obstaja možnost uporabe klica XMLHttpRequest.

8 Rezultati meritev hitrosti prenosa sporočil

Z namenom analize dveh različnih opisanih mehanizmov, tj. zahteva/odgovor (request/response) ter spletni vtičnik (WebSocket), smo testirali hitrosti prenosov sporočil.

Za mehanizem zahteve in odziva smo na strani klienta izvedli klic modula esp32 vsakih npr. 500 ms (slika 39).

```
var timeoutID = setInterval(prižgi1, 500);
function prižgi1(){
 location.href = '/1';
}
```

Slika 39: Klic modula esp32 vsakih 500ms
Vir: lasten.

Pri uporabi spletnega vtičnika (WebSocket) smo uporabili naslednjo kodo, kjer smo npr. vsakih 500 ms posredovali sporočilo modulu esp32 (slika 40).

```
var povezava = new WebSocket('ws://192.168.1.110:81/');
var timeoutID = setInterval(prižgi1, 500);
function prižgi1(){
  povezava.send('1');
}
```

Slika 40: Klic modula esp32 preko spletnega vtičnika vsakih 500 ms
Vir: lasten.

Pri testu je bil uporabljen WiFi router LinkSys ddwrt54, računalnik z Intel i5 procesorjem ter esp32 modul.

Tabela 1 prikazuje meritve prenosa sporočil na modul esp32 pri mehanizmu zahteve in odgovora (request/response) ter prenosa sporočil preko spletnega vtičnika. S t-testom na nivoju tveganja p=0.05 (t-test povp_rr=117.75, povp_ws=0.81, t= 9.579, n=4) lahko sklepamo, da so med dvema različnima pristopoma pomembne razlike pri odstopanju med želenim časovnim intervalom ter dejanskim.

Tabela 1: Meritev odzivov modula esp32 pri mehanizmu req/res ter prenosu preko spletnega vtičnika WebSocket

N=100	čas	100ms	500ms	1000ms	2000ms
req/res	povp.	184.61000	614.48000	1125.36000	2146.55000
	razlika [ms]	84.61000	114.48000	125.36000	146.55000
	delež razlike	0.84610	0.22896	0.12536	0.07328
	stdev	64.28895	2.19954	262.34635	297.09487
	CV	0.34824	0.00358	0.23312	0.13841
websocket	povp.	100.05000	499.72000	999.45000	2004.02000
	razlika [ms]	0.05000	-0.28000	-0.55000	4.02000
	delež razlike	0.00050	-0.00056	-0.00055	0.00201
	stdev	66.84119	33.52945	44.01753	68.03356
	CV	0.66808	0.06710	0.04404	0.03395

Izvedena meritev okvirno prestavlja razliko med načinoma prenosa sporočil. Pri izvedbi meritev je več različnih faktorjev, tako da bi morali samo eksperimentalno postavitev bolj podrobno opisati.

V nadaljnjih raziskavah velja pripraviti eksperiment, kjer bi spremljali krožno pot potovanja sporočila, saj je le-ta pomembna za razvoj kontrolnih sistemov. Prav tako bi morale biti meritve dopolnjene z mehanizmom spremljanja izgubljenih sporočil.

9 Zaključek

Razvoj interneta stvari in kiberfizičnih sistemov ne bi bil možen brez tehnološkega razvoja na področju strojne in programske opreme. S platformo Arduino ter razvojem JavaScript ter spletnih tehnologij je postalo možno, da enostavno integriramo uporabniške aplikacije s strojno opremo.

Platforma esp32 je v zadnjem času postala ena od prevladujočih pri razvoju interneta stvari in kiberfizičnih sistemov. Enostavni priklop v omrežje ter cenovna dostopnost omogočata razvoj naprednih aplikacij.

Zaradi povezave različnih tehnologij je razvoj tovrstnih sistemov zahteven, saj moramo razviti aplikacijo z uporabniškim vmesnikom na strani klienta, strežniški del, npr. na strežniku node.js, ter kontrolo strojne opreme na samem modulu. S predstavljenimi funkcionalnostmi smo podali izhodišča za razvoj tovrstnih sistemov in tudi koncepte za razvoj bolj zahtevnih aplikacij.

