

Realno-časovni sistem za spremljanje rasti rastlin

Timotej Petrovčič

Univerza v Ljubljani, Fakulteta za elektrotehniko, Tržaška cesta 25, Ljubljana

Abstract

The goal of this project was to create a custom solution for monitoring plant growth using an analog soil moisture sensor, DHT22 air temperature and humidity sensor, and a relay module for controlling the water pump. The system is designed to run on a Raspberry Pi, with a Python async API and is deployed using Docker. The Raspberry Pi is connected to the server via Tailscale VPN, and on the server side, there is a middleware Golang API that forwards all data to a PostgreSQL database. Additionally, there is a NextJS frontend server written in React and Typescript that serves a real-time website of the sensors that are connected to the Raspberry Pi. Both the Raspberry Pi API and Server are forwarded through Nginx reverse proxy for additional customization and security.

1 Uvod

Cilj tega projekta je bila izdelava prilagojene rešitve za spremljanje rasti rastlin s pomočjo analognega senzorja vlage tal, senzorja za temperaturo in vlago zraka DHT22 ter relejni modula za nadzor vodne črpalke. Sistem deluje na vgradnem računalniku Raspberry Pi, z uporabo asinhronnega vmesnika za namensko programiranje v Pythonu, katerega zaganjamo v kontejnerski arhitekturi Docker. Raspberry Pi je povezan s strežnikom preko virtualnega zasebnega omrežja Tailscale. Strežniška stran je sestavljena iz posredniškega vmesnika za namensko programiranje v Golangu, ki podatke posreduje v podatkovno bazo PostgreSQL. Poleg tega je na strežniški strani postavljen spletni strežnik NextJS, napisan v Reactu in Typescriptu, ki na spletni strani v realnem času prikazuje podatke senzorjev. Raspberry Pi in strežnik sta preusmerjena preko Nginx posredniškega strežnika za dodatno prilagajanje delovanja in varnost storitev.

2 Strojna oprema

Strojna oprema, ki smo jo uporabili je sledeča:

- Vgradni računalnik Raspberry Pi 4
- Digitalni senzor temperature in vlažnosti zraka DHT22
- Analogni kapacitivni senzor vlažnosti tal
- Analogno digitalni pretvornik MCP3008 (10-bitni)

- Relejni modul za nadzor vodne črpalke

Vse komponente smo na vgradni računalnik povezali preko izhodne preizkusne plošče. Vgradni računalnik nima analognih vhodov zato je bilo potrebno dodati analogno-digitalni pretvornik, ki je kompatibilen z izbrano knjižnico za branje podatkov senzorjev.

3 Programska oprema

Programsko opremo najlažje razdelimo na različne storitve, glede na to kje v postavljenem omrežju se nahajajo. Prav tako je zaradi uporabe kontejnerske arhitekture leta prenosljiva med različnimi vgradnimi sistemi, ki podpirajo operacijski sistem Linux. Edina zahteva je sprejemba okoljskih spremenljivk v direktoriju projekta.

3.1 Omrežna struktura projekta

V osnovi je projekt razdeljen na dva ločena lokalna omrežja povezana z virtualnim zasebnim omrežjem. Lokalna omrežja sta:

