

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Wasp mote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Wasp mote i Pycom Tehnička dokumentacija Verzija <1.0>

Studentski tim: Laura Abramović
Filip Đuran
Benjamin Horvat
Domagoj Kolega
Luka Lacković
Josip Lukačević
Borna Majstorović
Ana Mrkonjić
Mihael Rodek

Nastavnik: prof. dr. sc. Mario Kušek

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

Sadržaj

1. Opis razvijenog proizvoda	4
2. Arhitektura sustava	4
3. LoRaWAN	6
4. Narrowband IoT	8
5. Senzori	9
6. Poslužitelj	12
7. Android aplikacija	18
8. iOS aplikacija	20
9. Upute za korištenje	23
10. Literatura	24

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Wasp mote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

Tehnička dokumentacija

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

1. Opis razvijenog proizvoda

U sklopu projekta razvijen je sustav Interneta Stvari za mjerenje nekoliko različitih vrijednosti važnih za poljoprivredu i prikaz tih vrijednosti na Android i iOS aplikacijama. Razvijeni proizvod sastoji se od četiri glavna dijela:

- Sklopovski dio koji se sastoji od dva Waspote uređaja i jednog Pycom uređaja sa povezanim senzorima
- Poslužiteljski dio
- Android aplikacija
- iOS aplikacija

Podaci koji se mjere i prikazuju u aplikacijama uključuju osvjetljenje, te temperaturu, vlagu i tlak zraka. Ti podaci šalju se s uređaja na poslužitelj koristeći LoRaWAN i Narrowband IoT komunikacijske protokole, dok aplikacije te podatke dohvaćaju HTTP protokolom koristeći REST standard.

Aplikacije služe za stalno praćenje mjerenja i njihov prikaz na linijskim grafovima. Korisniku se daje opcija za dodavanje uređaja u aplikaciju po volji, ovisno o tome koja mjerenja ga zanimaju.

Razvoj mobilnih aplikacija temeljio se na istim funkcionalnostima, ali ipak su razvijene u različitim timovima stoga postoje određene razlike u implementaciji i složenosti. Detalji o implementaciji aplikacija mogu se pronaći u Android i iOS cijelinama ovog dokumenta.

2. Arhitektura sustava

Arhitektura sustava može se podijeliti u nekoliko podsustava:

- Mobilne aplikacije
- Platforma interneta stvari
- Pametna okolina

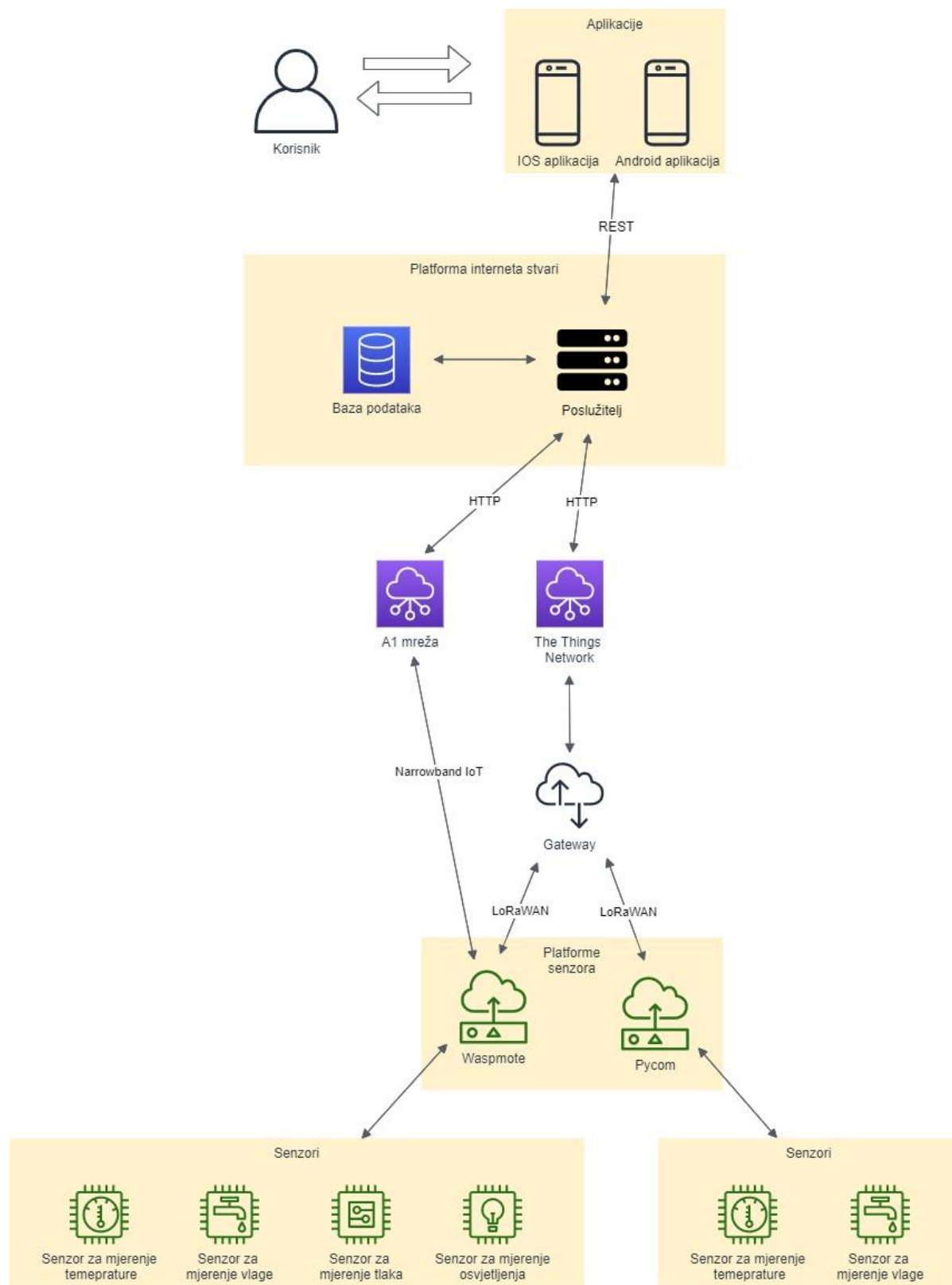
Mobilna aplikacija (*eng. mobile application*) programska je potpora za mobilne uređaje koja korisniku omogućuje pregled sadržaja za koji je ta aplikacija namijenjena. Mobilne aplikacije razvijene su okruženju Android Studio za uređaje bazirane na operacijskom sustavu Android, te u okruženju Xcode za uređaje bazirane na operacijskom sustavu iOS.

Platforma interneta stvari (*eng. IoT platform*) sadrži poslužitelj (*eng. server*) koji je osnova rada aplikacije. Poslužiteljski dio razvijen je u Java Springu. Klijent koristi aplikaciju za obrađivanje željenih zahtjeva. Aplikacija obrađuje zahtjev te ovisno o njemu pristupa bazi podataka nakon čega preko poslužitelja vraća odgovor klijentu. Komunikacija između klijenta i poslužitelja temelji se na REST standardu i koristi se HTTP protokol.

Pametna okolina sastoji se od senzora, sklopovskih platformi i poveznika. Poveznik (*engl. gateway*) povezuje uređaje i poslužiteljski dio aplikacije. Senzori su uređaji koji mjere neku fizikalnu veličinu te ju pretvaraju u signal pogodan za daljnju obradu. Senzori se spajaju na sklopovske platforme koje su u našem slučaju Waspote i PyCom platforme. Sklopovske platforme komuniciraju s poveznikom putem nekoliko komunikacijskih protokola, a to su: LoRaWAN za jedan Waspote i jedan Pycom uređaj, te Narrowband IoT za drugi Waspote uređaj. Podaci koje senzori mjere u ovom projektu su osvjetljenje, te temperatura, vlaga i tlak zraka.

