







**Sadržaj**

[**1.**](#_gjdgxs) **Uvod 3**

[**2.**](#_30j0zll) **Opis rješenja 3**

[**3.**](#_1fob9te) **IoT platforma 4**

[**4.**](#_3znysh7) **Korisničke aplikacije 4**

# Uvod

SeizmoStat je IoT rješenje koje detektira potrese koristeći senzore i tehnologiju povezanih IoT-uređaja. SeizmoStat kontinuirano nadzire seizmičke aktivnosti i pruža trenutne informacije o intenzitetu potresa. Ovo rješenje bi se integriralo u zgrade kako bi automatski detektiralo i procijenio intenzitet potresa te aktivirao alarm upozorenja. SeizmoStat omogućava brzu detekciju potresa, omogućujući trenutne reakcije i smanjenje štete.

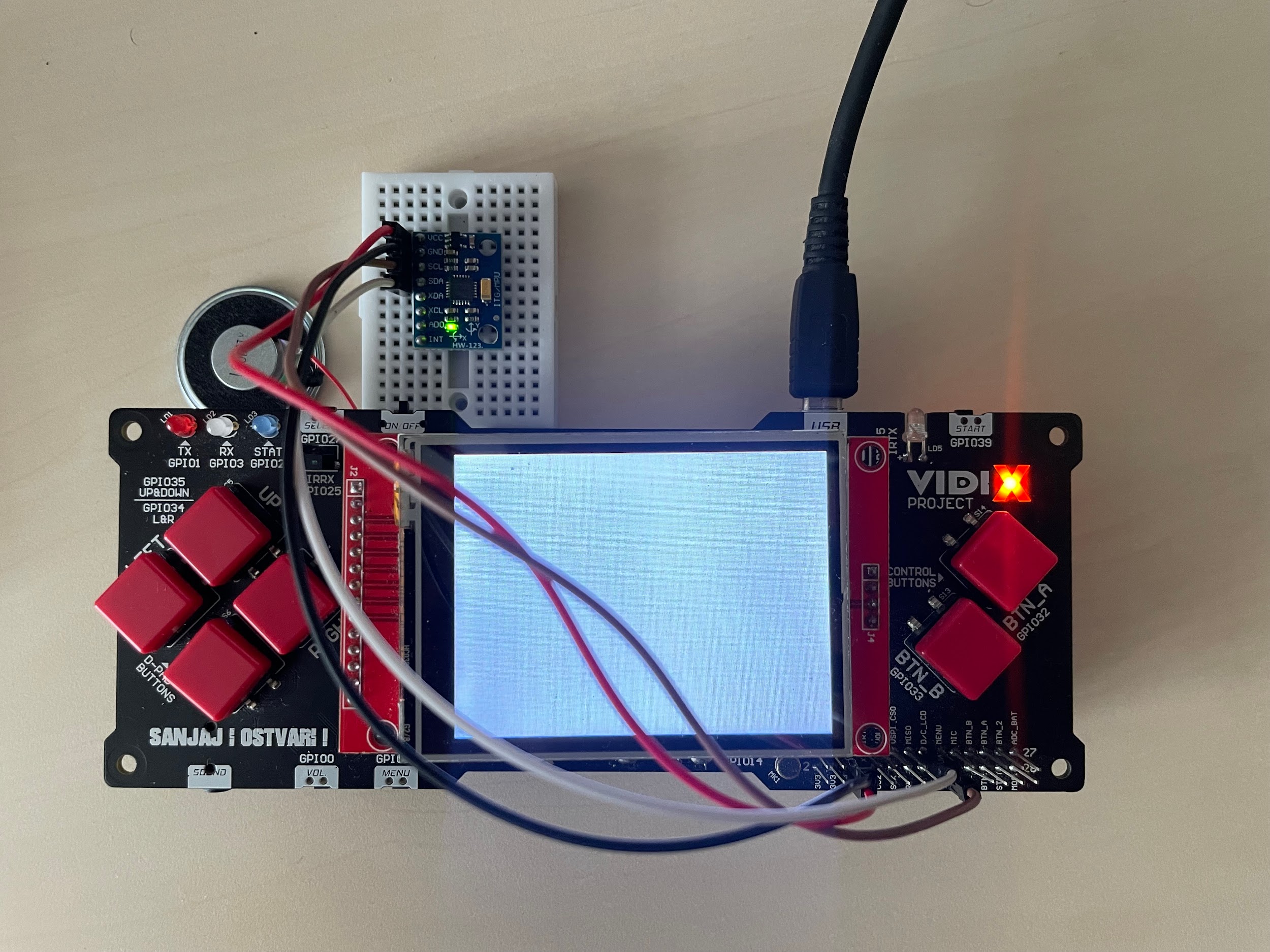
Rješenje se sastoji od senzorskog čvora, Data JediX IoT platforme, Web korisničke aplikacije i mobilne korisničke aplikacije. Senzorski čvor sastoji se od ESP32-WROVER-B mikrokontrolera, MPU6050 akcelerometra u ulozi senzora i zujalice za upozoravanje u ulozi aktuatora. Virtualni senzori ostvareni su simuliranjem sklopovlja u simulatoru Wokwi.

Razvijeno rješenje se može koristiti u analizi seizmičke aktivnosti, promatranju na daljinu, odgovoru na katastrofu, osiguravanju kritične infrastrukture, itd. Također se rješenje može koristiti za evakuaciju zgrade dok su očitana mjerenja još mala, ali dovoljno velika da se može smatrati potencijalno opasnom seizmičkom aktivnosti.

Implementacija projekta slična je projektu "[IoT Based Eearthquake Detection by THINGSPEAK](https://acadpubl.eu/hub/2018-119-16/2/320.pdf)" koji se koristi za detekciju ranog podrhtavanja i upozorenje dolaska razornih potresa.