- Omrežje vgradnega računalnika Raspberry Pi
- Omrežje strežnika

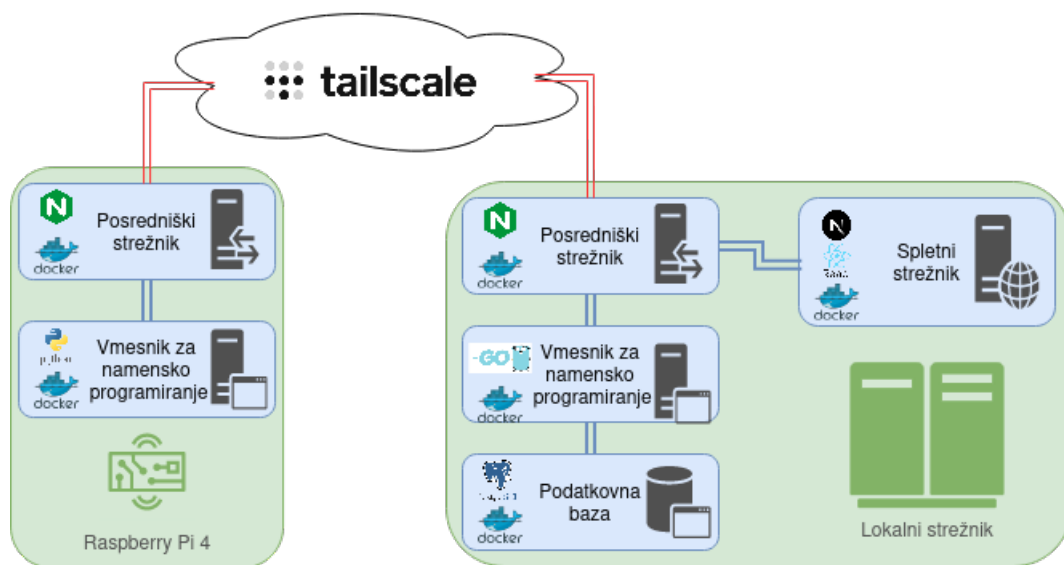
Znotraj samih lokalnih omrežij so zasnovana podomrežja, ki med seboj ločujejo posamezne storitve. Vsako omrežje vsebuje več kontejnerjev (Docker), ki so med seboj povezani preko pred-konfiguriranih omrežnih mostov.

Omrežje vgradnega računalnika Raspberry Pi je sestavljeno iz dveh kontejnerjev:

- Kontejner vmesnika za namensko programiranje v Pythonu
- Kontejner posredniškega strežnika Nginx

Omrežje lokalnega strežnika je sestavljeno iz štirih kontejnerjev:

- Kontejner vmesnika za namensko programiranje v Golangu
- Kontejner podatkovne baze Postgres
- Kontejner spletnega strežnika NextJS
- Kontejner posredniškega strežnika Nginx



Slika 1: Omrežna struktura projekta

3.2 Vmesnik za namensko programiranje v Pythonu

Vmesnik za namensko programiranje skrbi za pridobivanje podatkov iz senzorjev ob prejeti primerni poizvedbi preko protokola HTTP. Vsi podatki so zapisani v standardizirani obliki za komunikacijo formata JSON. Zaradi uporabe programskega jezika Python je potrebno zakleniti verzije knjižnic, ki jih uvozimo. To opravimo s pomočjo orodja Poetry, ki hrani arhiv različnih verzij knjižnic v Pythonu.

V splošnem je vmesnik razdeljen na dva nivoja abstrakcije in sicer:

- Nivo spletnega dostopa
- Nivo podatkovnega dostopa

Omenjena delitev je ključna, saj omogoča ločeno obravnavo napak zaradi napačnih poizvedb ali napak zaradi pridobivanja podatkov s strani senzorjev.

3.2.1 Nivo spletnega dostopa

Glavne komponente oz deli nivoja spletnega dostopa so uvoz okoljskih spremenljivk za določitev ddelovanja strežnik; razreda za nadzor strežnika ter posameznih metod znotraj omenjenega razreda, ki skrbijo za posredovanje pravih podatkov nivoju podatkovnega dostopa. Nivo spletnega dostopa tudi določi katere poizvedbe so mogoče na vmesnik in le-te so:

- **/healthz** - Vedno prisotna poizvedba s katero preverimo delovanje vmesnika
- **/singleAirTemperatureHumidityReading** - Poizvedba za pridobitev ene meritve senzorja DHT22
- **/bulkAirTemperatureHumidityReading** - Poizvedba za pridobitev večih meritev senzorja DHT22

(Število meritev določimo v poizvedbi preko spremenljivke *numOfReadings* v JSON formatu)