Na slici 2.1 prikazana je ova arhitektura sustava.

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>



Slika 2.1 – Arhitektura sustava

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

3. LoRaWAN

U ovom projektu, jedan Waspote uređaj i jedan Pycom uređaj komunicirali su s poslužiteljem koristeći LoRaWAN komunikacijski protokol.

LoRaWAN je protokol za upravljanje pristupom mediju (MAC) za mreže širokog područja (WAN). Osmišljen je kako bi uređajima s malim napajanjem omogućio komunikaciju s internetskim aplikacijama putem bežičnih veza. Djeluje u nelicenciranom radiofrekvencijskom spektru, a koristi niže frekvencije s većim dometom.

Poveznik (eng. *gateway*) je posrednik između uređaja i poslužitelja. Poslužitelj komunicira s poveznikom putem interneta i govori mu kako ostvariti komunikaciju s uređajem. Uređaji koriste LoRaWAN kako bi se spojili na poveznik, koji onda njihove poruke proslijeđuje na The Things Network.

Slanje podataka s uređaja na aplikaciju (eng. *uplink*)

Sadržaj (eng. *payload*) koji se šalje trebao bi biti što manji. Stoga se podaci ne bi trebali slati kao JSON ili običan tekst, već kodirati kao binarni podaci. Primjer kodiranja podataka na Waspote uređju prikazan je na slici 3.1, dok je kodiranje podataka na Pycom uređaju prikazano na slici 3.2.

```

payload[0] = temp >> 24;
payload[1] = temp >> 16;
payload[2] = temp >> 8;
payload[3] = temp;

payload[4] = vlaga >> 24;
payload[5] = vlaga >> 16;
payload[6] = vlaga >> 8;
payload[7] = vlaga;

payload[8] = tlak >> 24;
payload[9] = tlak >> 16;
payload[10] = tlak >> 8;
payload[11] = tlak;

```

Slika 3.1 – Kodiranje podataka na Waspote uređaju

```

payload = struct.pack(">ff", temperature, humidity)
try:
    s.send(payload)
except:
    pass

```

Slika 3.2 – Kodiranje podataka na Pycom uređaju

Kao što se može vidjeti na slici 3.2, kod Pycom uređaja koristi se knjižnica `struct.py` koja omogućava pakiranje više vrijednosti u bajtove, koristeći zadani format „>ff“ (dvije float vrijednosti u big-endian poretaku).

Slanje podataka s aplikacije na uređaj (eng. *downlink*)

Budući da poveznik ne može primiti prijenose uređaja dok odašilje poruke, u vrijeme emitiranja uređaji postaju beznačajni. Kako bi dostupnost poveznika bila što veća, brzina prijenosa bi trebala biti što učinkovitija, a ako je moguće, slanje poruka prema uređaju bi trebalo izbjegavati.

LoRaWAN definira tri vrste uređaja, koji se razvrstavaju u klase A, B i C. Svi LoRaWAN uređaji moraju

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

implementirati klasu A, dok su klase B i C proširenja specifikacije uređaja klase A. Uređaji klase A podržavaju dvosmjernu komunikaciju između uređaja i poveznika. Poruke s uređaja na poslužitelj mogu biti poslane u bilo kojem trenutku. Uređaj nakon toga otvara dva prozora za primanje poruke od poslužitelja. Poslužitelj može odgovoriti u prvom ili drugom prozoru, ali ne bi trebao koristiti oba. Uređaji klase B proširuju klasu A dodavanjem prozora za zakazano primanje poruka poslužitelja. Uređaji klase C proširuju klasu A držeći prozore za prijam otvorenima. To omogućuje komunikaciju s malim kašnjenjem, ali i mnogo većom potrošnjom energije od klase A. U ovom projektu korišten je LoRaWAN 868 modul koji implementira klasu A.

The Things Network razvija implementaciju mrežnog poslužitelja LoRaWAN. Aplikacijama i uređajima upravlja se putem The Things Network konzole. Konzola je web aplikacija putem koja se može koristiti za registraciju aplikacija, uređaja ili poveznika, te nadgledanje mrežnog prometa. Na konzoli se najprije registrira uređaj, te se generiraju podaci koje je potrebno unijeti u kod kojim se upravlja uređajem. Primjer tog procesa prikazan je na slici 3.3.

DEVICE OVERVIEW

Application ID

lorawan_application_fer

Device ID

waspote_device_iot

Activation Method

ABP

Device EUI

<> 00 F2 90 48 B7 91 47 23

Application EUI

<> 70 B3 D5 7E D0 03 A3 47

Device Address

<> 26 01 17 A2

Network Session Key

<>

App Session Key

<>

```

char DEVICE_EUI[] = "00F29048B7914723";
char DEVICE_ADDR[] = "260117A2";
char NWK_SESSION_KEY[] = "97C11C3F0B95BDA7AB3068A48A52FA43";
char APP_SESSION_KEY[] = "1058EEBFFD791F1C69F10071B312CCB4";

uint8_t PORT = 1;
uint8_t error;

float floatTemp;
float floatVlaga;
float floatTlak;

uint8_t payload[12];
uint16_t payload_size = sizeof(payload);

void setup()
{
  USB.ON();

  LoRaWAN.ON(socket);
  LoRaWAN.setDeviceEUI(DEVICE_EUI);
  LoRaWAN.setDeviceAddr(DEVICE_ADDR);
  LoRaWAN.setNwkSessionKey(NWK_SESSION_KEY);
  LoRaWAN.setAppSessionKey(APP_SESSION_KEY);
  LoRaWAN.saveConfig();

  Events.ON();
}

```

Slika 3.3 – Spajanje na LoRaWAN koristeći podatke s The Things Network stranice

Podaci koje šalje uređaj vidljivi su na konzoli. Budući da se na mrežu šalju kodirani binarni podaci, potrebno je napisati dekodeer podataka kako bi se oni mogli proslijediti na poslužitelj u razumljivom formatu. Na slici 3.4 prikazan je primjer dekodeera.

```

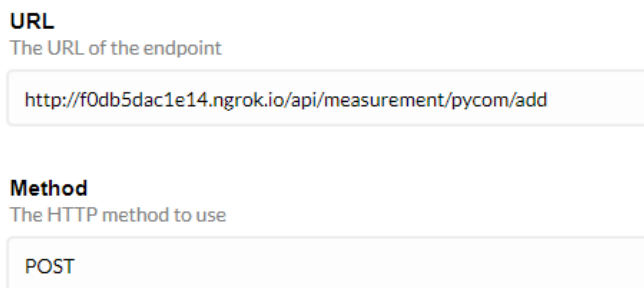
1 function Decoder(bytes, port) {
2   var decoded = {};
3   var tempInt = (bytes[0] & 0x80 ? 0xFFFF<<32 : 0) | bytes[0]<<24 | bytes[1]<<16 | bytes[2]<<8 | bytes[3];
4   decoded.airTemp = tempInt / 100;
5   var humInt = (bytes[4] & 0x80 ? 0xFFFF<<32 : 0) | bytes[4]<<24 | bytes[5]<<16 | bytes[6]<<8 | bytes[7];
6   decoded.airHumidity = humInt / 100;
7   var pressInt = (bytes[8] & 0x80 ? 0xFFFF<<32 : 0) | bytes[8]<<24 | bytes[9]<<16 | bytes[10]<<8 | bytes[11];
8   decoded.airPressure = pressInt / 100;
9   return decoded;
10 }

```

Slika 3.4 – Primjer dekodeera

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Wasp mote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

Kako bi konzola komunicirala s poslužiteljem, potrebno je dodati integraciju putem koje će se prenositi podaci. U ovom projektu za to je korišten HTTP protokol. U konzolu se upiše putanja do poslužiteljske strane i HTTP metoda koja se koristi u komunikaciji. Primjer toga prikazan je na slici 3.5.



