Sustav Interneta Stvari za mjerenje i prikaz vrijednosti senzora koristeći Waspmote i Pycom
Tehnička dokumentacija
Verzija <2.0>

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

Sadržaj

Tehnička dokumentacija

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 Waspmote uređaja i jednog Pycom uređaja sa povezanim senzorima,
- poslužiteljski dio,
- Android aplikacija i
- 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 grafovim. 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:

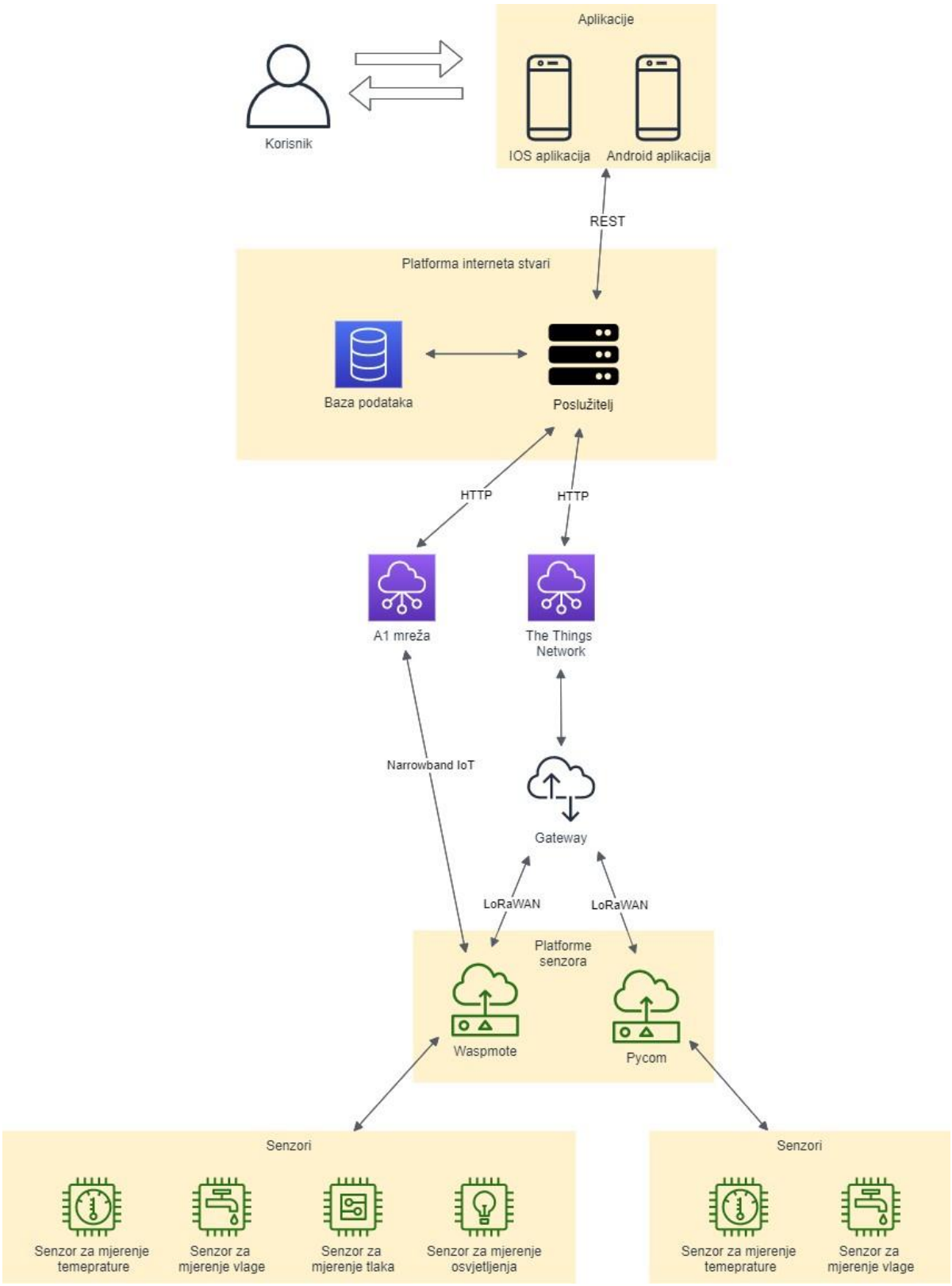
- mobilne aplikacije,
- platforma Interneta Stvari i
- pametna okolina.

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. Mrežni prilaz (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 Waspmote i PyCom platforme. Sklopovske platforme komuniciraju s poveznikom putem nekoliko komunikacijskih protokola, a to su: LoRaWaN za jedan Waspmote i jedan Pycom uređaj, te Narrowband IoT za drugi Waspmote uređaj. Podaci koje senzori mjere u ovom projektu su osvjetljenje, te temperatura, vlaga i tlak zraka.

Ova arhitektura sustava može se vidjeti na Slika 1.



Slika 1 - Arhitektura sustava

3. LoRaWAN

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

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.

Mrežni prilaz (eng. *gateway*) je posrednik između uređaja i poslužitelja. Poslužitelj komunicira s ‚režnim prilazom putem Interneta i govori mu kako ostvariti komunikaciju s uređajem. Uređaji koriste LoRaWAN kako bi se spojili na poveznik, koji onda njihove poruke prosljeđuje na The Things Network.

The Things Network (TTN) je infrastruktura koja omogućava povezivanje uređaja Interneta Stvari. U ovom projektu koristimo TTN stranicu za konfiguraciju komunikacije uređaja s poslužiteljem i dekodiranje podataka koje šaljemo.