Tehnologija omogoča različne možnosti povezav med klientom in moduli. Z namenom opredelitve učinkovitosti smo izvedli primerjavo prenosov sporočil preko dveh različnih mehanizmov: zahteve in odgovora ter spletnega vtičnika. Spletni vtičnik je bolj primeren za aplikacije v realnem času. Mehanizem zahteve in odgovora pa na drugi strani zagotavlja stabilno delovanje.

Nadaljnji razvoj tovrstnih sistemov bo usmerjen v integracijo metod umetne inteligence in okrepitve povezav med uporabniki interneta. Le-to nam omogoča razvoj sistemov z izjemnimi tehničnimi lastnostmi.

Zahvala

Raziskavo je podprla Javna agencija za raziskovalno dejavnost Republike Slovenije (ARRS) v okviru raziskovalnega programa "Sistemi za podporo odločanju v digitalnem poslovanju", št.: P5-0018, ter bilateralnih projektov: "Razvoj kiberfizičnega sistema za nadzor stresa pri ogroženih posameznikih in skupinah", št. BI-ME/18-20-009, "Numerično-analitična obravnava biološko navdahnjenih algoritmov za upravljanje kiberfizičnih sistemov", št. BI-RU/19-20-034, "Evolucijski in biološko navdahnjeni algoritmi za učinkovito upravljanje kiberfizičnih sistemov in interneta stvari", št. BI-RU/16-18-040.

Literatura

Adelantado, F., Vilajosana, X., Tuset-Peiro, P., Martinez, B., Melia-Segui, J., & Watteyne, T. (2017). Understanding the Limits of LoRaWAN. *IEEE Communications Magazine*, *55*(9). https://doi.org/10.1109/MCOM.2017.1600613

Ali, Z., Henna, S., Akhunzada, A., Raza, M., & Kim, S. W. (2019). Performance Evaluation of LoRaWAN for Green Internet of Things. *IEEE Access*, *7*. https://doi.org/10.1109/ACCESS.2019.2943720

Del Sole, A. (2021). Visual Studio Code Distilled. In *Visual Studio Code Distilled*. https://doi.org/10.1007/978-1-4842-6901-5

Google. (2018). *Chrome DevTools*. Tools for Web Developers.

Kofjač, D., Stojanović, R., Kološvari, A., & Škraba, A. (2018). Designing a low-cost real-time group heart rate monitoring system. *Microprocessors and Microsystems*, *63*. https://doi.org/10.1016/j.micpro.2018.08.010

Kološvari, A. (2020). *Zasnova invalidskega vozička kot kiberfizičnega sistema in analiza vpliva uporabe na organizacijo rehabilitacijskega procesa* [Univerza v Mariboru]. https://dk.um.si/IzpisGradiva.php?lang=slv&id=76261

Kološvari, A., Stojanović, R., Zupan, A., Semenkin, E., Stanovov, V., Kofjač, D., & Škraba, A. (2019). Speech-recognition cloud harvesting for improving the navigation of cyber-physical wheelchairs for disabled persons. *Microprocessors and Microsystems*, *69*. https://doi.org/10.1016/j.micpro.2019.06.006

Li, S., Xu, L. Da, & Zhao, S. (2018). 5G Internet of Things: A survey. In *Journal of Industrial Information Integration* (Vol. 10). https://doi.org/10.1016/j.jii.2018.01.005

Platformio.org. (2022). *platformio*. https://platformio.org/

Severance, C. (2013). Eben Upton: Raspberry Pi. *Computer*, *46*(10), 14–16. https://doi.org/10.1109/MC.2013.349

Severance, C. (2014). Massimo Banzi: Building Arduino. *Computer*, *47*(1), 11–12. https://doi.org/10.1109/MC.2014.19

Skraba, A., Kolozvari, A., Kofjac, D., & Stojanović, R. (2015). Wheelchair maneuvering using leap motion controller and cloud based speech control: Prototype realization. *Proceedings - 2015 4th Mediterranean Conference on Embedded Computing, MECO 2015 - Including ECyPS 2015, BioEMIS 2015, BioICT 2015, MECO-Student Challenge 2015*, 391–394. https://doi.org/10.1109/MECO.2015.7181952

Skraba, A., Kološvari, A., Kofjač, D., & Stojanović, R. (2014). Prototype of speech controlled cloud based wheelchair platform for disabled persons. *Proceedings - 2014 3rd Mediterranean Conference on Embedded Computing, MECO 2014 - Including ECyPS 2014*, 162–165.