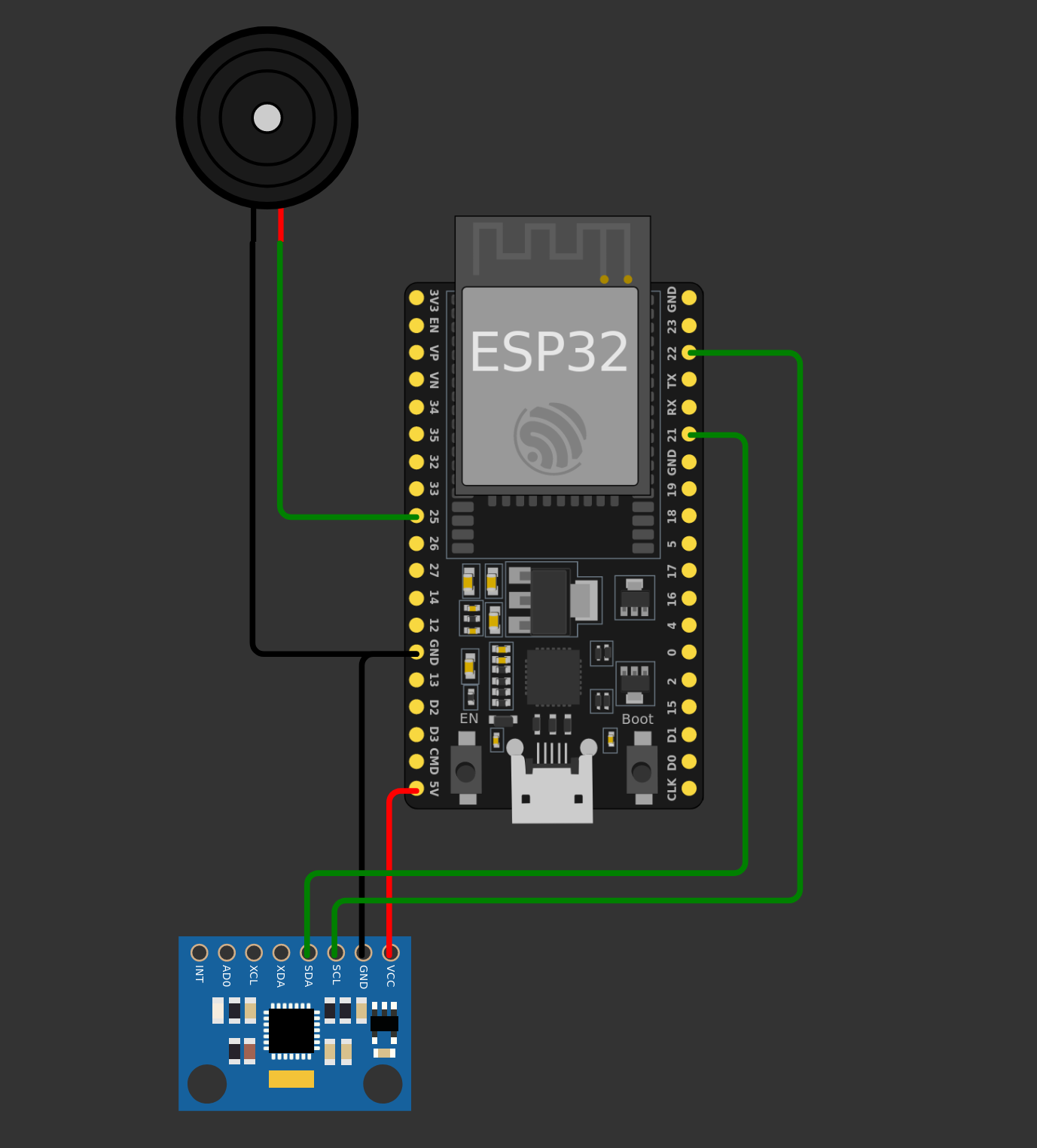
# Opis rješenja

U rješenju su korištene dvije vrste senzorskih čvorova - fizički i virtualni. Fizički senzor sastoji se od VIDI X pločice koja koristi ESP32-WROVER-B mikrokontroler na kojeg je putem eksperimentalne pločice spojen Adafruit MPU6050 senzor sa troosnim akcelerometrom te zujalica (buzzer), koja ima ulogu aktuatora. Komponente su napajane od strane prijenosnog računala korištenjem USB kabla. Ova konfiguracija prikazana je na *Slici 1*.



*Slika 1.* Fizički senzorski čvora

Virtualni senzorski čvorovi simulirani su koristeći Wokwi simulator sklopovlja (<https://wokwi.com/>). U ovim čvorovima koristi se ESP32-DevKit-C4 mikrokontroler na kojeg su spojeni IMU senzor s troosnim akcelerometrom te buzzer. Shema spoja prikazana je na *Slici 2*.

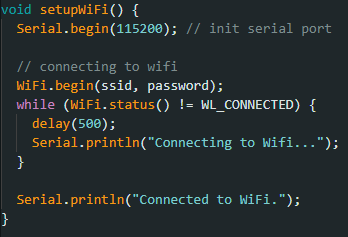


*Slika 2.* Shema virtualnog senzorskog čvora

Softver za senzorski čvor pisan je u C++ programskom jeziku koristeći ArduinoIDE razvojnu okolinu, pri čemu su korištene sljedeće knjižnice:

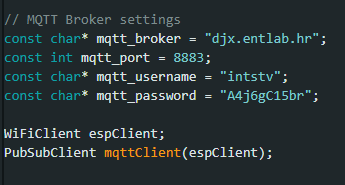
* WiFi.h: omogućuje WiFi povezivanje
* PubSubClient.h: MQTT klijent za ESP32
* Adafruit\_MPU6050.h, Adafruit\_Sensor.h, Wire.h: knjižnice za rad s MPU6050 senzorom
* cmath: matematičke funkcije za obradu podataka
* ArduinoJson.h: knjižnica za rad s JSON objektima.