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 uređaju Waspmote prikazan je na Kod 1, dok je kodiranje podataka na uređaju Pycom prikazano na Kod 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;
```

Kod 1 - Kodiranje podataka na Waspmote uređaju

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

Kod 2 - Kodiranje podataka na Pycom uređaju

Kao što se može vidjeti na Kod 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 poretku).

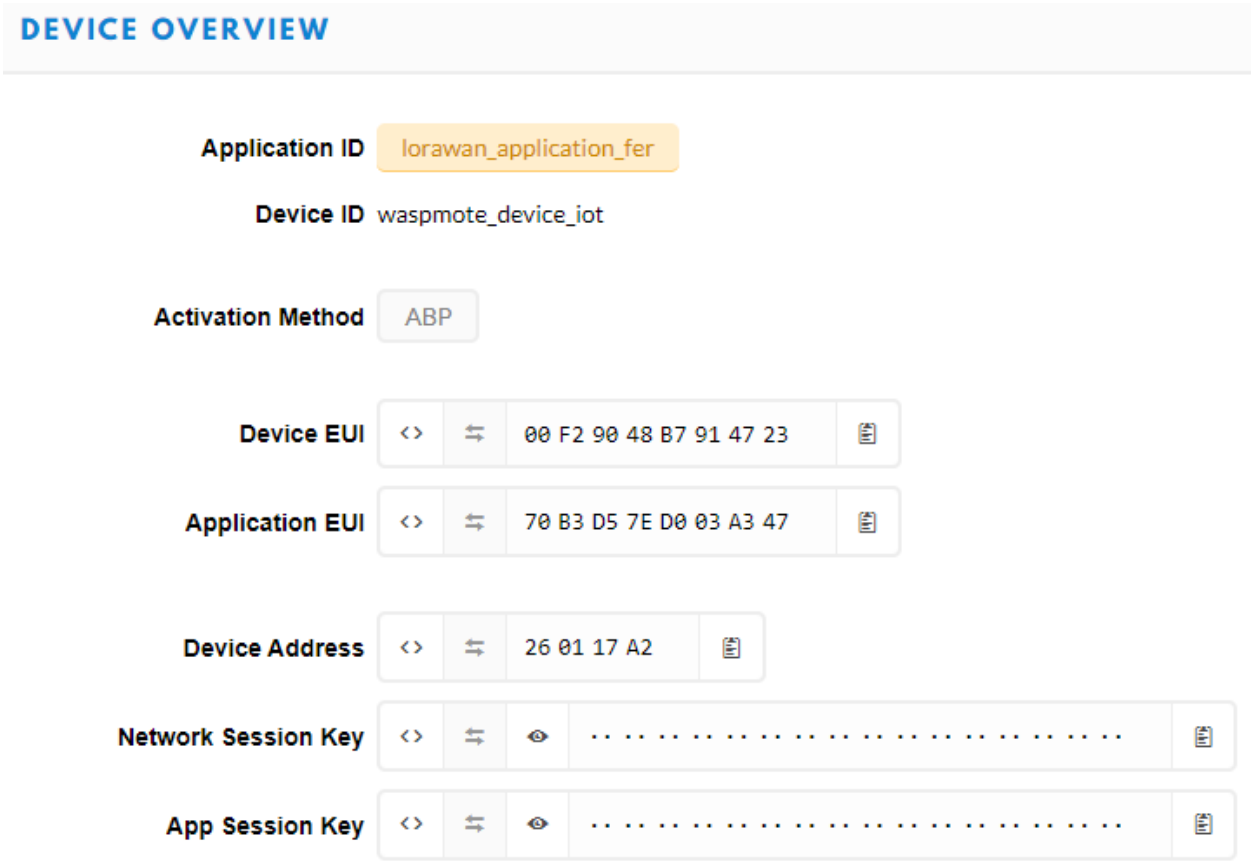
Slanje podataka s aplikacije na uređaj (eng. *downlink*)

Budući da poveznik ne može primati 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 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 konzole na The Things Networku. 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 podataka u The Things Network konzoli prikazan je na Slika 2, dok je korištenje tih podataka u kodu prikazano na Kod 3.



Slika 2 - Prikaz podataka u TTN konzoli

```
char DEVICE_EUI[]  = "00F29048B7914723";
char DEVICE_ADDR[] = "260117A2";
char NWK_SESSION_KEY[] = "97C11C3F0B95BDA7AB3068A48A52FA43";
char APP_SESSION_KEY[] = "1058EEBFFD791F1C68F10071B312CCB4";

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();
}
```

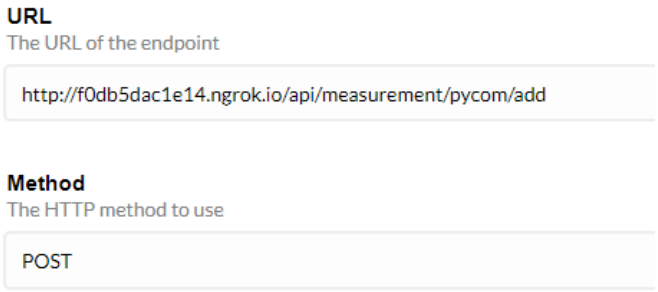
Kod 3 - Korištenje TTN podataka u kodu

Podaci koje šalje uređaj vidljivi su na konzoli. Budući da se na mrežu šalju kodirani binarni podaci, potrebno je napisati dekoder podataka kako bi se oni mogli proslijediti na poslužitelj u razumljivom formatu. Primjer dekodera prikazan je na Kod 4.

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

Kod 4 - Primjer dekodera

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 Slika 3.



Slika 3 - HTTP integracija na TTN 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 Libeliumov BG96 čipset tvrtke Quectel koji podržava standarde NB-IoT i CAT-M1. 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 operatora 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 metoda GET i POST direktno komunicira s poslužiteljem te mu šalje podatke o mjerenjima sa svojih senzora.

Sve funkcije vezane za komunikaciju preko NB-IoT modula dio su biblioteke WaspBG96.h te ju iz tog razloga nužnu 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 operatora** i **vrsta operatora**. Nakon svega navedenog modul je spreman za slanje HTTP zahtjeva na poslužitelj što u ovom slučaju radimo preko GET metode koja u upitu (query) 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 Waspmoteovog 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 znakove te iz takvog zapisa parsirati podatke potrebne za spremanje u bazu.

Ovaj proces prikazan je na Kod 5 do Kod 10.

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

Kod 5 - Postavljanje operatera

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

Kod 6 - Paljenje modula

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

Kod 7 - Konfiguriranje konekcije

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

Kod 8 - Uspostavljanje konekcije s mrežom

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

Kod 9 - Postavljanje DNS-a

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

Kod 10 - Slanje zahtjeva na HTTP poslužitelj

5. Senzori

5.1 Waspmoteovi senzori

Senzori se na uređaj Waspmote 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 Waspmote uređaju koristi senzor BME280, koji može mjeriti vlagu, tlak i temperaturu. U nastavku su prikazane specifikacije svakog od senzora.

Senzor temperature zraka
- Područje rada: -40º ~ +85º C
- Područje mjerenja: 0 ~ +65º C
- Točnost: ± 1º C
- Vrijeme odziva: 1.65 sekundi

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

Senzor tlaka zraka
- Područje rada: -40º ~ +85º C
- Područje mjerenja: 30 ~ 110 kPa
- Točnost: ± 0.1 kPa (na temperaturi od 0 ~ +65º 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 Kod 11.

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

Kod 11 - Pozivanje funkcija za vrijednosti senzora na Waspmote uređaju

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

Senzor osvjetljenja:
- Područje očitanja: 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 ºC

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 luxima.

Očitavanje podataka:

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

5.2 Pycomovi senzori

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

Senzor temperature zraka

- Područje optimalnog rada: -10º ~ +85º C
- Područje mjerenja: -40º ~ +125º C
- Točnost: ± 1º C (na temperaturi -10 do 85 º C)
- Vrijeme odziva: 5.1 s

Senzor vlage zraka

- Područje optimalnog rada: -10º ~ +60º C
- Područje mjerenja: 0 ~ 100% relativne vlage
- Točnost: ± 5%
- Vrijeme odziva: 18 s

Programiranje senzora na Pycomu moguće je uz pomoć dodatka Pymakr 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 Kod 12 i Kod 13 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')
```

Kod 12 - Pozivanje metoda za dohvat vrijednosti senzora na uređaju Pycom

```
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
```

Kod 13 - Metode za dohvat vrijednosti temperature i vlage zraka sa senzora

6. Poslužitelj

6.1 REST API

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

Četiri osnovna dijela REST API zahtjeva su:

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

Korištene API-jeve krajnje točke i njihove HTTP metode:

- /api/measurement/pycom/add (POST) – dodavanje mjerenja s Pycom i Waspmote uređaja,
- /api/measurement/waspmote/add (GET) – dodavanje mjerenja s Waspmote 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 culture,
- /api/culture/all (GET) – dohvaćanje svih kultura,
- /api/culture/{cultureId}/devices/delete/{devId} (DELETE) – uklanjanje uređaja iz culture,
- /api/culture/delete/{id} (DELETE) – brisanje culture,
- /api/boundaries (POST) – dodavanje granice mjerenja,
- /api/boundaries/{cultureId} (GET) – dohvaćanje granica mjerenja određene culture,