- **/singleSoilMoistureReading** - Poizvedba za pridobitev ene meritve senzorja vlažnosti tal
- **/bulkSoilMoistureReading** - Poizvedba za pridobitev večih meritev senzorja vlažnosti tal (Število meritev določimo preko spremenljivke *numOfReadings* v JSON formatu)
- **/setRelayON** - Poizvedba za vklop rele modula
- **/setRelayOFF** - Poizvedba za izklop rele modula

Vse poizvedbe so napisane v podobnem formatu kode, edina razlika je da je pri poizvedbah za več meritev senzorjev potrebno preveriti ali je spremenljivka *numOfReadings* nastavljena.

```
# GET one air temperature reading
@routes.get("/singleAirTemperatureHumidityReading")
async def single_air_temperature_humidity_reading(request):
    log.info("GET single air temperature/humidity reading ##")

    try:
        # Create DAL object using context manager
        with RPI_dal() as dataAbstractionLayer:
            json_response = await dataAbstractionLayer.get_air_temperature_humidity()

        # Check if DAL returned False
        if json_response == False:
            raise web.HTTPInternalServerError("!! GET single air temperature/humidity reading error: Couldn't read the temperature !!")
        else:
            return web.json_response({"status": 200, "message": json_response})
    except:
        log.exception("!! GET single air temperature/humidity reading error: Couldn't read the temperature !!")
        raise web.HTTPInternalServerError("!! GET single air temperature/humidity reading error: Couldn't read the temperature !!")
```

Slika 2: Primer poizvedbe za eno meritev temperature na nivoju spletnega dostopa

3.2.2 Nivo podatkovnega dostopa

Nivo podatkovnega dostopa skrbi za pridobivanje podatkov s strani senzorjev oziroma anstavlja rele modul. Deluje v načinu kontekstnega upravljalca (ang. context manager) s pomočjo posebnih funkcij `__enter__` in `__exit__`, ki

se izvršiti ob vsakem vhodu in izhodu v novo instanco razreda podatkovnega dostopa. Funkciji skrbiti za inicializacijo oziroma de-inicializacijo senzorjev na pravih vhodih ter postavitev primernih struktur za digitalno oziroma SPI komunikacijo z senzorji/relejem. Ostale funkcije pa so korespondenčne že omenjenim funkcijam v nivoju spletnega dostopa. Funkcije branja senzorjev so narejene tako, da dokler preko komunikacijskega vmesnika ne pridobijo verodostojnih meritev ne prenehajo z poizkušanjem. Ob vsaki poizvedbi se podatkom doda tudi časovna oznaka. Podatki se nato pretvorijo v format JSON ter pošljejo kot odziv na dano poizvedbo.

```
def __enter__(self):
    # Initialize logger
    self.logger = logging.getLogger(__name__)
    self.logger.info("RPI_dal enter")

    # Initialize DHT22 sensor
    self.DHT_PIN = os.getenv('DHT_PIN')
    if not self.DHT_PIN:
        self.logger.warning("DHT_PIN environment variable not set!")
    else:
        self.DHT_SENSOR = adafruit_dht.DHT22(getattr(board, "D" + str(self.DHT_PIN)), use_pulseio=False)
```

Slika 3: Primer inicializacije senzorja DHT22 na nivoju podatkovnega dostopa

```
# GET one temperature/humidity measurement
async def get_air_temperature_humidity(self):
    # Read temperature from sensor and get timestamp
    try:
        # Log start of readings
        self.logger.info("## GET bulk air temperature/humidity reading started ##")

        # Set reading boolean to False
        reading = False
        # Read DHT22 sensor until successful
        while not reading:
            try:
                # Read temperature from sensor and get timestamp
                humidity, temperature = self.DHT_SENSOR.humidity, self.DHT_SENSOR.temperature
                timestamp = datetime.datetime.now().strftime("Y-%m-%d %H:%M:%S.%f")[:-3]
                reading = True
            except RuntimeError as error:
                # Reading failed, retry
                self.logger.warning("!! GET single air temperature/humidity reading error: Couldn't read DHT22 temperature/humidity !!")
                self.logger.warning("!! GET single air temperature/humidity reading error: Retrying in 2 seconds !!")
                time.sleep(2.0)
                continue
            except Exception as error:
                # Reading failed, retry
                self.logger.exception("!! GET single air temperature/humidity reading error: Fatal read DHT22 temperature/humidity !!")
                self.DHT_SENSOR.exit()
                return False

        # Log timestamp and temperature, humidity
        self.logger.info("## Timestamp: " + str(timestamp) + " ##")
        self.logger.info("## Air Temperature: " + str(temperature) + " ##")
        self.logger.info("## Air Humidity: " + str(humidity) + " ##")

        return ("timestamp": timestamp, "air-temperature": temperature, "air-humidity": humidity)

    except:
        self.logger.exception("!! GET single air temperature/humidity reading error: Couldn't read DHT22 temperature/humidity !!")
        return False
```