URL
The URL of the endpoint

`http://f0db5dac1e14.ngrok.io/api/measurement/pycom/add`

Method
The HTTP method to use

`POST`

Slika 3.5 – HTTP integracija na The Things Network stranici

4. Narrowband IoT

NB-IoT (Narrowband IoT) je tehnologija niskoenergetske širokopojasne mreže (LPWAN) razvijena kako bi omogućila široki spektar IoT uređaja i servisa. Koristi se kada su nam važni mala potrošnja energije, niska cijena i veliki domet. Također, odlikuje je i velika pokrivenost osobito u ruralnim područjima i područjima unutrašnje pokrivenosti, kao što su velike zgrade i skladišta, a to se postiže zbog veće osjetljivosti. Mnogi operateri zbog svega navedenog ulažu u razvoj NB-IoT mreže za Internet stvari.

U sklopu ovog projekta koristi se Libelium-ov BG96 čipset tvrtke Quectel koji podržava NB-IoT i CAT-M standarde. Također, navedeni modul podržava i EGPRS povezivost koja omogućava slanje podataka kada nema pokrivenosti gore navedena dva standarda. Od protokola koje podržava NB-IoT standard tu su HTTP(S), FTP(S), SSL, TCP i UDP. Modul također podržava i GNSS (globalni navigacijski satelitski sustav) što omogućava precizno praćenje uređaja. Kako bi modul ispravno funkcionirao potrebna je SIM kartica operatera koji ima razvijenu NB-IoT mrežu, u ovom slučaju A1.

Dakle, u sklopu ovoga projekta korišten je NB-IoT modul koji preko protokola HTTP i njegovih GET i POST metoda direktno komunicira sa poslužiteljem te mu šalje podatke o mjerenjima sa svojih senzora.

Sve funkcije vezane za komunikaciju preko NB-IoT modula dio su WaspBG96.h biblioteke te ju iz tog razloga nužno uvrstiti u implementaciju kako bismo mogli slati i primati podatke.

Kako bismo se spojili na mrežu i uspostavili konekciju sa poslužiteljem potrebno je poduzeti niz koraka koji konfiguriraju modul za pravilno pristupanje mreži i uspostavljanje konekcije. Pri tome su nam potrebni neki bitni parametri koje moramo predati funkcijama za konfiguriranje a to su: **APN** (ime pristupne točke), **login** i **password** koje dobivamo od mobilnog operatera, u ovom slučaju A1, zatim **band**(pojas), **ime operatera** i **vrsta operatera**.

Nakon svega navedenog modul je spreman za slanje HTTP zahtjeva na poslužitelj što u ovom slučaju radimo preko GET metode koja u queryju prima varijable i njihove vrijednosti. Šalju se podatci o ID-u uređaja i podatci očitani sa senzora (luminosity – osvjetljenje)

Također, treba napomenuti kako je moguće slanje podataka preko Wasp mote-ovog ugrađenog frame-a u heksadekadskom obliku, no međutim to ovdje nije korišteno zbog jednostavnosti tj. lakšeg čitanja podataka na poslužitelju jer bi na taj način na poslužitelju bilo potrebno dodatno prevesti heksadekadske podatke u ASCII

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

znakove te iz takvog zapisa parsirati podatke potrebne za spremanje u bazu.

```
//postavljanje parametara operatora
BG96.set_APN(apn, login, password);
BG96.show_APN();
```

Slika 4.1: Postavljanje operatora

```
//upali NB-Iot modul
error = BG96.ON();
```

Slika 4.2: Paljenje modula

```
//konfiguriraj konekciju
error = BG96.nbiotConnection(apn, band, network_operator, operator_type);
```

Slika 4.3: Konfiguriranje konekcije

```
//uspostavi konekciju
error = BG96.contextActivation(1,5);
```

Slika 4.4: Uspostavljanje konekcije s mrežom

```
//postavi DNS poslužitelj
error = BG96.setDNSServer("8.8.8.8", "8.8.4.4");
```

Slika 4.5: Postavljanje DNS

```
//pošalji HTTP GET na poslužitelj sa podacima u queriju
error = BG96.http(WaspBG96::HTTP_GET, host, port, resource);
```

Slika 4.6 Slanje HTTP requesta na poslužitelj

5. Senzori

5.1 Waspote senzori

Senzori se na Waspote uređaj spajaju pomoću neke od pločica za senzore. U ovom projektu za to se koristi pločica Events Sensor Board v30, na koju se istovremeno može spojiti najviše pet senzora. Senzori su na toj pločici aktivni dok je uređaj u načinu mirovanja, a aktivira se kada senzor generira signal.

Na prvom Waspote uređaju koristi senzor BME280, koji može mjeriti vlagu, tlak i temperaturu. U nastavku su prikazane specifikacije svakog od senzora.

Senzor temperature zraka

- Područje rada: -40° ~ +85° C
- Područje mjerenja: 0 ~ +65° C
- Točnost: ± 1° C
- Vrijeme odziva: 1.65 sekundi

Senzor vlage zraka

- Područje rada: -40° ~ +85° C
- Područje mjerenja: 0 ~ 100% relativne vlage
- Točnost: ± 3% (na temperaturi od 0 ~ +65° C)
- Vrijeme odziva: 1 sekunda

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

Senzor tlaka zraka

- Područje rada: -40° ~ +85° C
- Područje mjerenja: 30 ~ 110 kPa
- Točnost: ± 0.1 kPa (na temperaturi od 0 ~ +65° C)

BME280 zasnovan je na dokazanim principima osjetljivosti. Temperaturni senzor optimiran je za najveću razlučivost, a njegov se izlaz koristi za temperaturnu kompenzaciju senzora tlaka i vlage. Senzor vlage pruža brzo vrijeme odziva za brze primjene konteksta i visoku ukupnu točnost u širokom rasponu temperatura. Senzor tlaka je apsolutni barometarski senzor s velikom preciznošću i razlučivošću te smanjenom razinom smetnji.

Kako bi se pročitale vrijednosti senzora, koriste se funkcije *getTemperature()*, *getHumidity* i *getPressure*, a pozivaju se nad instancom klase *WaspSensorEvent_v30*. Njihovo korištenje prikazano je u kodu na slici 5.1.

```
floatTemp = Events.getTemperature();
floatVlaga = Events.getHumidity();
floatTlak = Events.getPressure();
```

Slika 5.1 – Pozivanje funkcija za vrijednosti senzora na Waspote uređaju

Na drugom Waspote uređaju koristi se senzor osvjetljenja (Luminosity sensor) koji podatke bilježi u mjernoj jedinici lux.

Senzor osvjetljenja:

- Područje očitavanja: 0.1 to 40000 lux
- Spektar: 300 ~ 1100 nm
- Napon: 2.7 ~ 3.6 V
- Potrošnja struje u radu: 0.24 mA
- Potrošnja struje u mirovanju: 0.3 µA
- Temperatura rada: -30 ~ +70 °C

Senzor pretvara intenzitet svjetlost u digitalne signale na izlazu. Uređaj kombinira jednu širokopojasnu foto-diodu (vidljiva i infracrvena) i jednu foto-diodu koja reagira na infracrvenu svjetlost na jednom CMOS integriranom krugu. Postoje dva ADC sklopa koji pretvaraju struju na diodi u digitalni izlaz koji je izražen u lux-ima.

Očitavanje podataka:

```
uint32_t luxes = 0;
Events.ON();
luxes = Events.getLuxes (INDOOR);
```

5.2 Pycom senzori

Kod Pycom sklopovske platforme senzori se nalaze na Pysense V2.0 X ploči. Pysense sadrži više vrsta senzora koje je moguće isprogramirati koristeći već postojeće Pycom knjižnice, ali nama su od važnosti samo senzori za temperaturu i vlagu zraka.