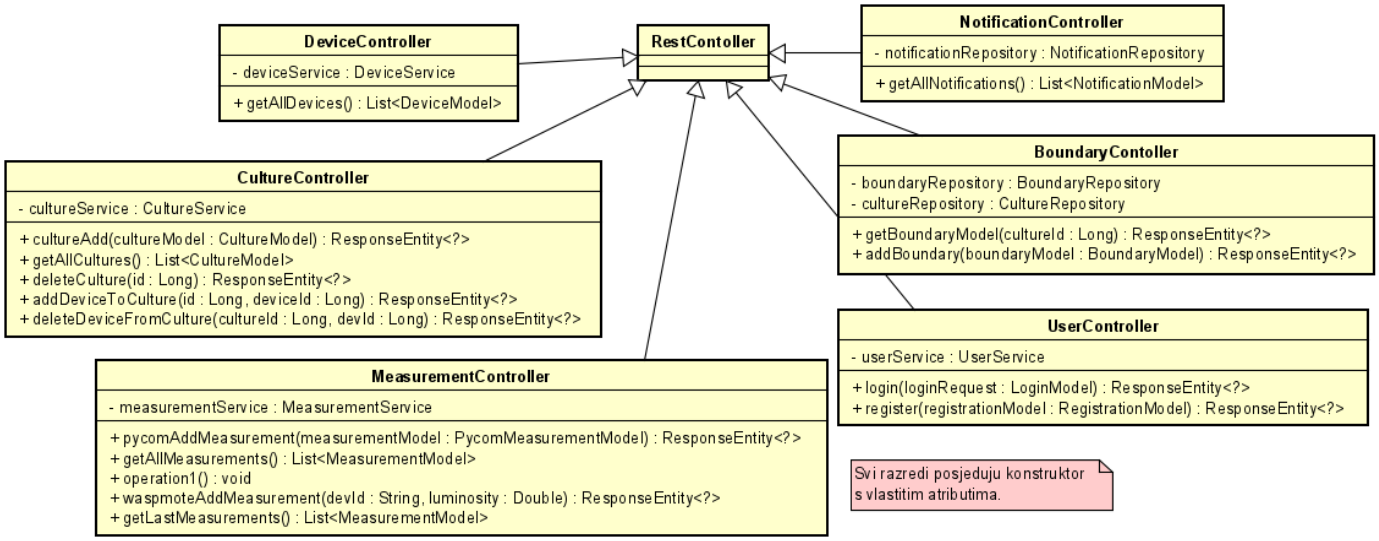
- /api/auth/register (POST) – slanje podataka za registraciju,
- /api/auth/login (POST) – slanje podataka za prijavu,
- /api/notifications (GET) – dohvaćanje svih obavijesti i
- /api/devices (GET) – dohvaćanje svih uređaja.

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

6.2 Dijagrami razreda

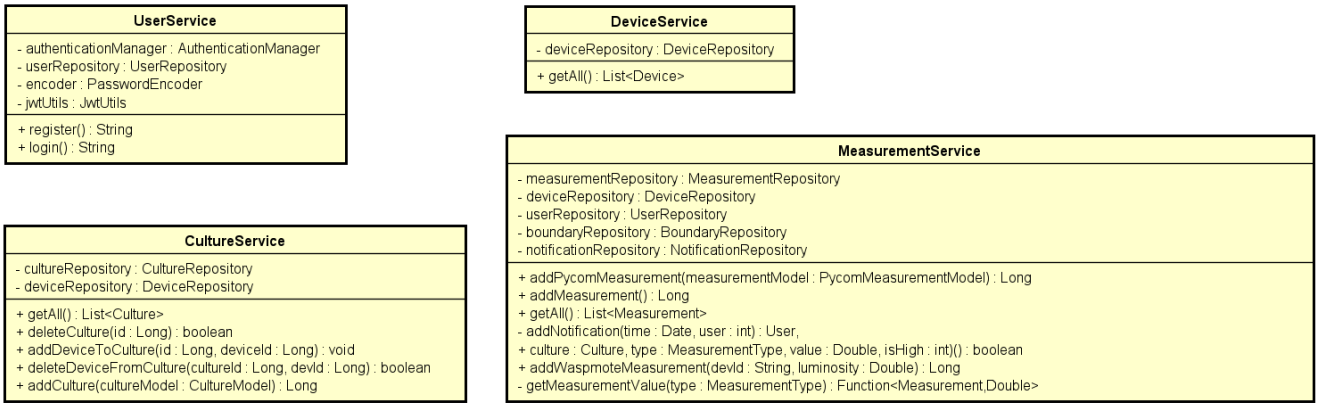
Na nalaze se razredi koji pripadaju poslužiteljskom dijelu (REST API).

Na Slika 4 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 preslikavaju iz JSON objekta.



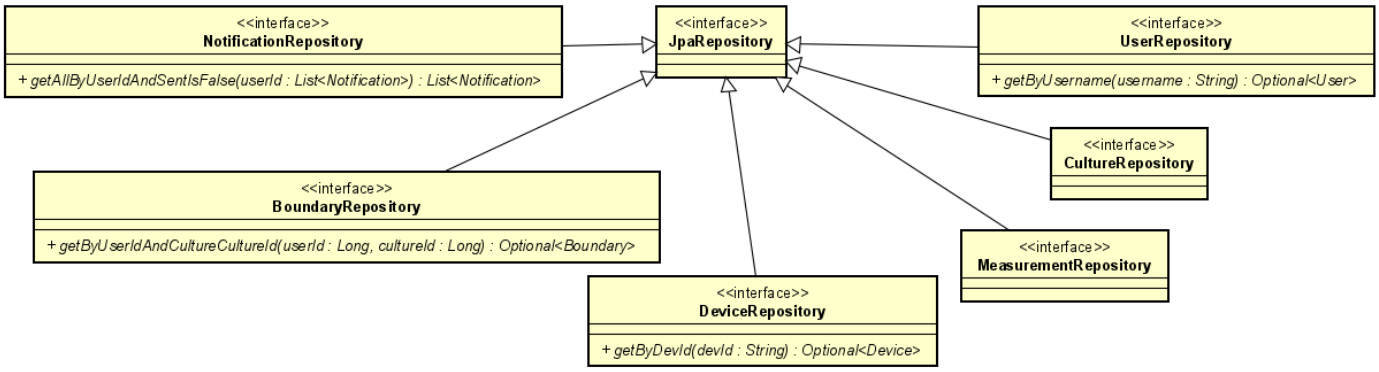
Slika 4 - Kontroleri

Slika 5 prikazuje razrede koji služe kontroleru za manipulaciju podacima u bazi.