Slika 4: Primer meritve senzorja DHT22 na nivoju podatkovnega dostopa

3.3 Vmesnik za namensko programiranje v Golangu

Vmesnik za namensko programiranje v Golangu je posrednik med spletnim strežnikom in vmesnikom na vgrajenem računalniku. Skrbi za hitro prepošiljanje prejetih poizvedb ter vse podatke zapiše na podatkovno bazo. Predvsem je pri zapisu na podatkovno bazo pomembna konsistentnost izročitve podatkov ter v primeru napake povrnitev stanja baze na prejšnje. Omenjeno skrbi za ohranjanje stabilnosti podatkovne baze brez degradiranih podatkov ter posledično lažjo migracijo, ko je to potrebno.

3.4 Podatkovna baza Postgres

Podatkovna baza Postgres je relacijska, kar omogoča medsebojne povezave med različnimi tipi podatkov. V našem primeru teh povezav nismo potrebovali, saj smo podatke zapisovali v štiri ločene tabele, ki so vsebovale prebrane vrednosti senzorjev (temperatura zraka, vlažnost zraka,

```
// Context manager for DB transactions
func executeInTransaction(db *gorm.DB, f func(tx *gorm.DB) error) error {
    // Start a transaction
    tx := db.Begin()
    // Rollback the transaction if there is an error
    defer func() {
        if r := recover(); r != nil {
            tx.Rollback()
        }
    }()

    // Execute the function
    err := f(tx)
    // Rollback the transaction if there is an error
    if err != nil {
        tx.Rollback()
        return err
    }

    // Commit the transaction
    return tx.Commit().Error
}
```

Slika 5: Primer funkcije za obravnavo zapisov na podatkovno bazo

vlažnost zemlje) oz. releja (stanje releja) ter časovno oznako. Glede na to da nismo potrebovali relacij se omejena baza lahko zelo hitro zamenja z manjšo prilagodljivo kodo posredniškega vmesnika v Golangu. Zaradi samih strojnih omejitev hitrosti branja senzorjev omenjeno ne vpliva na sam sistem, če pa bi uporabili hitrejše senzorje bi bil smiseln prehod na ne-relacijsko podatkovno bazo, katere glavna prednost je hitrost zapisovanja.

```
root@3b8d0f5893c8:/# psql postgres://admin:RaspbianP1@db:5432/RPIdatabase
psql (15.1 (Debian 15.1-1.pgdg10+1))
Type "help" for help.

RPIdatabase=# \dt+

```

Schema	Name	Type	Owner	Persistence	Access method	Size	Description
public	air_humidities	table	admin	permanent	heap	192 KB	
public	air_temperatures	table	admin	permanent	heap	192 KB	
public	relay_states	table	admin	permanent	heap	96 KB	
public	soil_moistures	table	admin	permanent	heap	208 KB	

```
(4 rows)
```