Senzor temperature zraka

- Područje optimalnog rada: -10° ~ +85° C
- Područje mjerenja: -40° ~ +125° C

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

- Točnost: $\pm 1^{\circ}\text{C}$ (na temperaturi -10 do 85°C)
- Vrijeme odziva: 5.1 s

Senzor vlage zraka

- Područje optimalnog rada: $-10^{\circ} \sim +60^{\circ}\text{C}$
- Područje mjerenja: $0 \sim 100\%$ relativne vlage
- Točnost: $\pm 5\%$
- Vrijeme odziva: 18 s

Programiranje senzora na Pycomu moguće je uz pomoć Pymakr dodatka za Visual Studio Code. Koristeći navedeni dodatak, na uređaj se mogu učitati sve potrebne knjižnice za rad senzora, nakon čega se mogu pozivati metode za dohvat pojedinih vrijednosti.

Na slikama 5.2 i 5.3 prikazane su metode koje se pozivaju nad klasom SI7006A20 i služe za dohvat vrijednosti temperature i vlage zraka sa senzora.

```
py = Pysense()
si = SI7006A20(py)
while True:

    temperature = si.temperature()
    print('Temp: ' + str(temperature) + ' Celsius')
    humidity = si.humidity()
    print('Humidity: ' + str(humidity) + '% \n')
```

Slika 5.2 – Pozivanje metoda za dohvat vrijednosti senzora na Pycom uređaju

```
def temperature(self):
    """ obtaining the temperature(degrees Celsius) measured by sensor """
    self.i2c.writeto(SI7006A20_I2C_ADDR, bytearray([0xF3]))
    time.sleep(0.5)
    data = self.i2c.readfrom(SI7006A20_I2C_ADDR, 3)
    #print("CRC Raw temp data: " + hex(data[0]*65536 + data[1]*256 + data[2]))
    data = self._getWord(data[0], data[1])
    temp = ((175.72 * data) / 65536.0) - 46.85
    return temp

def humidity(self):
    """ obtaining the relative humidity(%) measured by sensor """
    self.i2c.writeto(SI7006A20_I2C_ADDR, bytearray([0xF5]))
    time.sleep(0.5)
    data = self.i2c.readfrom(SI7006A20_I2C_ADDR, 2)
    data = self._getWord(data[0], data[1])
    humidity = ((125.0 * data) / 65536.0) - 6.0
    return humidity
```

Slika 5.3 – Metode za dohvat vrijednosti temperature i vlage zraka sa senzora

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

6. Poslužitelj

6.1 REST API

RESTful API za upravljanje resursima koristi HTTP metode POST, GET, PUT, DELETE ili druge. Kada klijent zadraži neki URL, vraća se izravna reprezentacija resursa nad kojim su izvršene navedene metode. Svi podaci koje naš poslužitelj dostavlja klijentu su u JSON format.

Četiri osnovna dijela REST API zahtjeva su:

- URI – URL adresa (krajnja točka)
- HTTP metoda
- Zaglavlja – uključuju tokene za provjeru autentičnosti, definiraju format podatka u dogovoru
- Tijelo – sadrži stvarni dio zahtjeva

Korištene API krajnje točke i njihove HTTP metode:

- /api/measurement/pycom/add (POST) – dodavanje mjerenja s Pycom i Waspote uređaja
- /api/measurement/waspote/add (GET) – dodavanje mjerenja s Waspote uređaja za mjerenje osvjetljenja
- /api/measurement/last (GET) – dohvaćanje zadnjih 10 mjerenja
- /api/measurement/all (GET) – dohvaćanje svih mjerenja
- /api/culture/{id}/devices/add/{deviceId} (POST) – dodavanje uređaja kulturi
- /api/culture/add (POST) – dodavanje nove kulture
- /api/culture/all (GET) – dohvaćanje svih kultura
- /api/culture/{cultureId}/devices/delete/{devId} (DELETE) – uklanjanje uređaja iz kulture
- /api/culture/delete/{id} (DELETE) – brisanje kulture
- /api/boundaries (POST) – dodavanje granice mjerenja
- /api/boundaries/{cultureId} (GET) – dohvaćanje granica mjerenja određene kulture
- /api/auth/register (POST) – slanje podataka za registraciju
- /api/auth/login (POST) – slanje podataka za prijavu
- /api/notifications (GET) – dohvaćanje svih obavijesti
- /api/devices (GET) – dohvaćanje svih uređaja

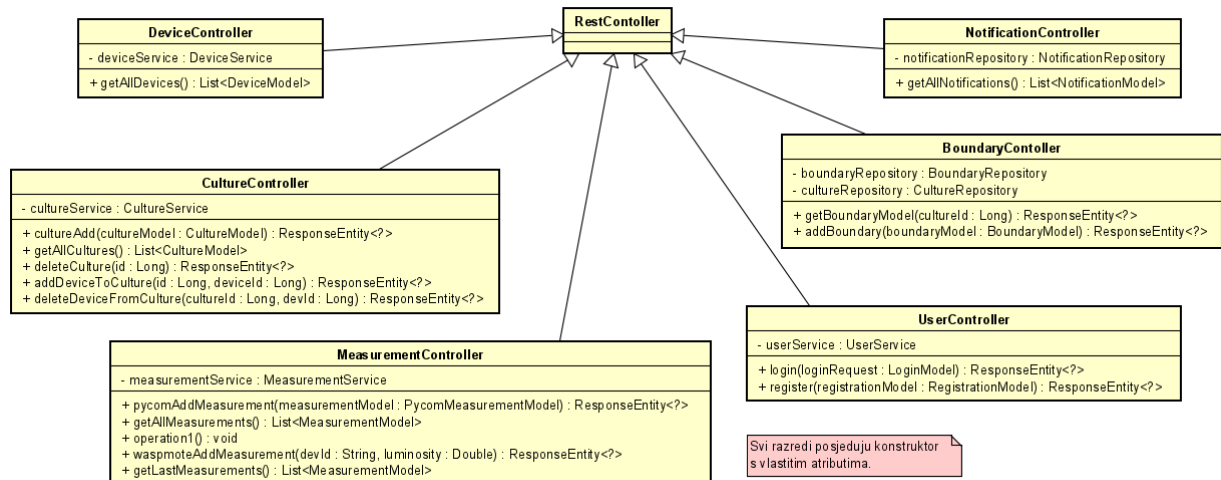
Korištene API krajnje točke kao i resursi koje primaju ili šalju detaljnije su opisani u .yaml datoteci.

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspnote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

6.2 Dijagrami razreda

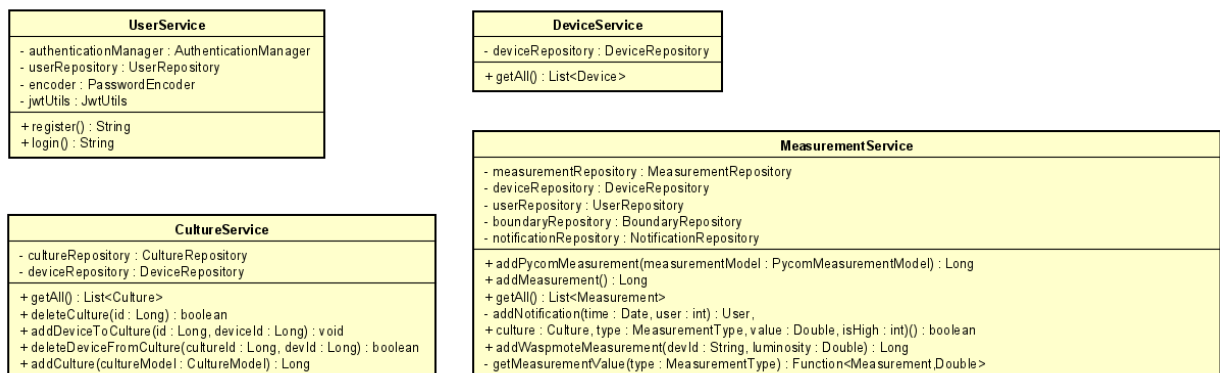
Na slikama 6.1 – 6.5 nalaze se razredi koji pripadaju poslužiteljskom dijelu (REST API).