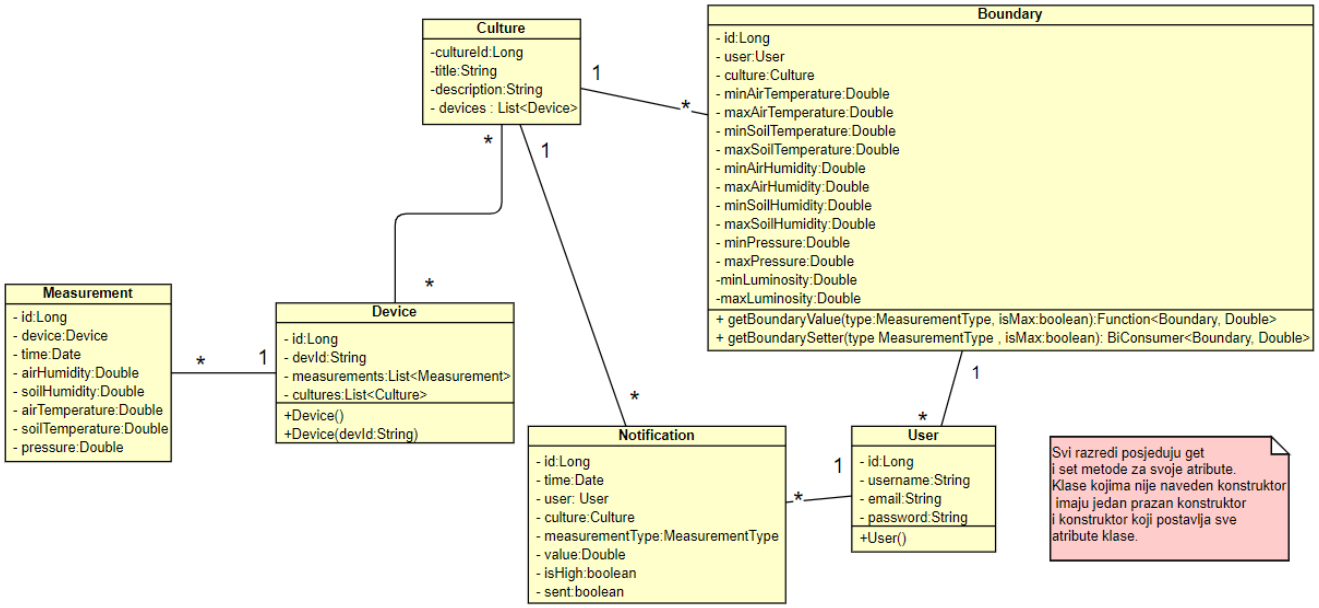
Slika 5 - Usluge

Na Slika 6 prikazana su sučelja koja nasljeđuju JpaRepository koji je posebno proširenje sučelja Repository. JpaRepository namijenjen je izvedbi CRUD operacija, odnosno operacija za stvaranje, čitanje, ažuriranje i brisanje objekata.

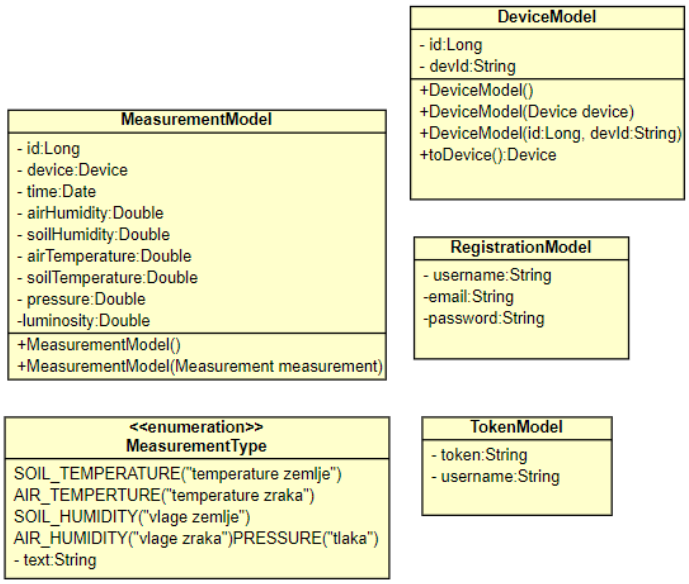


Slika 6 - Repozitoriji

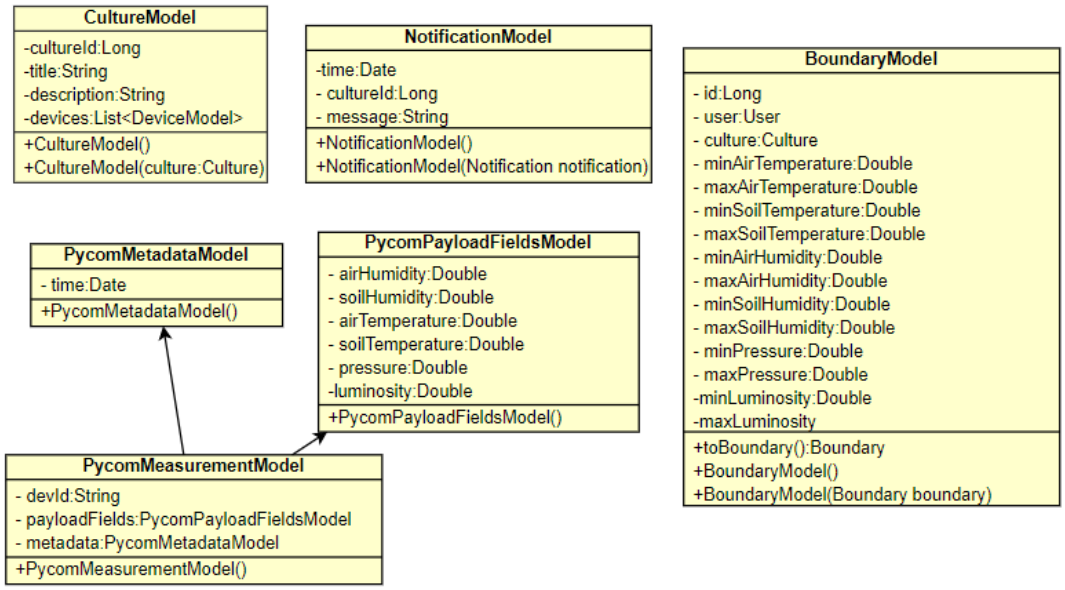
Na Slika 7 do Slika 9 prikazani su razredi dijela entiteta koji preslikavaju strukturu baze podataka i razredi modela koji služe za prikaz podataka na klijentskoj strani.