Slika 6: Primer povezave preko kontejnerja ter tabele podatkovne baze

3.5 Spletni strežnik NextJS

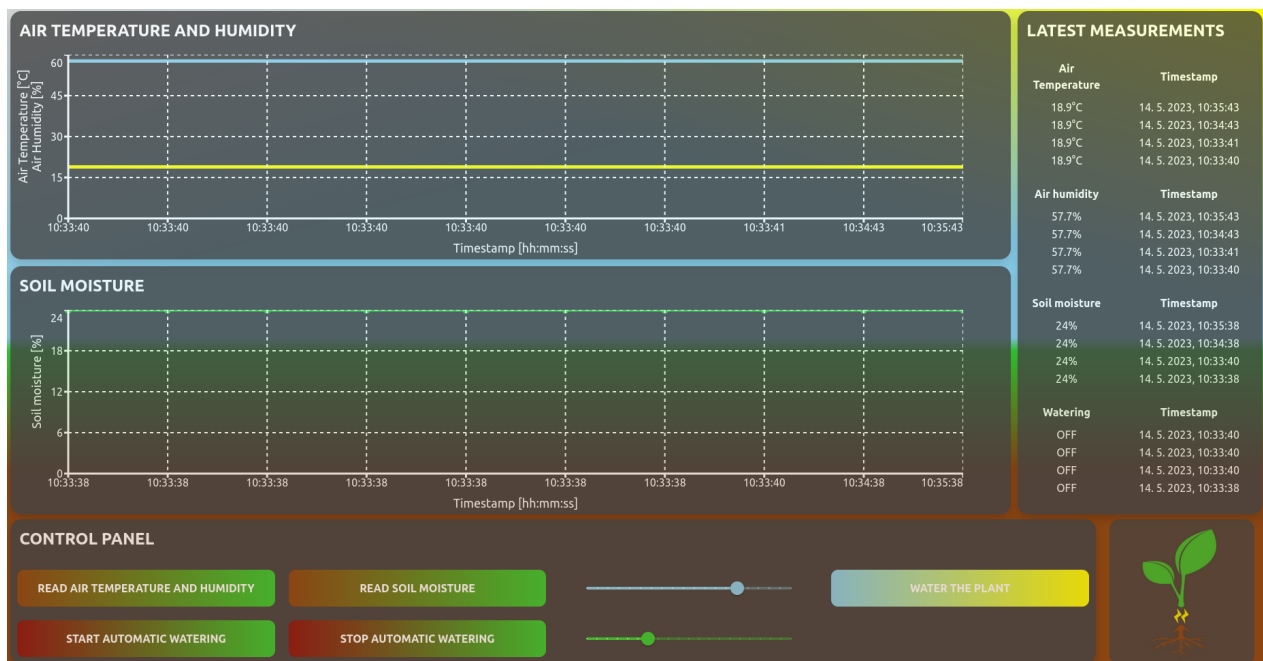
Spletni strežnik NextJS je hitrejša različica znanega NodeJSa in omogoča hitro postavitev spletnih storitev za potrebe produkcijskih okolij. Omogoča nabor jezikov v katerih je mogoče napisati spletno stran ter jo nato gostiti na strežniku. Odločili smo se za kombinacijo ogrodja React ter pisanja v jeziku Typescript, ki je kombinacija jezikov Javascript, HTML in CSS. V splošnem je razvoj potekal po komponentah, kar omogoča ogroddje React. Vsako komponento smo oblikovali, napisali primerno kodo ter na koncu vstavili v koren spletne strani ali pa eno v drugo ter s tem ustvarili modularno ogroddje za sestavo spletne strani. Algoritemski del kode je sestavljen iz večih razredov, ki so medsebojno odvisni. V jedru je osnovni razred za izvedbo poizvedb na vmesnik v Golangu, ki razširja razred za dinamično posodabljanje tabele s podatki, sledijo si razred za posodabljanje grafov, razred za avtomatsko zalivanje, razred za ročno zalivanje ter razred za branje senzorjev. Posodabljanje podatkov v tabelah teče v realnem času, grafi pa naredijo poizvedbo vsako minuto ter posodobijo vrednost. Vsi razredi vsebujejo asinhrono funkcije za sočasno izvajanje različnih storitev, ki jih omogoča sama spletna stran.

```

// Soil moisture graph component
class SoilMoistureGraphComponent extends Component {
  // Interval time in milliseconds
  intervalTime: number = 60000;
  // Define the api request class instance
  apiRequestClassInstance = new apiRequestClass();
  // Define the state
  state = {
    chartData: Array<{ soilMoisture: number, date: string }>()
  };
  // Fetch the data and set the state
  fetchData = async () => {
    // Fetch the latest data point
    const [soilMoisture, timestamp] = await this.apiRequestClassInstance.getSoilMoisture();
    // Create the new chart data
    const newChartData = [...this.state.chartData];
    // Split the timestamp into date and time and format it
    const formattedTimestamp = new Date(timestamp).toLocaleString();
    const [datePart, timePart] = formattedTimestamp.split(", ");
    // If the chart data array is empty, add the first data point
    if (newChartData.length === 0) {
      newChartData.push({ soilMoisture: soilMoisture, date: timePart });
    } else {
      // Remove the first data point and add the new data point at the end
      newChartData.shift();
      newChartData.push({ soilMoisture: soilMoisture, date: timePart.replace("Z", "") });
    }
    // Set the state
    this.setState({ chartData: newChartData });
  };
  // Fetch the data initially and start the interval
  componentDidMount() {
    // Fetch the initial 10 data points
    this.apiRequestClassInstance.getBulkSoilMoisture(10).then(requestArray => {
      // Create the chart data array and format the timestamp
      const soilMoistureArray = requestArray[0];
      const timestampArray = requestArray[1];
      const chartData = [];
      for (let i = 0; i < soilMoistureArray.length; i++) {
        const formattedTimestamp = new Date(timestampArray[i]).toLocaleString();
        const [datePart, timePart] = formattedTimestamp.split(", ");
        chartData.push({ soilMoisture: soilMoistureArray[i], date: timePart });
      }
      // Add the initial 10 data points to the chart data
      this.setState({ chartData });
      // Fetch the latest data point and start the interval
      this.fetchData();
      setInterval(this.fetchData, this.intervalTime);
    });
  }
  // Render the chart
  render() {