Kako bi se omogućila komunikacija s MQTT brokerom te slanje podataka prikupljenih senzorom, senzorski čvor povezuje se na WiFi mrežu korištenjem WiFi klijenta. Proces povezivanja na WiFi traje dok uređaj nije u povezan na mrežu. *Slika 3* prikazuje isječak koda za povezivanje na WiFi mrežu.

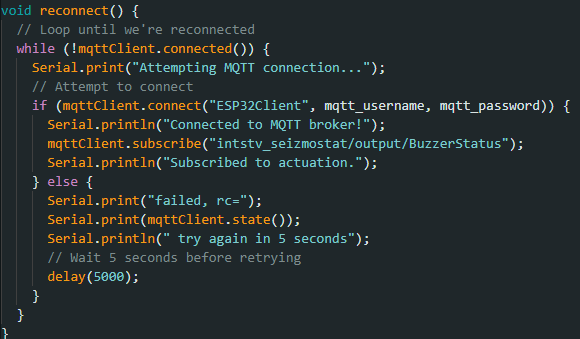


*Slika 3.* WiFi konfiguracija

Komunikacija između senzorskih čvorova i DataJediX platforme ostvarena je putem MQTT brokera, na kojeg se uređaj spaja korištenjem PubSubClient klijenta (*Slika 4*, *Slika 5*).



*Slika 4.* Postavke MQTT brokera



*Slika 5.* Povezivanje na MQTT broker

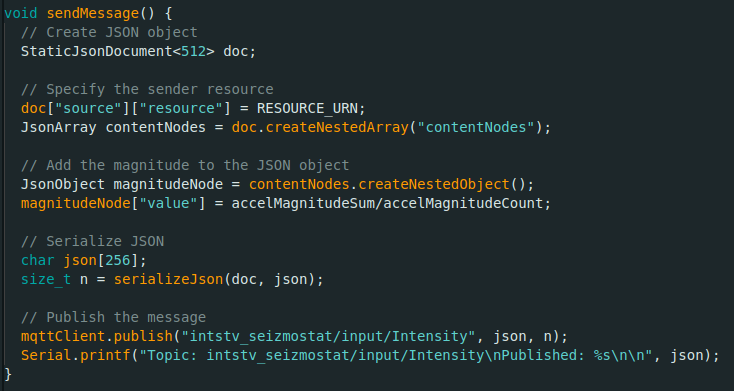
U slučaju prestanka konekcije na MQTT broker, uređaj se pokušava ponovno spojiti dok veza nije uspostavljena. Nakon uspješnog povezivanja, uređaj se pretplaćuje na MQTT poruke za aktuaciju.

Na *Slici 6* prikazan je programski kod glavne petlje u kojoj čitamo akceleraciju s MPU6050 senzora, računamo prosjek tijekom 1 sekunde koji onda u funkciji sendMessage() tu vrijednost zajedno s identifikatorom resursa koji šaljemo platformi preko MQTT protokola.



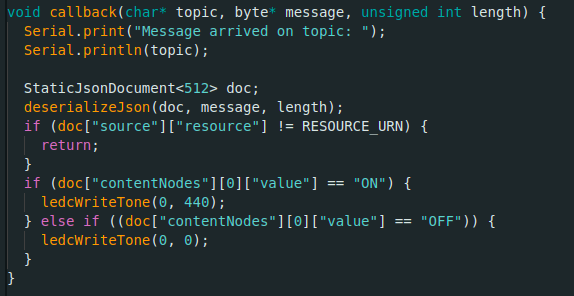
*Slika 6.* Glavna petlja

Stvaranje JSON poruke i njezina objava na MQTT topic intstv\_seizmostat/input/Intensity s kojeg platforma prima podatke je prikazana na *Slici 7*. Pritom je korištena ArduinoJson biblioteka za lakše koja omogućuje jednostavno formatiranje poruke JSON tipa.