Na slici 6.1 prikazani su razredi iz dijela *controllers* koji nasljeđuju razred *Rest Controller*. *Rest Controller* je u kodu prikazan kao anotacija i brine se o mapiranju podataka zahtjeva za definiranu metodu obrade zahtjeva. Podaci zahtjeva se u našem slučaju mapiraju iz JSON objekta.



Slika 6.1 - Dio Controller

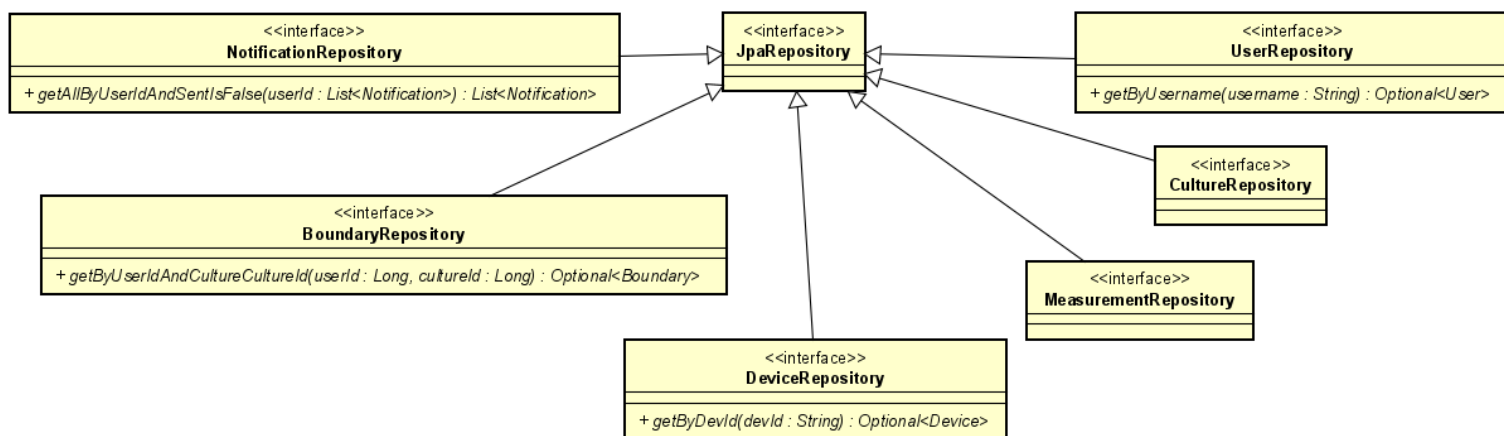
Slika 6.2 prikazuje razrede koji služe kontroleru za manipulaciju podacima u bazi.



Slika 6.2 - Dio Service

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

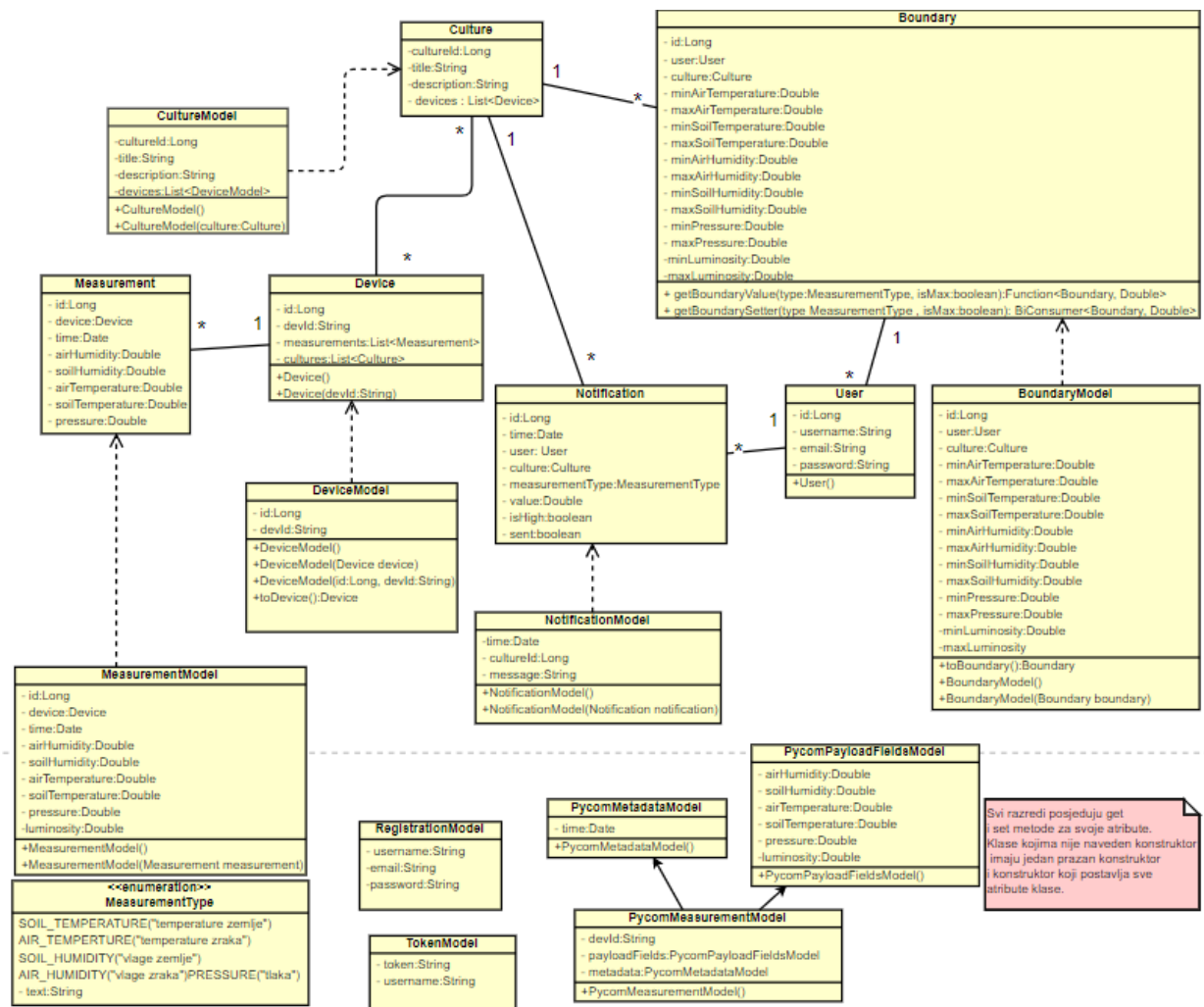
Na slici 6.3 prikazana su sučelja koja nasljeđuju JpaRepository koji je posebno proširenje Repository sučelja. JpaRepository namijenjen je izvedbi CRUD operacija, odnosno operacija za stvaranje, čitanje, ažuriranje i brisanje objekata.