```

Slika 7: Primer razredne komponente grafa za vlažnost zemlje



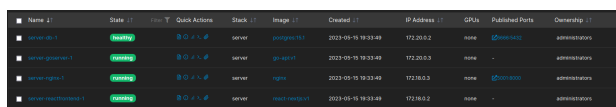
Slika 8: Nadzorna plošča projekta

3.6 Posredniški strežnik Nginx

V samem omrežju imamo dva posredniška strežnika: eden na Vgradnem sistemu ter drugi na strežniku. Vsak skrbi za dostop do točno določene poti. Tako naprimer na vmesnik v Pythonu smemo dostopati le preko poti `/rpi-api/v1/` podobno za posredniški vmesnik v Golangu je potrebno dostopati po poti `/goserver/v1/api` v primeru spletnega strežnika pa je celotna pot odprta, da lahko uporabnik dostopa do vseh domen na spletnem strežniku. Posredniški strežniki se v večini primerov uporabljajo za namen varnosti ter hitrega usmerjanja podatkovnih tokov v omrežju.

3.7 Postavitev storitve in Docker

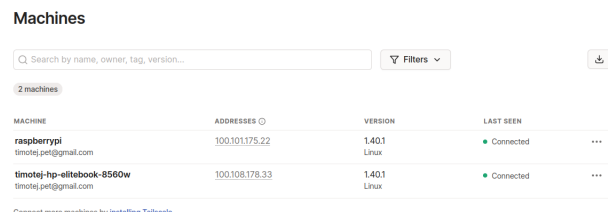
Za postavitev storitve smo izbrali orodje Docker ter Docker-compose, ki omogočata prenosljivost kode na praktično katerikoli računalnik. V našem primeru to za primer vgradnega sistema ni popolnoma res, saj še vedno zahtevamo neko osnovno strojno podporo digitalnih vhodov in izhodov ter SPI vmesnika in možnost uporabe operacijskega sistema Linux. Izven omenjenih strojnih in programske omejitve pa je sistem prenosljiv na praktično katerikoli vgradni sistem z manjšim prilagajanjem okoljskih spremenljivk.