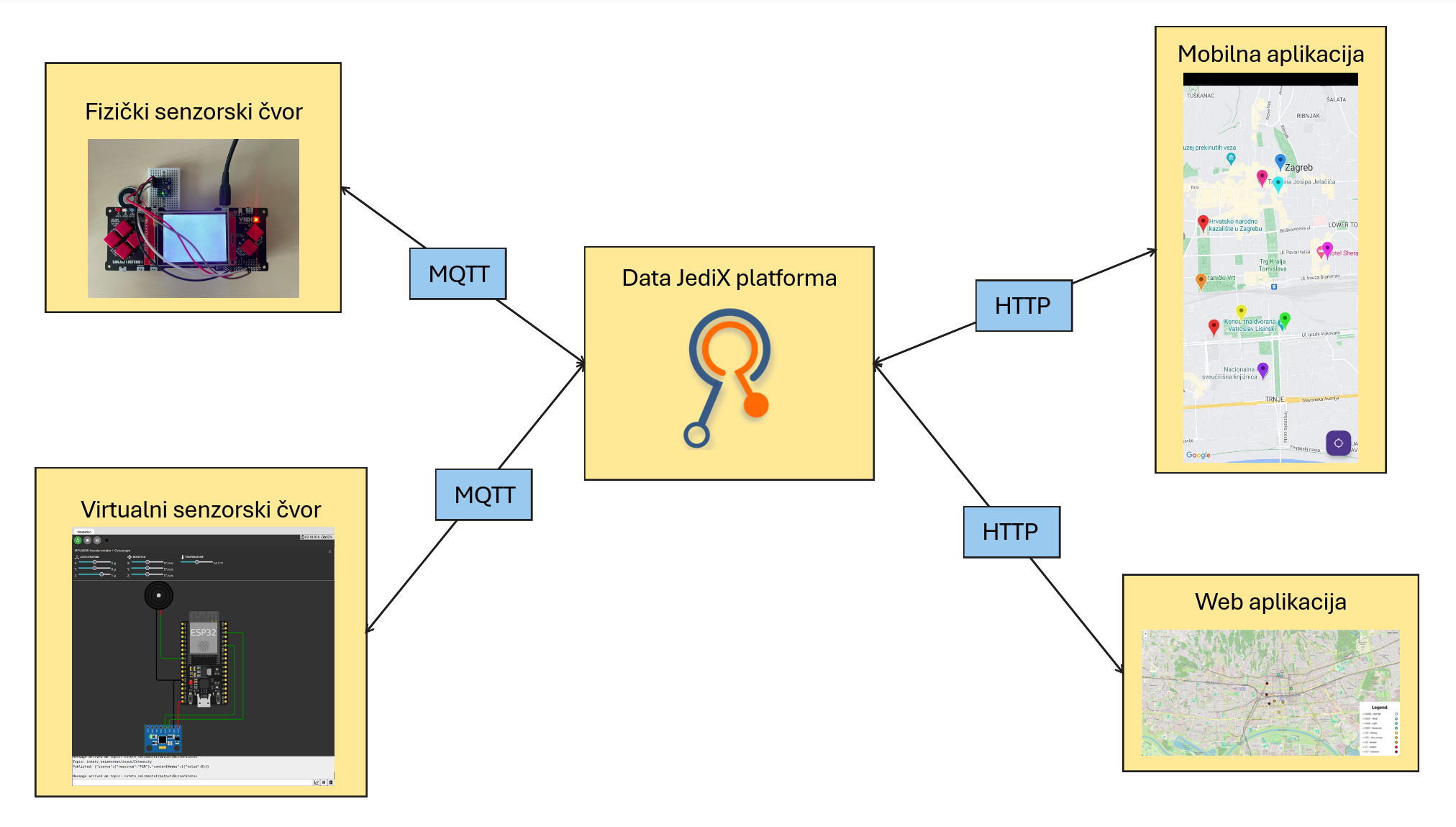
****

*Slika 7.* Slanje MQTT poruke s očitanjem

Aktuacija, tj. uključivanje buzzera kada podrhtavanje tla prijeđe zadani prag je ostvareno je u callback funkciji (*Slika 8.*) koja se poziva kada MQTT klijent primi poruku na topic za kojeg je pretplaćen. Ta poruka je u JSON formatu koji se parsira, provjerava se je li uređaj koji je primio poruku ciljani resurs za poduzeti akciju te zatim uključuje ili isključuje buzzer koristeći funkciju ledcWriteTone().