Slika 7 - Prvi dio entiteta

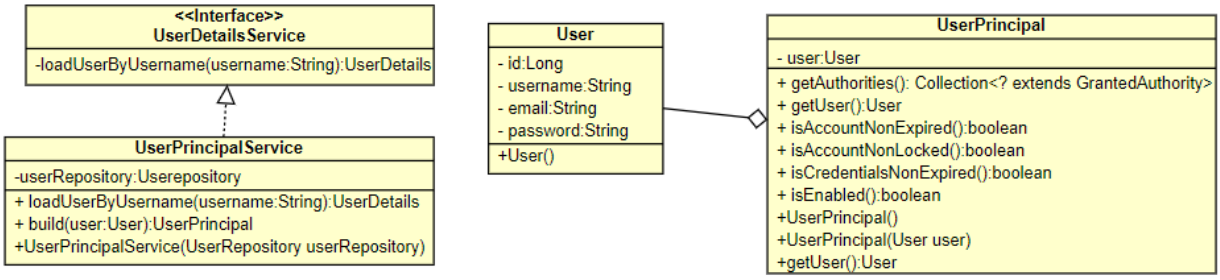


Slika 8 - Drugi dio entiteta

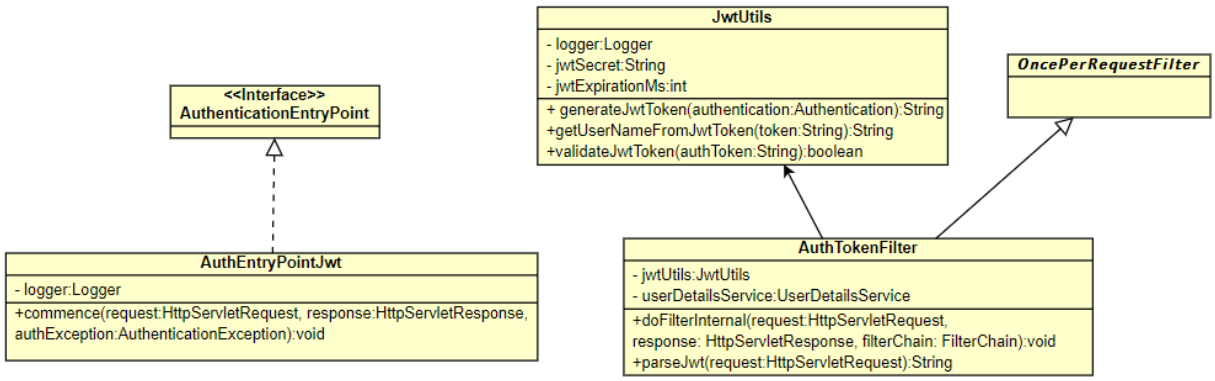


Slika 9 - Treći dio entiteta

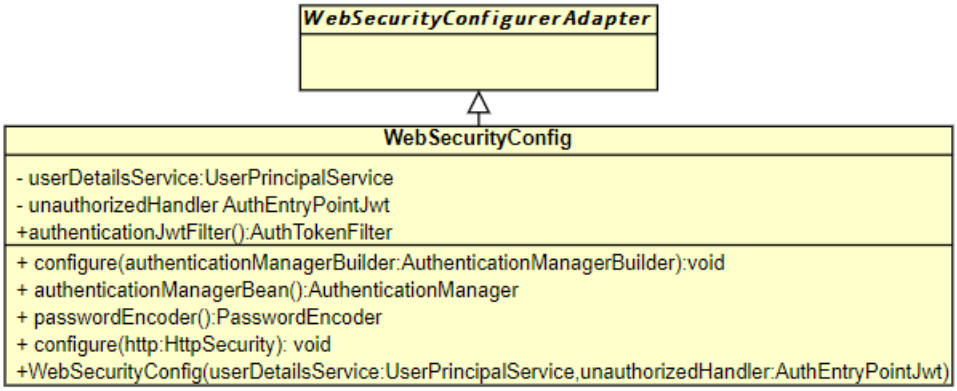
Na Slika 10Slika 11 i Slika 12. prikani su razredi koji se koriste u sklopu Spring security-ja za dodavanje sigurnosnih značajki. Služi kod registracije i prijave u sustav te omogućava ograničavanje pristupa određenom sadržaju za čiji pristup se traži autorizacija.



Slika 10 - Prvi dio sigurnosti



Slika 11 - Drugi dio sigurnosti

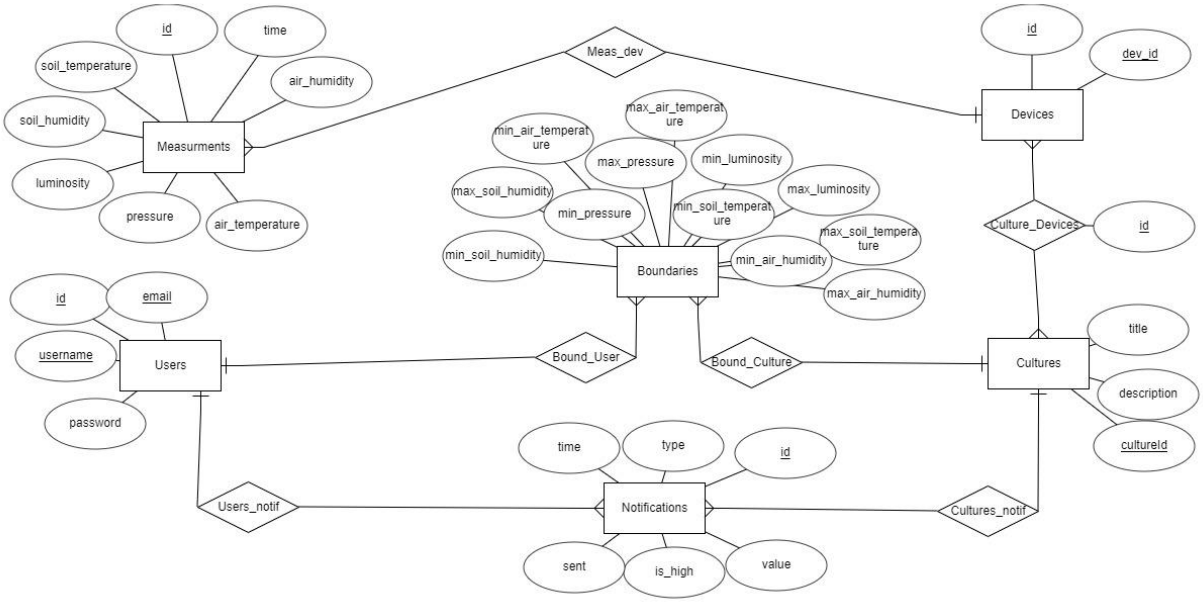


Slika 12 - Treći dio sigurnosti

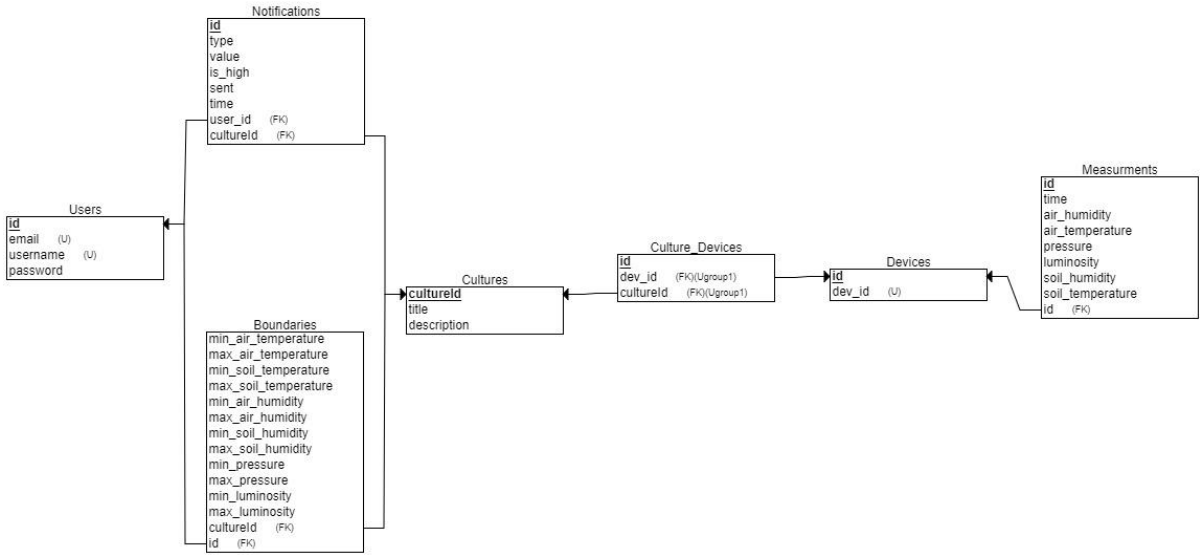
6.3 Baza podataka

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) i
- Cultures – sprema podatke o kulturama (ID, naziv i opis kulture).