Slika 9: Sklad kontejnerjev ob zagonu strežnika

3.8 Virtualno zasebno omrežje Tailscale

Virtualno zasebno omrežje ponudnika Tailscale smo v našem primeru uporabili predvsem zaradi enostavnosti uporabe in postavitve. V splošnem Tailscale deluje tako, da vzopstavi ločene prenosne kanale do različnih naprav na katerih je nameščen in vse naprave, ki so dodeljene in odobrene s strani uporabnika postavi v svoje omrežje. Za primere demonstracije in domače uporabe je omenjeni pristop primeren za uporabo v produkcijskih okoljih pa bi bilo potrebno enega izmed strežnikov izpostaviti omrežju preko javnega IP naslova, kamor bi nato druga naprava pošiljala poizvedbe.



Slika 10: Primer povezanih naprav v virtualno zasebno omrežje

4 Povzetek

Projekt je bil uspešno zaključen z uporabo relativno modernih programskih paketov in orodij. Sama programska

zasnova bi ob boljši strojni opremi omogočala tudi precej bolj tesne časovne tolerance brez večjih popravkov. Glede zanesljivosti izbranih senzorjev se je izkazalo, da je točnost senzorja DHT22 znotraj zapisanih karakteristik na podatkovnem list, medtem ko kapacitivni senzor za vlago tal ni najbolj zanesljiv, ker ni kalibriran. Ob kalibraciji bi bilo potrebno delno popraviti kodo v nivoju podatkovnega dostopa, a je sama količina popravkov majhna. Prav tako bi bilo potrebno izvesti praktične preizkuse ali so podane metode za avtomatsko zalivanje glede na določeno mejo vlažnosti tal primerne ter jih v primeru neustreznosti prilagoditi. Samemu prikazu na nadzorni plošči bi bilo smiselno dodati določanje časovnih razmakov med meritvami prikazanih na grafih, za omejeno funkcionalnost je v trenutni izvedbi zmanjkalo časa, nadaljnje pa je implementacija pogojena z modifikacijo vmesnika v Golangu (za dostop do starejših podatkov na podatkovni bazi) ter spletnega strežnika (za prikaz podatkov na grafih). Sam projekt je bil razvit odprtokodno in je dostopen na javnem repozitoriju: Automatic-virtual-measurement-systems

Literatura

- [1] Timotej Petrovčič, javno dostopen repozitorij: <https://github.com/Timotej979/Automatic-virtual-measurement-systems>
- [2] Adafruit, MCP3008 - 8-Channel 10-Bit ADC With SPI Interface in Python & CircuitPython: <https://learn.adafruit.com/mcp3008-spi-adc/python-circuitpython>
- [3] Različni avtorji, dokumentacija uporabljenih programskih jezikov, orodij in knjižnic:
 - Python programski jezik (Knjižnice):
 - Nivo spletnega dostopa - *logging, os, sys, asyncio, aiohttp*
 - Nivo podatkovnega dostopa (Uporaba CircuitPython knjižnice za dostop do strojne opreme)
 - * *time, datetime, board, busio, digitalio, adafruit_dht, adafruit_mcp3xxx.mcp3008, adafruit_mcp3xxx.analog_in*
 - Golang programski jezik (Knjižnice):
 - *fmt, io/ioutil, log, os, time, bytes, net/http*
 - *github.com/goccy/go-json, github.com/shopspring/decimal, gorm.io/driver/postgres, gorm.io/gorm, github.com/gofiber/fiber/v2*
 - PostgreSQL podatkovna baza (Dostop in postavitve sheme)
 - NextJS spletni strežnik
 - Programski jezik: *Typescript*
 - Označevalna jezika: *HTML5, Tailwind CSS3*
 - Ogrodje: *React*
 - NGINX posredniški strežnik (Konfiguracijske datoteke)
 - Docker in Docker-compose (Postavitev storitev)