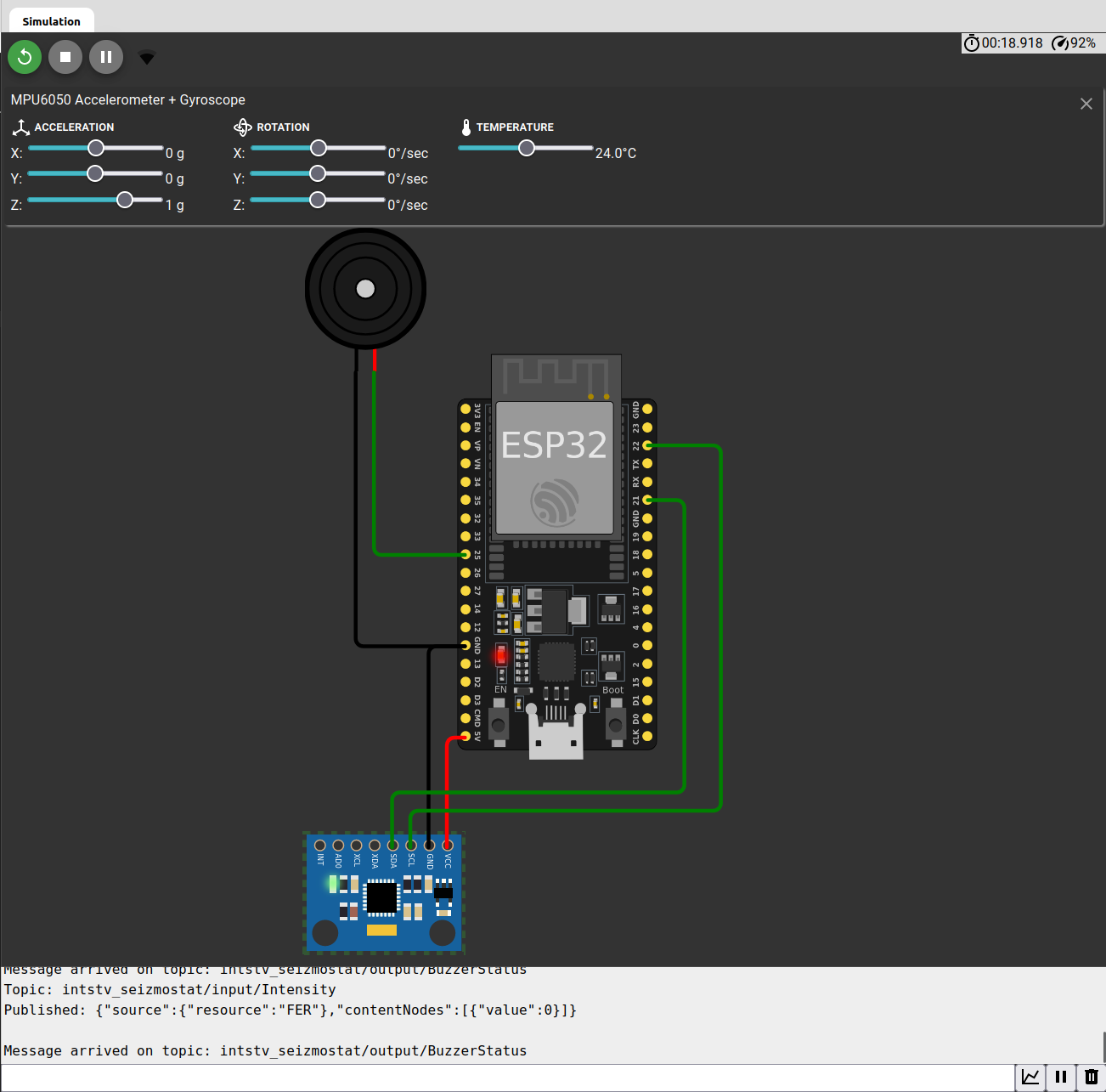


*Slika 8.* Callback funkcija koja uključuje i isključuje buzzer



*Slika 9.* Skica rješenja

Za detekciju podrhtavanja tla korišten je MPU6050 akcelerometar koji može mjeriti akceleraciju na x,y,z osima te komunicira s ESP32 mikrokontrolerom preko I2C sučelja, a za aktuaciju je korištena zujalica (buzzer) čija se frekvencija zvuka može mijenjati PWM-om. Senzorske vrijednosti simulirane su koristeći simulator sklopovlja Wokwi u kojem korisnik može za vrijeme izvođenja kliknuti na MPU6050 senzor i mijenjati vrijednosti koje očitava kao što je prikazano na *Slici 10*.

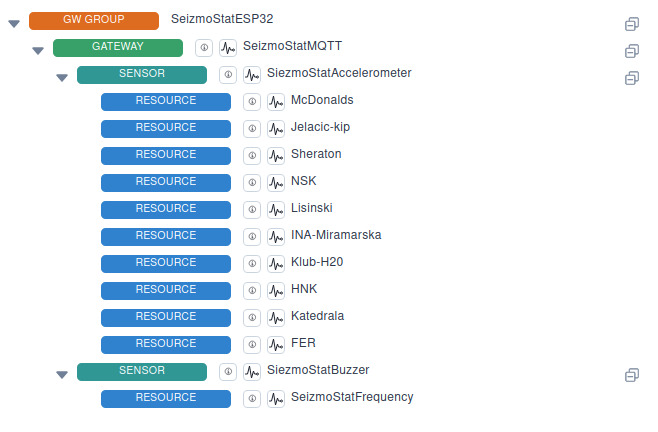


*Slika 10.* Virtualni senzor ostvaren pomoću simulacije

Senzorski čvor komunicira s Data JediX platformom pomoću WiFi-ja i MQTT protokola, a aplikacije s platformom preko HTTP-a. Za jednostavniji pregled svih poslanih poruka na MQTT brokeru korišten je MQTT Explorer. Koristili smo i Postman za testiranje slanja HTTP upita prema platformi, a uz Postman smo koristili i curl.

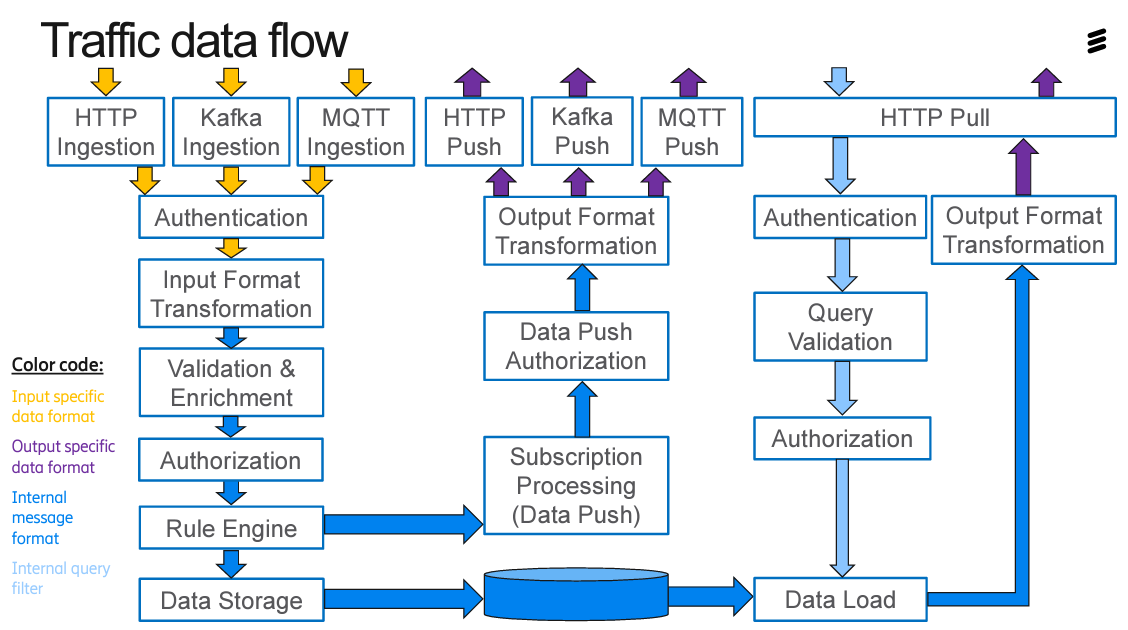
# IoT platforma

U sklopu rješenja projekta koristila se Data JediX platforma koju je razvila te i dalje razvija tvrtka Ericsson Nikola Tesla. Osnovna shema toka podataka na IoT platformi (*Slika 11.*) podržava mnoštvo ulaza (HTTP, Kafka, MQTT) među kojima se projektni tim primarno fokusirao na MQTT Ingestion (za očitanja) i HTTP Push (za aplikacije). Naravno, pritom se ne izostavlja dolazni HTTP promet kojim se pristupalo osnovnim funkcionalnostima platforme te njenom sučelju. Ericsson Nikola Tesla je projektni tim opskrbio s vjerodajnicama za autorizaciju na platformi što je omogućilo konfiguraciju putem HTTP zahtjeva i web sučelja (<https://djx.entlab.hr/ui>).