Slika 13 - ER dijagram baze podataka

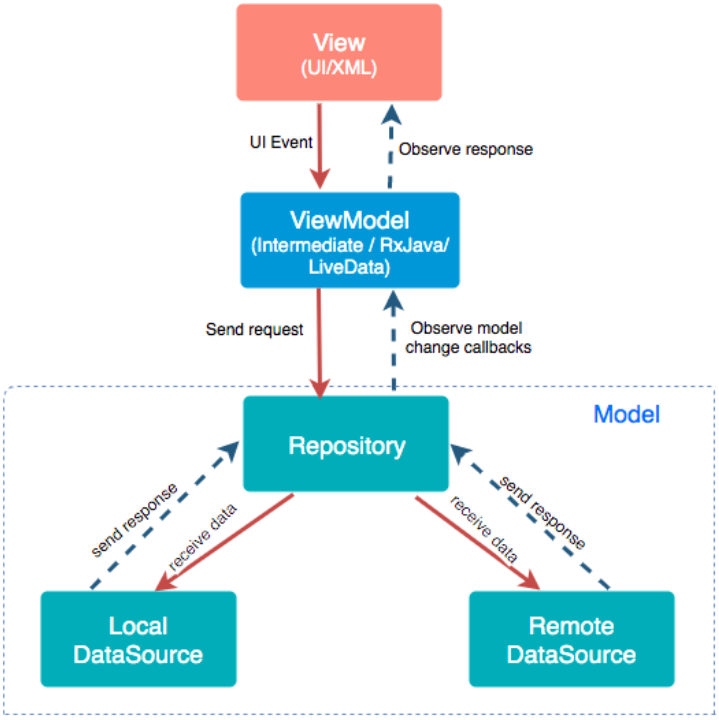


Slika 14 - Relacijska shema baze podataka

7. Android aplikacija

Aplikacija Smart Agriculture je aplikacija razvijena za Android mobilne uređaje. Minimalna verzija operacijskog sustava Android koju aplikacija podržava je API SDK verzija Level 26 odnosno Android 8.0 (Oreo), a ciljana Android SDK verzija je API Level 29 odnosno Android 10.0(Q). Aplikacija je razvijena u okruženju Android studio u programskom jeziku Kotlin verzije 1.4.21.

Odabrana arhitektura za aplikaciju je *Model-View-ViewModel* (MVVM) - oblikovni obrazac koji poštuje dobru programersku praksu *„Separation of concerns“* u kojem je poslovna logika razdvojena 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 prikazati naše modele podataka te nam ona također služi i za interakciju sa samim korisnikom. *ViewModel* komponenta je zadužena za obavljane većine logike koja se odvija u pozadini našeg pogleda (*View*). *ViewModel* tako obavlja logičke operacije s podatcima 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 aplikacija Smart Agriculture sve podatke dobavlja iz vanjske baze podataka preko udaljenog poslužitelja.



Slika 15 - Skica arhitektura MVVM (https://s3.ap-south-1.amazonaws.com/mindorks-server-uploads/mvvm.png)

```
signInButton.setOnClickListener {  it: View!


    loading.visibility = View.VISIBLE

    val repository = Repository()
    val viewModelFactory = LoginViewModelFactory(repository)

    viewModel = ViewModelProvider( owner: this, viewModelFactory).get(LoginViewModel::class.java)
    val loginModel = LoginModel(username.text.toString(), password.text.toString())
    viewModel.login(loginModel)

    viewModel.responseLiveData.observe( owner: this, Observer { response ->
        loading.visibility = View.GONE
        if (response.isSuccessful) {
            User.user.token = response.body()?.token.toString()
            User.user.username = response.body()?.username.toString()
            val intent = Intent( packageContext: this@LogInActivity, CulturesListActivity::class.java)
            startActivity(intent)
        } else {
            loginErrorTextView.visibility = View.VISIBLE
            loginErrorResponse.text = "${response.message()}  ${response.code()}"
            loginErrorResponse.visibility = View.VISIBLE
        }
    })
}
```

Kod 14 - Primjer korištenja MVVM-a

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

Retrofit je biblioteka koja omogućava komunikaciju s REST poslužiteljem pretvarajući API pozive u metode koji se mogu koristiti i pozivati u programskom kodu napisanom u Kotlinu. Verzija Retrofit-a korištena u izradi aplikacije Smart Agriculture je posljednja izdana verzija koja je trenutno bila dostupna (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 Threadu* 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.

```
class LoginViewModel (private val repository: Repository) : ViewModel() {

    val responseLiveData: MutableLiveData<Response<TokenModel>> = MutableLiveData()

    fun login(loginModel: LoginModel) {
        viewModelScope.launch {  this: CoroutineScope
            val response = repository.login(loginModel)
            responseLiveData.value = response
        }
    }
}
```

```
class Repository {

    suspend fun login(loginModel: LoginModel): Response<TokenModel> {
        return Retrofit.api.login(loginModel)
    }
```

```
interface ApiService {

    @POST( value: "/api/auth/login")
    suspend fun login(@Body loginModel: LoginModel) : Response<TokenModel>
```

Kod 15 - Hijerarhija poziva Retrofita

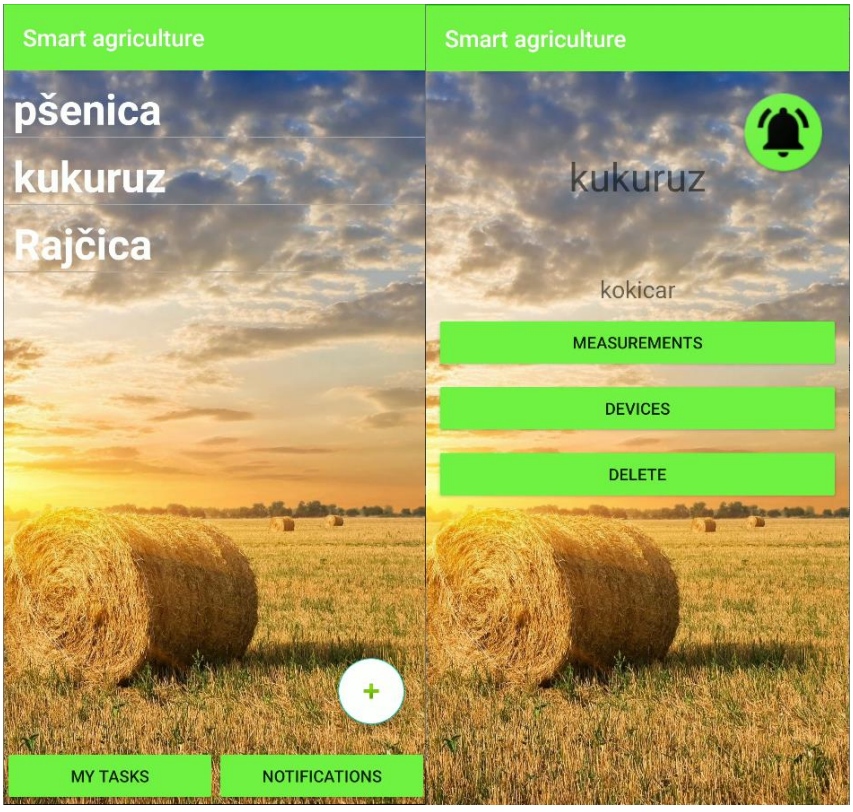
Za prikaz rezultata mjerenja koje aplikacija dohvaća sa udaljenog poslužitelja koristi se biblioteka MPAndroidChart (verzija 3.1.0) čiji kreator je Philipp Jahoda. Navedena biblioteka nam omogućuje da za ulaznu listu točaka koje se sastoje od x i y vrijednosti, odnosno u našem slučaju trenutak obavljanja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

```
        val airTemperatureEntries = mutableListOf<Entry>()
        if (measurementsList != null) {
            var i: Float = 1.0f
            for (m: MeasurementModel in measurementsList) {
                airTemperatureEntries.add(
                    Entry(
                        i++,
                        m.airTemperature.toFloat()
                    )
                )
            }
        }
```