Slika 6.3 - Dio Repositories

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

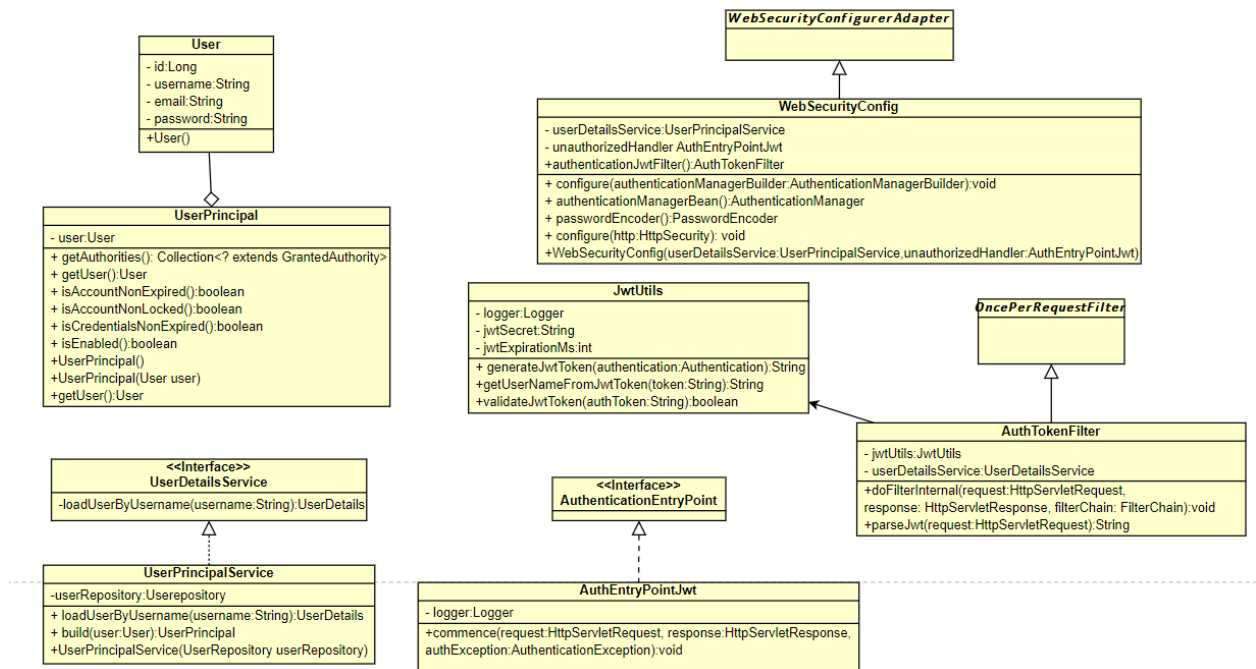
Na slici 6.4 prikazani su razredi dijela *entities* koji preslikavaju strukturu baze podataka. Rezredi dijela *models* služe za prikaz podataka na klijentskoj strani.



Slika 6.4 - Dio Entities i Models

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

Na slici 6.5 prikani su razredi koji se koriste u sklopu Spring security-ja za dodavanje sigurnosnih značajki. Služi kod registracije i prijave u sustav te omogućava ograničavanje pristupa određenom sadržaju za čiji pristup se traži autorizacija.



Slika 6.5 - Dio Security

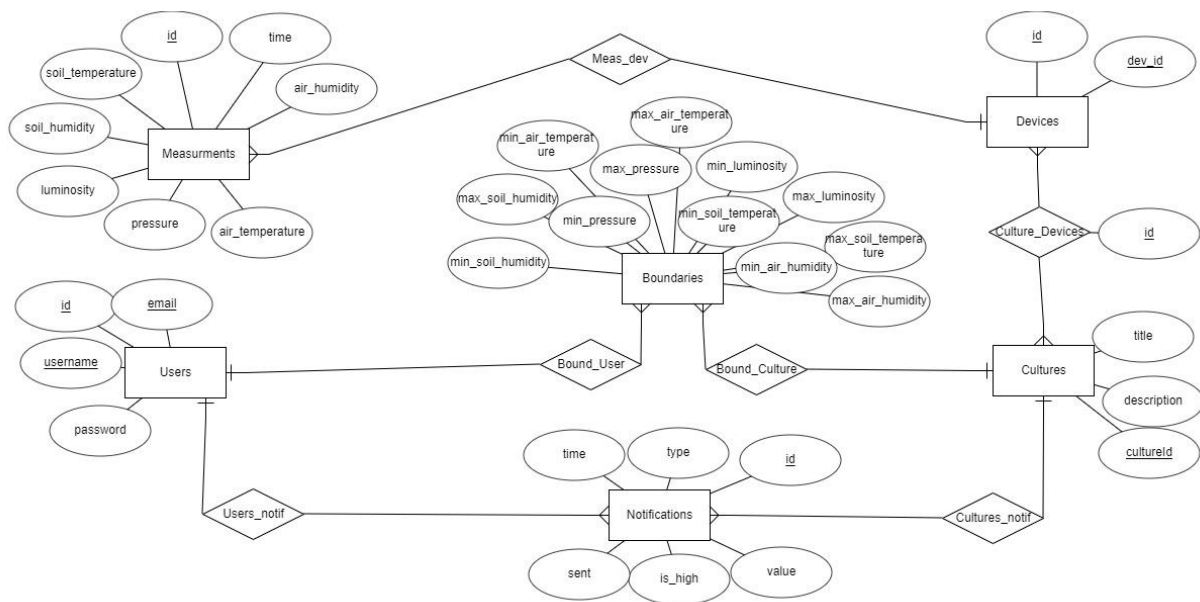
6.3 Baza podataka

Za dodavanje, pristupanje i procesiranje podataka u bazi korišten je sustav PostgreSQL. To je sustav za upravljanje objektno-relacijskim bazama podataka. Komunikaciju s PostgreSQL bazom podataka ostvarili smo alatom pgAdmin.

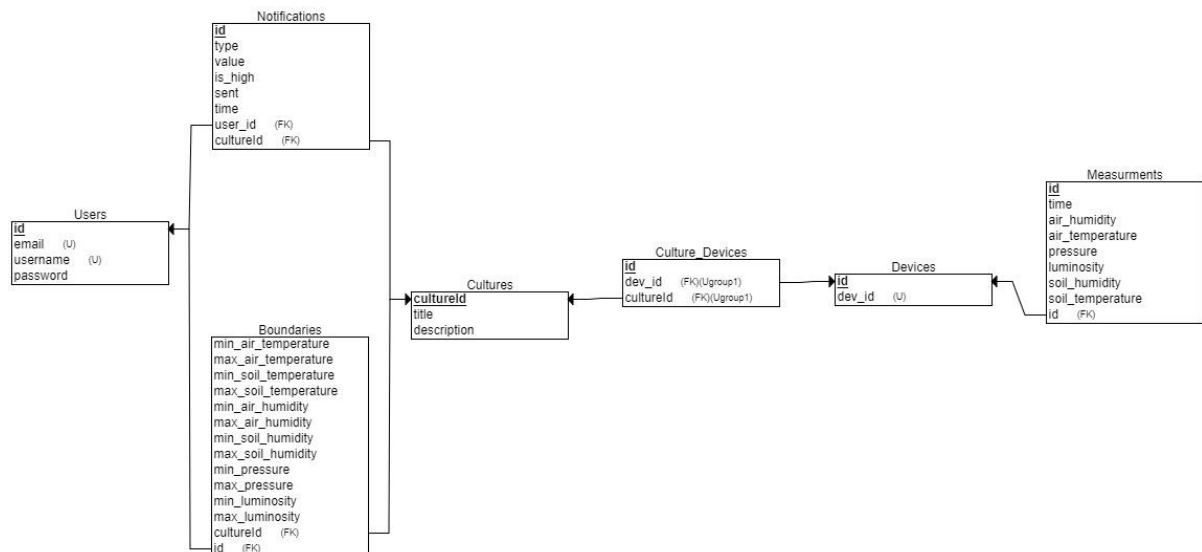
Baza podataka ove aplikacije sastoji se od sljedećih entiteta:

- Users – sprema osnovne podatke o korisniku (identifikacijski broj, korisničko ime, lozinku, email)
- Devices – sprema podatke o uređajima koji šalju svoja mjerenja (ID i naziv uređaja)
- Measurements – sprema podatke mjerenja koja šalju senzori (ID, temperaturu, vlažnost, tlak, osvjetljenje i vrijeme mjerenja)
- Boundaries – sprema gornju i donju vrijednost ograničenja za svaku vrstu mjerenja
- Notifications – sprema podatke vezane za obavijesti o odstupanju od željenih vrijednosti ili o nekoj drugoj nepravilnosti (tip obavijesti, vrijeme bilježenja obavijesti, je li obavijest već poslana korisniku, šalje li se obavijest o mjerenju koje je izvan ili ispod željenih vrijednosti i sadržaj obavijesti)
- Cultures – sprema podatke o kulturama (ID, naziv i opis kulture)

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>



Slika 6.6 - ER dijagram baze podataka



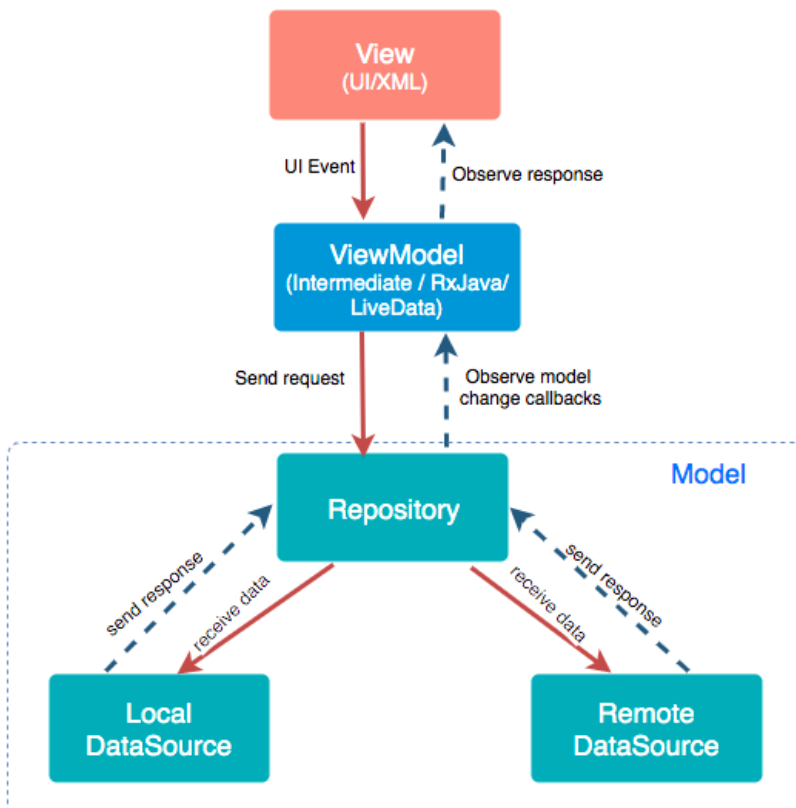
Slika 6.7 - Relacijska shema baze podataka

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

7. Android aplikacija

Aplikacija Smart Agrikulture je razvijena za mobilne uređaje s operacijskim sustavom android s minimalnom android SDK verzijom API Level 23 odnosno Android 6.0 (Marshmallow), a ciljanom android SDK verzijom API Level 29 odnosno Android 10.0(Q). Aplikacija je razvijena u okruženju Android studio u programskom jeziku Kotlin verzije 1.3.71.

Odabrana arhitektura za aplikaciju je Model-View-ViewModel (MVVM) oblikovni obrazac koji poštuje dobru programersku praksu „Separation of concerns“ odnos razdvaja poslovnu logiku od modela podataka i od pogleda. U MVVM organizaciji koda podatci se odvajaju u modele koji predstavljaju objekte u objektno-orijentiranom programiranju odnosno u našem slučaju modele podataka koji se pohranjuju u bazu podataka. U View komponenti definiramo na koji način ćemo korisniku reprezentirati naše modele podataka te nam View komponenta također služi za interakciju od strane korisnika. ViewModel komponenta je zadužena za obavljanje većine logike koja se odvija u pozadini našeg pogleda (View). ViewModel tako obavlja logičke operacije s podacima koje zatim View komponenta promatra i reprezentira korisniku. Podatci u ViewModel najčešće ne dolaze izravno iz modela podataka nego ViewModel komunicira s repozitorijem koji za njega pribavlja podatke s udaljenog servera ili iz lokalne baze podataka. U našem slučaju Smart Agriculture sve podatke dobavlja iz vanjske baze podataka preko udaljenog poslužitelja.



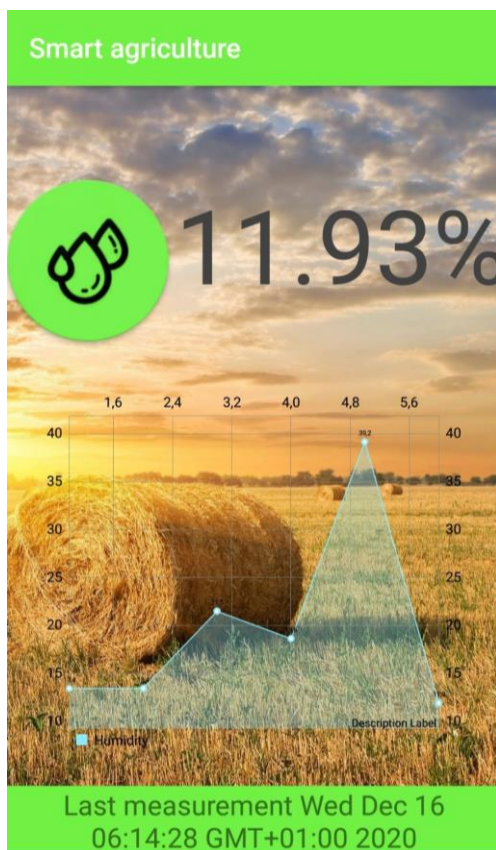
Slika 7.1 - Skica MVVM arhitekture (<https://s3.ap-south-1.amazonaws.com/mindorks-server-uploads/mvvm.png>)

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspnote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

Prilikom razvoj Aplikacije korištene su i vanjske biblioteke kao što su Retrofit za povezivanje s poslužiteljem i MPAndroidChart za prikazivanje grafova.

Retrofit je biblioteka koja nam omogućava da komuniciramo s REST poslužiteljem pretvarajući API pozive u metode koje možemo koristiti i pozivati u našem programskom kodu napisanom u Kotlinu. Verzija Retrofit-a korištena u izradi aplikacije Smart Agriculture je posljednja izdana verzija koja je u ovom trenutku (Siječanj 2021.) Retrofit 2.9.0. Slanje zahtjeva na server je asinkrona funkcija što znači da na dobivanje njenog rezultata moramo čekati određeno vrijeme. U aplikacijama koje se izrađuju za mobilne uređaje s operacijskim sustavom Android takve vrste zadatak ne smijemo izvoditi na UI Thredu odnosno glavnoj dretvi aplikacije koju koristimo za prikaz naših aktivnosti korisniku. Iz tog razloga komunikacija sa poslužiteljem je implementirana pomoću Kotlin korutina (Kotlin Corutines) koje nam omogućuju izvođenje suspend metoda čije će se izvođenje pokrenuti u pozadini, ne blokirajući glavnu dretvu.