*Slika 11.* Tree view entiteta

Platforma podržava četiri različitih vrsta uređaja. Vrste uređaja su organizirane u hijerarhiju od četiri razine. Najviša razina je Grupa Pristupnika (engl. *Gateway Group*), dok su njena djeca uređaji vrste Pristupnik (engl. *Gateway*). Pristupnik sadržava uređaje vrste Senzor (engl. *Sensor*), dok Senzor može sadržavati Resurse (engl. *Resource*). Konfiguracija uređaja na DataJediX platformi za projekt Seizmostat vidljiva je na slici (*Slika 11.*).

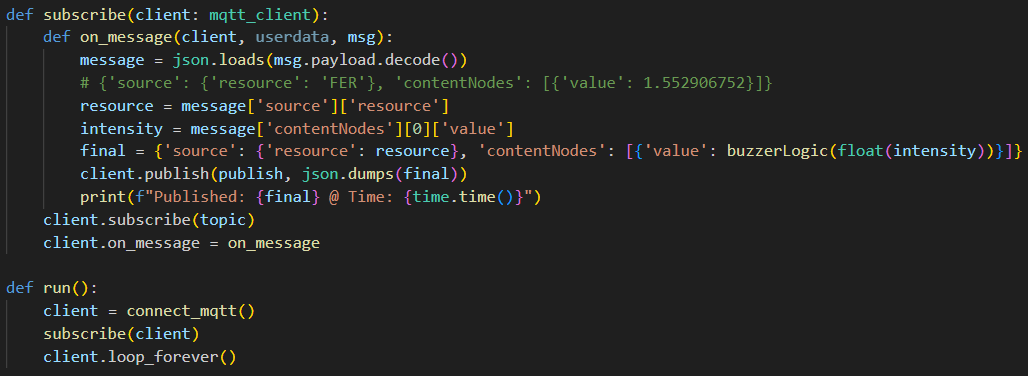


*Slika 12.* Shema toka podataka Data JediX platforme

Nadalje, među početnim konceptima realizacije aktuacije na projektu bilo je korištenje Rule Engine komponente na platformi u vidu uvjetovanja objavljenog (engl. *published*) MQTT sadržaja. Rule Engine komponenta uspostavlja logiku čitanja i utjecanja na senzorska očitanja takozvanim okidačima (engl. *trigger*) i učincima (engl. *effect*). Prilikom dolaska poruke na Rule Engine komponentu sadržaj očitanja se uspoređuje s aktivnim okidačima temeljem vremena, vrijednosti, postojanja i/ili logičke istinitosti te uzrokuje aktivaciju učinaka odgovarajućeg pravila. Učinak pravila može postavljati vrijednosti očitanja nekog resursa, odnosno senzora, i/ili objavljivati obavijest u obliku tekstualne poruke. U pravilu, izlaz Rule Engine komponente odnosno poruka generirana učinkom pravila pojavljuje se kao dolazni promet (unatoč dijagramu toka na *Slici 12.*) te zadržava originalnu poruku, koja je izazvala okidanje okidača, u generiranoj poruci. Za generirane poruke koje su namijenjene ostvarivanju aktuacije u sustavu stvorena je dodatna pretplata (engl. *subscription*) s odredištem zujalice (engl. *buzzer*) kojoj bi se propagirala uvjetovana vrijednost te čijim bi se dolaskom zujalica palila ili gasila ovisno o sadržaju. Za svaki je projektni tim bila osigurana specifična tema za pretplatu na očitanja senzora. Taj skup funkcionalnosti koje omeđuju Rule Engine i Subscription Processing komponente toka podataka završavaju autorizacijom guranja sadržaja na stog za MQTT Push.

Nažalost, tijekom konfiguracije pravila došlo je do tehničkih poteškoća koje je bilo teško reproducirati, a još teže rektificirati – validacija okidača i učinaka pravila bila bi uspješna, no validacija pri pohrani samog pravila bi rezultirala nepoznatom pogreškom koju razvojni inženjeri iz Ericssona Nikole Tesle nisu uspjeli reproducirati na svom sustavu. Povrh toga, nakon što se pohranjivanje pravila konačno ostvarilo, vrijednosti koje bi rezultirale učincima bile bi nepredvidive, te su to najčešće bile *null* vrijednosti ili neuspjeh u generiranju poruke, a izmjenom bilo koje stavke u pretplati u Subscription Processing komponenti bi se onemogućio rad pretplate na neko vrijeme zbog primjene novih parametara, što je dodatno otežalo proces debugiranja.

U cilju skraćivanja muke i timu i osoblju zaduženom za timove, aktuacija je alternativno realizirana koristeći paho-mqtt Python modul u obliku skripte (*Slika 13.*).