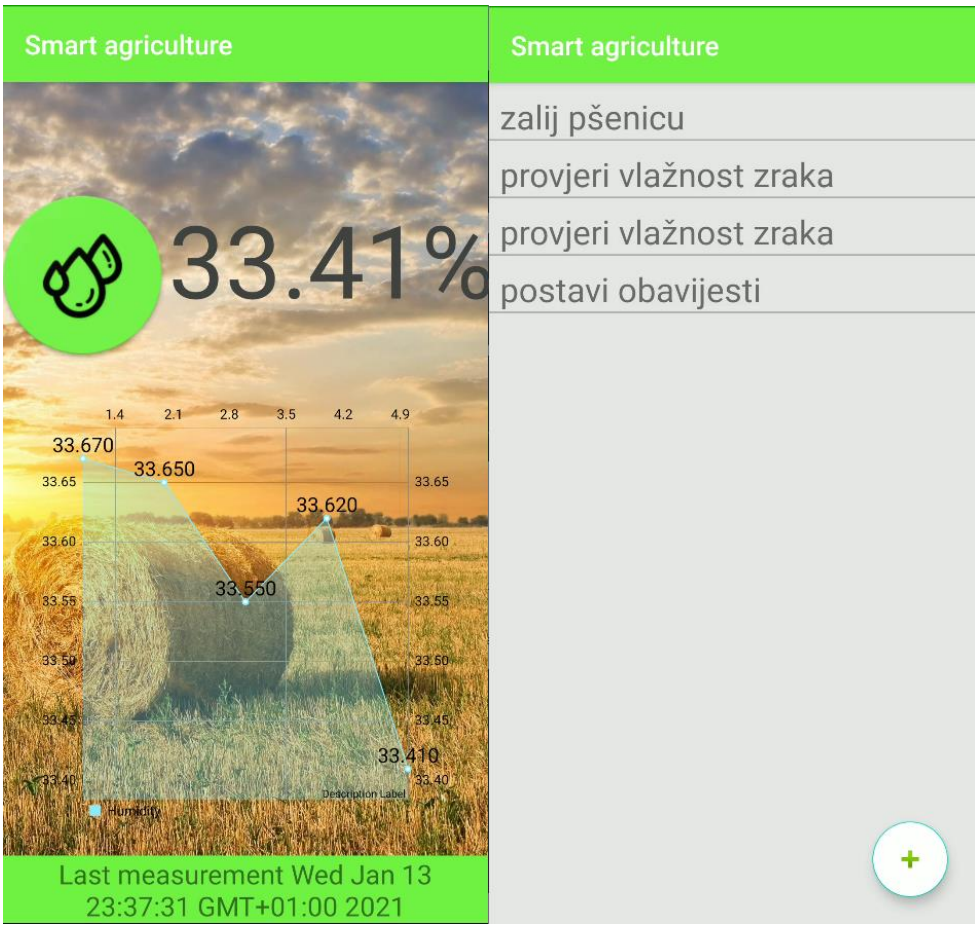
```
val lineDateSetAirTemperature: LineDataSet =
    LineDataSet(airTemperatureEntries, label: "Temperatures")
lineDateSetAirTemperature.valueTextSize = 15f
lineDateSetAirTemperature.setDrawFilled(true)
```

Kod 16 - Dodavanje podataka za graf

Sama aplikacija sastoji se od nekoliko ekrana koji se nadovezuju jedan na drugog. Početni pogled aplikacije nudi korisniku mogućnost prijave ili registracije na sustav. Nakon uspješne prijave u sustav, pojavljuje se prikaz lista kultura za zadanog korisnika. Korisnik može pritiskom na gumb dodati novu kulturu sa naslovom i njezinim opisom te zadati zadani uređaj za tu kulturu. Na listi za kulture dostupni su i gubmi za zadatke u kojima korisnik može zapisivati zadatke koje treba napraviti te gumb za obavijesti u kojem može uključiti ili isključiti obavijest od strane aplikacije. Pritiskom na kulturu u listi kultura dolazi se do stranice određene kulture. Na toj stranici korisnik može za pojedinu kulturu postaviti granice za njezina mjerenja. Te granice služe kako bi korisnika obavijestile kada mjerenja prelaze zadane kriterije. Za svaku kulturu dostupna je i lista uređaja za tu kulturu te je tu listu moguće uređivati dodavanjem ili brisanjem uređaja. Osim prikaza uređaja moguće je i dohvatiti mjerenja koje su ti uređaji izvršili. Svaka kultura sadrži 4 razlčičita mjerenja koja prikazuje : temperaturu zraka, vlažnost zraka, tlak te svjetlost.



Slika 16 - Lista kultura i detalji kutlure



Slika 17 - Prikaz mjerenja i liste zadataka

8. iOS aplikacija

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

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 klasom JSONSeriliazation.

```swift
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?
}
```

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

Slika 18 – Primjeri modela

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

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

        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 = MeasurmentsService()
        service.deleteCulture(cultureID: cultureID)
    }
}
```

```swift
class MeasurmentsService {

    let baseUrlString = Constants.baseUrl

    func fetchMeasurmentData(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 19 - Primjeri viewModela i servisa

Za povezivanje je većinom korišten radni okvir URLSession, samo za register i login je korišten Alamofire. Ispod je primjer slanja zahtjeva pomoću metoda: POST (alamofire), POST (URLSession) i DELETE (URLSession).

```
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 20 - Primjeri slanja zahtjeva na poslužitelj

```swift
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 21 - Primjer slanja kulture na poslužitelj