Skraba, A., Kolozvari, A., Kofjac, D., Stojanovic, R., Semenkin, E., & Stanovov, V. (2019). Development of Cyber-Physical Speech-Controlled Wheelchair for Disabled Persons. *Proceedings - Euromicro Conference on Digital System Design, DSD 2019*. https://doi.org/10.1109/DSD.2019.00072

Škraba, A., Kolozvari, A., Kofjac, D., Stojanovic, R., Semenkin, E., & Stanovov, V. (2019). Prototype of Group Heart Rate Monitoring with ESP32. *2019 8th Mediterranean Conference on Embedded Computing, MECO 2019 - Proceedings*. https://doi.org/10.1109/MECO.2019.8760150

Škraba, A., Kološvari, A., Kofjač, D., Stojanović, R., Stanovov, V., & Semenkin, E. (2017). Prototype of group heart rate monitoring with NODEMCU ESP8266. *2017 6th Mediterranean Conference on Embedded Computing, MECO 2017 - Including ECYPS 2017, Proceedings*. https://doi.org/10.1109/MECO.2017.7977151

Škraba, A., Kološvari, A., Kofjač, D., Stojanović, R., Stanovov, V., & Semenkin, E. (2016). Streaming pulse data to the cloud with bluetooth le or NODEMCU ESP8266. *2016 5th Mediterranean Conference on Embedded Computing, MECO 2016 - Including ECyPS 2016, BIOENG.MED 2016, MECO: Student Challenge 2016*, 428–431. https://doi.org/10.1109/MECO.2016.7525798

Škraba, A., Kološvari, A., Kofjač, D., Vavtar, B., Stojanović, R., Stanovov, V., & Semenkin, E. (2018). Development of educational cyber-physical Internet of Things platform: Study of the PID controller. *2018 7th Mediterranean Conference on Embedded Computing, MECO 2018 - Including*

ECYPS 2018, Proceedings. https://doi.org/10.1109/MECO.2018.8405982

Škraba, A., Stanovov, V., & Semenkin, E. (2019). Modelling of DC motor and educational application in Cyberphysical systems. *IOP Conference Series: Materials Science and Engineering*, *537*(4). https://doi.org/10.1088/1757-899X/537/4/042008

Skraba, A., Stanovov, V., Semenkin, E., Kolozvari, A., Stojanovic, R., & Kofjac, D. (2016). Putting cloud 9 IDE on the wheels for programming Cyber-Physical / Internet of Things Platforms: Providing educational prototypes. *ICINCO 2016 - Proceedings of the 13th International Conference on Informatics in Control, Automation and Robotics*, *2*.

Škraba, A., Stojanović, R., Zupan, A., Kološvari, A., & Kofjač, D. (2015). Speech-controlled cloud-based wheelchair platform for disabled persons. *Microprocessors and Microsystems*, *39*(8), 819–828. https://doi.org/10.1016/j.micpro.2015.10.004

Soare, S., Predusca, G., & Diaconu, E. (2021). Applications Models Using Beaglebone FPGA. *The Scientific Bulletin of Electrical Engineering Faculty*, *21*(1). https://doi.org/10.2478/sbeef-2021-0003

Stojanović, R., Škraba, A., & Lutovac, B. (2020). A Headset Like Wearable Device to Track COVID-19 Symptoms. *2020 9th Mediterranean Conference on Embedded Computing (MECO)*, 1–4. https://doi.org/10.1109/MECO49872.2020.9134211