*Slika 13.* Isječak Python skripte - pretplata na temu te uvjetovano objavljivanje za svako očitanje

Logika ostvarena Python skriptom, ujedno i MQTT klijentom (varijabla client) se pretplaćuje na očitanja senzora (varijabla topic), funkcijom buzzerLogic uspoređuje varijablu intensity s globalno deklariranom varijablom threshold koja u trenutku pisanja izvještaja iznosi 0,12. Drugim riječima, kada vrijedi intensity > threshold, na temu na koju je pretplaćena zujalica (varijabla publish) se šalje MQTT u dolaznom formatu s vrijednosti ‘ON’ (*Slika 14.*). U protivnom šalje vrijednost ‘OFF’.



*Slika 14.* Primjer ispisa tijekom rada skripte

Ovim putem se očito stvara dodatni promet u protoku podataka do aktuatora, no dalo bi se argumentirati i da generiranje te preusmjeravanje poruke natrag na ulazno sučelje putem učinaka pravila također stvara dodatni promet, ali takva analiza je ipak van dosega zadatka projekta.

Od oblika naplate odnosno komercijalizacije rješenja zadatka projekta razmatrala se prodaja gotovog rješenja rizičnim lokacijama, pretplatničkog sustava dostave senzorskih očitanja, komercijalne analitike urbanog planiranja i suradnja s obrtima poput građevinskih obrta, osiguravajućih kuća, zavoda za prostorno planiranje, agencijama za procjenu rizika itd.

# Korisničke aplikacije

* 1. **Mobilna aplikacija**

Za razvoj mobilne aplikacije korišten je Android Studio s jezikom Java, koristeći gradle building tool. Aplikacija se sastoji od tri dijela, prikaza dostupnih senzora, prikaza senzora na mapi i graf očitanja vrijednosti sa senzora. Kako bi se mogli slati zahtjevi i pratiti lokacija potrebno je u AndoidManifest.xml dodati sljedeće 3 ovlasti:

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />

Za prikaz mape koristi se Googleov API za čiju uporabu, kao i za druge Google aktivnosti, Android Studio nudi automatsko ukomponiranje. Također su bili potrebni i dodatni importovi u build.gradle.kts za:

* okhttp - slanje zahtjeva (prva 3)
* graphview - prikazivanje podataka na grafu (zadnji)

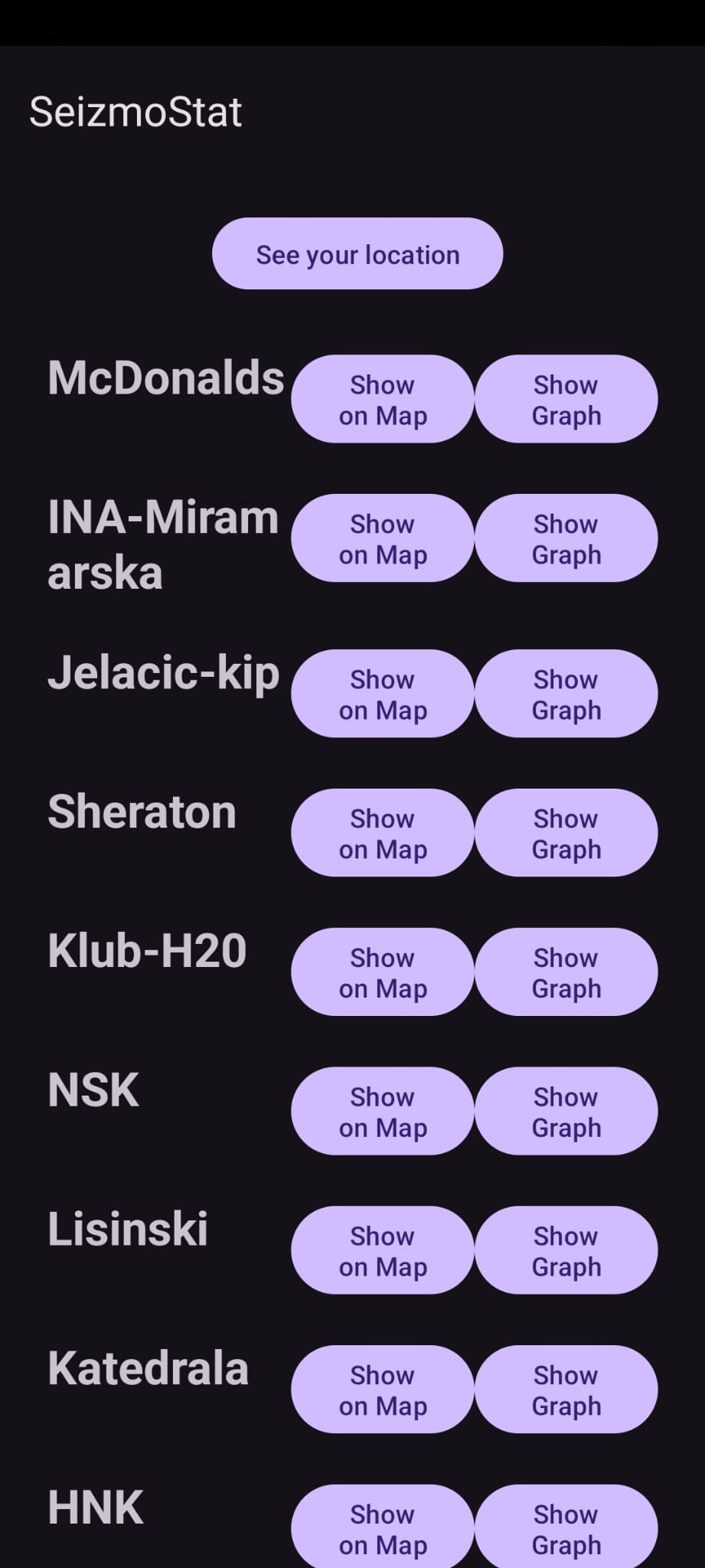
*implementation*(platform("com.squareup.okhttp3:okhttp-bom:4.12.0"))