Za crtanje grafova korišten je pod Charts, u metodu setupCharts se proslijedi polje Doubleova koje predstavlja različita mjerenja za određenu kulturu i senzor.
UserCredentials je singleton čija je jedina dužnost pratiti jedinstveni token korisnika.

```swift
func setUpCharth(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
}
```

```swift
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 22 - Charts i UserCredentials

9. Upute za korištenje

Dohvaćanje projekta s gitlaba:

1. Otvoriti terminal na željenoj lokaciji i kopirati sljedeću naredbu "git clone https://gitlab.tel.fer.hr/pdp/preddiplprojekt_2020_pametnapoljoprivreda.git".
2. Otvoriti klonirani direktorij na vašem operacijskom sustavu.

Konfiguracija The Things Networka:

1. Registrirati se na stranici: https://www.thethingsnetwork.org/
2. U The Things Network (TTN) konzoli registrirati novu aplikaciju.
3. Na profilu aplikacije kliknuti na *Integrations*.
4. Dodati HTTP integraciju tako da se u polje URL upiše URL pokrenutog servera.
5. Na profilu aplikacije kliknuti na *Payload Formats*.
6. U polju *decoder* kopirati kod koji se nalazi u datoteci *preddiplprojekt_2020_pametnapoljoprivreda/Software/Pycom/dekoder.txt* za uređaj Pycom ili *preddiplprojekt_2020_pametnapoljoprivreda/Software/Waspmote/LoRaWAN/Dekoder.txt* za uređaj Waspmote.
7. U profilu aplikacije registrirati novi uređaj (Za dohvaćanje Device EUI-ja pogledati poglavlje za inicijalizaciju uređaja.
8. U postavkama uređaja promijeniti metodu aktivacije na ABP.
9. Na dnu profila uređaja, zabilježiti vrijednosti polja *devAddr*, *nwkSKey* i *appSKey* koje će se koristiti na uređajima.

Inicijalizacija Waspmote uređaja sa senzorom osvjetljenja i NB-IoT komunikacijskim modulom;

Prije korištenja samog uređaja potrebno je instalirati Waspmote IDE u kojem će se otvarati kod i uređivati parametri potrebni za ispravno funkcioniranje uređaja. Također, preko IDE-a će se učitavati kod na sam Waspmote uređaj. Nakon spajanja uređaja na računalo, operacijski sustav će automatski prepoznati uređaj i instalirati potrebne upravljačke programe (*drivere*) za komunikaciju sa Waspmoteom, no ako sustav to ne uspije napraviti automatski, korisnik će to trebati napraviti ručno. Upute za oba navedena koraka dostupne su na službenim stranicama proizvođača Waspmote uređaja (https://development.libelium.com/waspmote-ide-v06/).

Prije spajanja uređaja na računalo potrebno je na uređaj **povezati**:

- bateriju koja se koristi za napajanje NB-IoT modula
- NB-Iot modul sa pripadajućim antenama i SIM karticom
- Events Sensor Board v30 na koju se spaja senzor za osvjetljenje

Nakon toga na uređaju se pomakne prekidač za **paljenje** te se preko mini-USB konektora uređaj **povezuje** sa računalom.

Tada u instaliranom Waspmote IDE-u treba otvoriti priloženi kod za korištenje i načiniti neke **izmjene**:

- u polje **`char apn[] = "", char login[] = ""`** te **`char password[] = ""`** upisati podatke mobilnog operatora čija se SIM kartica koristi za spajanje na mrežu,
- u polje **`char network_operator[] = ""`** upisati ime operatora,
- u polje **`char host[] = "" i uint16_t port = 80`** navesti adresu i vrata (*port*) poslužitelja na koji se šalju podatci,
- kod funkcije **`snprintf(resource, sizeof(resource),`**

"/api/measurement/waspmote/add?dev_id=%s&luminosity=%lu", "Waspmote NB", Events.getLuxes(INDOOR)) zamijeniti redom podcrtane dijelove koda sa:

- putanjom do poslužitelja na kojoj će se primati podaci,
- identifikacijskom oznakom uređaja,
- podatkom da li osvjetljenje očitavamo u zatvorenom ili na otvorenom (INDOOR/OUTDOOR)

Nakon toga, na panelu pri vrhu IDE-a potrebno je odabrati opciju *„Tools"* te tamo podesiti uređaj na koji se učitava kod („*Board*") i *„port"* na koji je uređaj spojen.
Uređaj je sada spreman, te preko tipke „*Upload"* učitavamo kod na uređaj. Nakon uspješnog učitavanja kod se automatski počinje izvršavati, a tijek izvršavanja koda možemo gledati preko opcije „*Serial monitor*" dostupne u izborniku „*Tools*".

Na kraju korištenja, uređaj treba odspojiti sa računala i ugasiti preko prekidača na samom uređaju.

Inicijalizacija uređaja Pycom sa senzorima temperature i vlage zraka:

Prije spajanja uređaja na računalo **obavezno** je potrebno na uređaj povezati:

- LoRa antenu
- Pysense modul sa senzorima za temperaturu i vlagu zraka

Nakon toga potrebno je pratiti sljedeće korake:

1. Skinuti Visual Studio Code sa sljedeće stranice: https://code.visualstudio.com/
2. U programu Visual Studio Code (VSC) preuzeti dodatak Pymakr.
3. Preko USB konektora spojiti uređaj na računalo.
4. VSC bi trebao automatski prepoznati uređaj, ali ako dođe do problema, referencirati se na Pycom dokumentaciju (https://docs.pycom.io/).
5. Koristeći sljedeće naredbe, dohvatiti Device EUI koji treba kopirati na The Things Network (TTN):

```
from network import LoRa
import binascii
print(binascii.hexlify(LoRa().mac()).upper())
```

6. U programu VSC navigirati u direktorij preddiplprojekt_2020_pametnapoljoprivreda/Software/Pycom.
7. Nakon konfiguracije The Things Networka, u datoteci main.py promijeniti vrijednosti polja *dev_addr*, *nwk_swkey* i app_swkey na vrijednosti dobivene na TTN stranici.
8. Desnim klikom na VSC odabrati opciju „Pymakr > Upload Project".

Upute za instalaciju Android i iOS aplikacije:

Za iOS:
1. Instalirati ovisnosti sa naredom "pod install", u slučaju da Cocoapods nisu instalirane na računalu, instalirati ih preko naredbe "sudo gem install cocoapods".
2. U direktoriju PametnaPoljoprivreda otvoriti PametnaPoljoprivreda.xcworkspace datoteku.
3. U Xcodeu u klasi Constants.swift postaviti baseUrl od servera.
4. Sa cmd+R pokrenuti aplikaciju.

Za Android:
1. Otvoriti aplikaciju u Android Studiu.
2. Povezati uređaj sa računalom ili instalirati novi virtualni uređaj pomoću AVD managera.
3. U Android Studiu u klasi Constants postaviti baseUrl od servera.
4. Klikom na "Run app" ili a shift + F10 pokrenuti aplikaciju.

Detaljne upute kako se koristiti samim aplikacijama mogu se pronaći u posebnim poglavljima za iOS i Android aplikaciju.

10. Zaključak

Ovaj projekt radili smo u periodu od tri mjeseca. U prvom dijelu projekta koji je trajao do početka prosinca smo se upoznavali s tematikom projekta i tehnologijama koje ćemo koristiti. U tom periodu smo se također podijelili na četiri razvojna tima kako bi mogli što efikasnije razviti ovakav sustav Interneta Stvari. U drugom dijelu smo se počeli fokusirati na sami razvoj aplikacija i ostvarivanje komunikacije sa svim dijelovima sustava.
Glavni problemi s kojima smo se susretali tijekom projekta bili su povezani s nedostatkom iskustva i znanja. Veliki broj članova se po prvi put susretao s tehnologijama koje smo trebali koristiti i mnogima je ovo bio jedan od prvih veći projekata.

Dodatne probleme stvarala je činjenica da smo se rijetko mogli uživo sastajati zbog globalne pandemije koja je trajala u periodu ovog projekta. Samo se manji broj članova uspio nekoliko puta sastati uživo, ali većina komunikacije odvijala se preko interneta koristeći aplikacije *WhatsApp Messenger* i *Microsoft Teams*.

Unatoč tome, svi članovi su marljivo radili na projektu i to je razlog zbog kojeg smo uspjeli implementirati sve funkcionalnosti koje smo definirali još na početku projekta.

Važno je napomenuti da ovaj rezultat ne bi bio moguć bez pomoći prof. Kušeka i svih asistenata koji su nam bili na raspolaganju kad god nam nešto nije bilo jasno ili nam je trebala pomoć.

Na kraju možemo reći da smo definitivno zadovoljni rezultatom koji smo postigli u ovom kratkom periodu. Naučili smo neke važne pouke vezane uz rad u timu i stekli nova tehnička znanja koja će nam sigurno pomoći u budućnosti.

11. 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 Corutines docs: https://kotlinlang.org/docs/reference/coroutines-overview.html

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

Pycom dokumentacija: https://docs.pycom.io/

The Things Network dokumentacija: https://www.thethingsnetwork.org/docs/

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

Waspmote development Wiki https://development.libelium.com/waspmote/

Waspmote Quickstart Guide http://www.libelium.com/downloads/documentation/v12/quickstart_guide.pdf

Spring Boot: https://spring.io/projects/spring-boot

PostegreSQL: https://www.postgresql.org/

LoRaWAN: https://lora-alliance.org/about-lorawan/

Narrowband IoT: https://www.gsma.com/iot/narrow-band-internet-of-things-nb-iot/

Swift: https://developer.apple.com/swift/

Visual Studio Code: https://code.visualstudio.com/