Za prikaz rezultata mjerenja koje aplikacija dohvaća sa udaljenog poslužitelja koristi se biblioteka MPAndroidChart čiji kreator je Philipp Jahoda. Korištena je verzija 3.1.0. koja je u ovom trenutku (Siječanj 2021) posljednja izdana verzija. Navedena biblioteka nam omogućuje da za ulaznu listu točaka koje se sastoje od x i y vrijednosti, odnosno u našem slučaju trenutak obavljanja mjerenja i vrijednost mjerene veličine, korisniku prikažemo linijski graf kretanja vrijednosti mjerene veličine.



Slika 7.2 - Snimka ekrana aplikacije

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

8. iOS aplikacija

iOS aplikacija rađena je za iOS 13.0, većina stvari rađena je sa native apple frameworkcima.

Za arhitekturu je korišten MVVM, gdje su servisi odvojeni od viewControllera te im se pristupa putem viewModela. Modeli se dekodiraju u JSON pomoću protokola Decodable, dok se enkodiranje JSON-a za POST pozive radi direktno u pozivu sa JSONSeriliazation klasom.

```
struct MeasurementsModel: Decodable {
    let id: Int?
    let device: DeviceModel?
    let time: String?
    let airHumidity: Double?
    let soilHumidity: Double?
    let airTemperature: Double?
    let soilTemperature: Double?
    let pressure: Double?
}
```

```
class CulturesModel: Decodable {
    let cultureId: Int
    let title: String
    var devices: [DeviceModel]
    let description: String
}
```

Slika 8.1 – Primjeri modela

```
class CulturesViewModel {
    var cultures: [CulturesModel] = []
    var selectedCulture: CulturesModel?
    var selectedIndex: Int?

    func fetchCultures(completion: @escaping ((Result<Void, Error>->Void)){
        let service = MeasurementsService()

        service.fetchCultures { (result) in
            switch result {
            case .success(let cultures):
                self.cultures = cultures
                completion(.success(()))
            case .failure(let error):
                self.cultures = []
                completion(.failure(error))
            }
        }
    }

    func culturesForTVC(forIndexPath indexPath: IndexPath) -> CultureCellModel? {
        return CultureCellModel(culture: cultures[indexPath.row])
    }

    func deleteCulture(cultureID: Int){
        let service = MeasurementsService()
        service.deleteCulture(cultureID: cultureID)
    }
}

class MeasurementsService {
    let baseUrlString = Constants.baseUrl

    func fetchMeasurementData(completion: @escaping ((Result<[MeasurementsModel], Error>->Void)){
        let urlString = baseUrlString + "/api/measurement/all"
        guard let url = URL(string: urlString) else {return}
        guard let token = UserCredentials.shared.getToken() else {
            return
        }

        var request = URLRequest(url: url)
        request.httpMethod = "GET"
        request.addValue("Bearer \(token)", forHTTPHeaderField: "Authorization")

        URLSession.shared.dataTask(with: request) { (data,response,error) in
            if let data = data {
                do {
                    let model = try JSONDecoder().decode([MeasurementsModel].self, from: data)
                    completion(.success(model))
                } catch let decodeError {
                    completion(.failure(decodeError))
                    return
                }
            } else if let error = error {
                completion(.failure(error))
            }
        }.resume()
    }
}
```

Slika 8.2 – Primjeri viewModela i servisa

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspnote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

Za networking je većinom korišten framework URLSession, samo za register i login je korišten Alamofire. Ispod je primjer POST (alamofire), POST (URLSession) i DELETE (URLSession) poziva.

```
extension LoginViewController {
    func loginUserWith(username: String, password: String) {

        let parameters: [String: String] = [
            "username": username,
            "password": password
        ]

        let urlStr = Constants.baseUrl + Constants.loginEndPoint

        AF.request(urlStr, method: .post, parameters: parameters, encoding: JSONEncoding.default)
            .validate()
            .responseDecodable(of: UserModel.self) { response in
                switch response.result {
                case .success(let user):
                    UserCredentials.shared.setToken(with: user.token)
                    self.userModel = user
                    self.navigateToHome()
                case .failure(let err):
                    print(err)
                }
            }
    }
}

func deleteCulture(cultureID: Int) {
    let urlString = baseUrlString + "/api/culture/delete/\(cultureID)"
    guard let url = URL(string: urlString) else {return}

    guard let token = UserCredentials.shared.getToken() else {return}

    var request = URLRequest(url: url)
    request.httpMethod = "DELETE"
    request.addValue("Bearer \(token)", forHTTPHeaderField: "Authorization")

    URLSession.shared.dataTask(with: request) { data, response, error in
        if let data = data {
            print(data)
        } else if let err = error {
            print(err.localizedDescription)
        } else if let response = response {
            print(response)
        }
    }.resume()
}
```

Slika 8.3 – Primjeri poziva

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

```

func addCulture(cultureID: String, title :String, deviceID: String, deviceDevID: String, description: String) {

    let urlString = baseUrlString + "/api/culture/add"
    guard let url = URL(string: urlString) else {return}

    guard let token = UserCredentials.shared.getToken() else {return}

    let data = [
        "cultureId": cultureID,
        "title": title,
        "devices": [
            ["id": deviceID,
            "devId": deviceDevID]
        ],
        "description": description
    ] as [String : Any]

    guard let jsonData = try? JSONSerialization.data(withJSONObject: data, options: []) else { return }

    var request = URLRequest(url: url)
    request.httpMethod = "POST"
    request.addValue("Bearer \(token)", forHTTPHeaderField: "Authorization")
    request.addValue("application/json", forHTTPHeaderField: "Content-Type")
    request.httpBody = jsonData

    URLSession.shared.dataTask(with: request) { data, response, error in
        if let data = data {
            print(data)
        } else if let err = error {
            print(err.localizedDescription)
        }
    }.resume()
}

```

Slika 8.4 – Primjeri poziva

Za crtanje grafova korišten je pod Charts, u metodu setupCharts se proslijedi polje Double-ova koje predstavlja različita mjerenja za određenu kulturu i senzor.

UserCredentials je singleton čija je jedina dužnost pratiti jedinstveni token korisnika.

```

func setUpChart(array: [Double]){
    chartView.delegate = self

    chartView.frame = CGRect(x: 0, y: 0, width: 350, height: 400)
    chartView.center = view.center
    view.addSubview(chartView)

    let xAxisLimit = ChartLimitLine(limit: 10)
    xAxisLimit.lineWidth = 2

    let leftAxis = chartView.leftAxis
    leftAxis.removeAllLimitLines()
    leftAxis.axisMinimum = 0
    leftAxis.axisMaximum = 60
    chartView.rightAxis.enabled = false

    chartView.legend.form = .line

    var entries = [ChartDataEntry]()

    for x in 0 ..< array.count {
        entries.append(ChartDataEntry(x: Double(x), y: Double(array[x])))
    }

    let set = LineChartDataSet(entries: entries, label: "Temperature")
    set.colors = ChartColorTemplates.pastel()

    set.fillColor = ■
    set.drawFilledEnabled = true

    let data = LineChartData(dataSet: set)
    chartView.data = data
}

```

```

final class UserCredentials {
    private var token: String?

    private init(){}
    static let shared = UserCredentials()

    func setToken(with token: String){
        self.token = token
    }

    func getToken() -> String? {
        return token
    }
}

```

Slika 8.5 – Charts i UserCredentials

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

9. Upute za korištenje

(biti će naknadno napisano)

- inicijalizacija uređaja sa senzorima
- pokretanje poslužitelja
- instalacija i korištenje aplikacija

Sustav Interneta stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspote i Pycom	Verzija: <1.0>
Tehnička dokumentacija	Datum: <11/11/2020>

10. Literatura

MVVM ref: <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel>

Retrofit docs: <https://square.github.io/retrofit/>

Retrofit src: <https://github.com/square/retrofit>

Kotlin Coroutines docs: <https://kotlinlang.org/docs/reference/coroutines-overview.html>

MPAndroidChart src & docs: <https://github.com/PhilJay/MPAndroidChart>