*implementation*("com.squareup.okhttp3:okhttp")

*implementation*("com.squareup.okhttp3:logging-interceptor")

*implementation*("com.jjoe64:graphview:4.2.2")

Na prvom dijelu su prikazani prijavljeni senzori na platformu s nazivom i mogućnošću otvaranja lokacije na mapi i grafa koji prikazuje mjerenja na senzoru, a može se i otvoriti lokacija korisnika na mapi. Prikaz je dan sljedećom slikom (*Slika 15.)*:



*Slika 15.* Prikaz početnog zaslona mobilne aplikacije

Na drugom prikazu je mapa. Implementiran je Google Maps API. Za ispravan rad ovog djela potrebno je uključiti lokaciju. Prati se live lokacija korisnika kao i zadnje mjerenje na svakom senzoru. Ovisno o jačini zadnjeg mjerenja mijenja se boja markera koji označava lokaciju senzora. Također postoji i neutralna boja koja označava da mjerenja još nije bilo s tog senzora. Kako boja ovisi o detektiranoj vrijednosti dano je sljedećim kodom:

if (measure < 0.0005) {

color = BitmapDescriptorFactory.HUE\_CYAN;

} else if (measure < 0.003) {

color = BitmapDescriptorFactory.HUE\_CYAN;

} else if (measure < 0.028) {

color = BitmapDescriptorFactory.HUE\_ROSE;

} else if (measure < 0.062) {

color = BitmapDescriptorFactory.HUE\_MAGENTA;

} else if (measure < 0.12) {

color = BitmapDescriptorFactory.HUE\_VIOLET;

} else if (measure < 0.22) {

color = BitmapDescriptorFactory.HUE\_GREEN;

} else if (measure < 0.40) {

color = BitmapDescriptorFactory.HUE\_YELLOW;

} else if (measure < 0.75) {

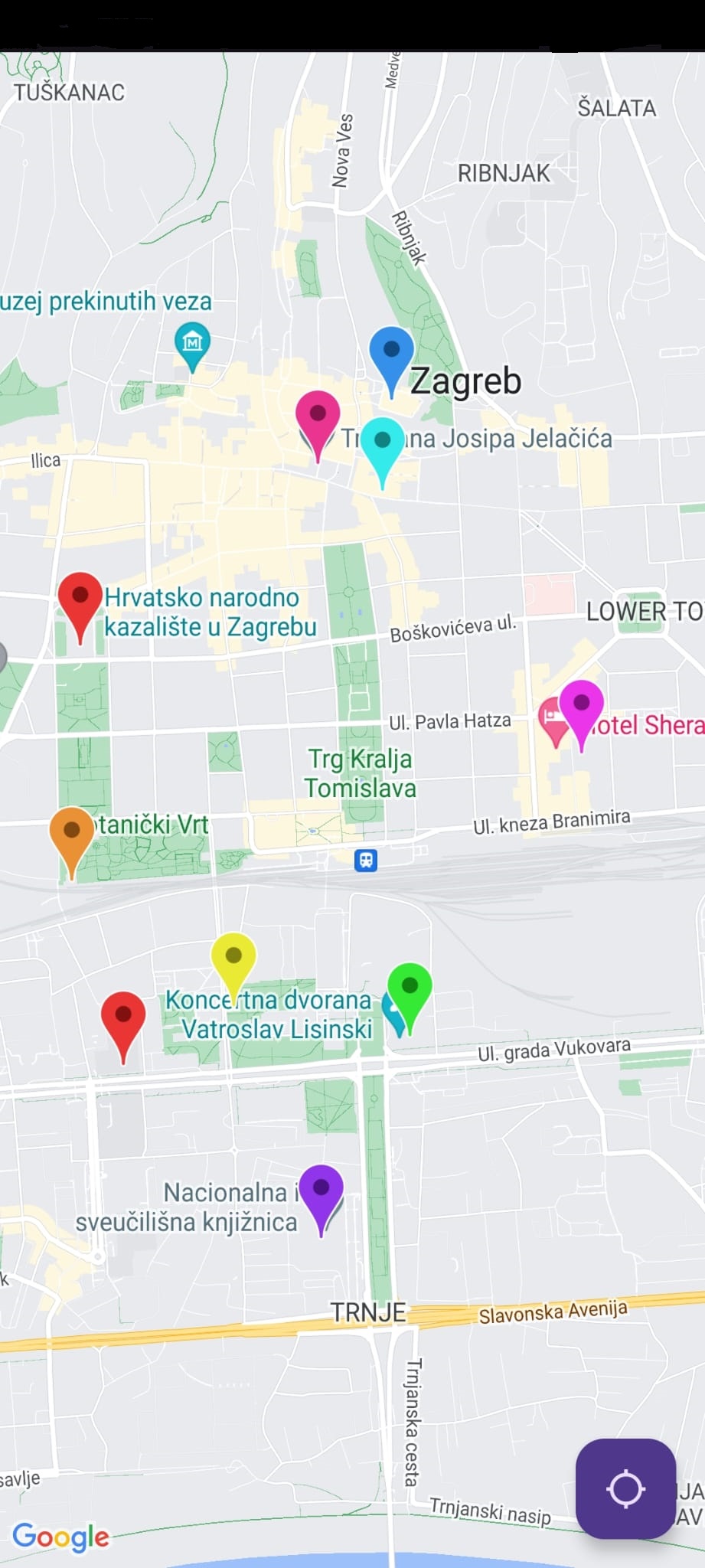
color = BitmapDescriptorFactory.HUE\_ORANGE;

} else {

color = BitmapDescriptorFactory.HUE\_RED;

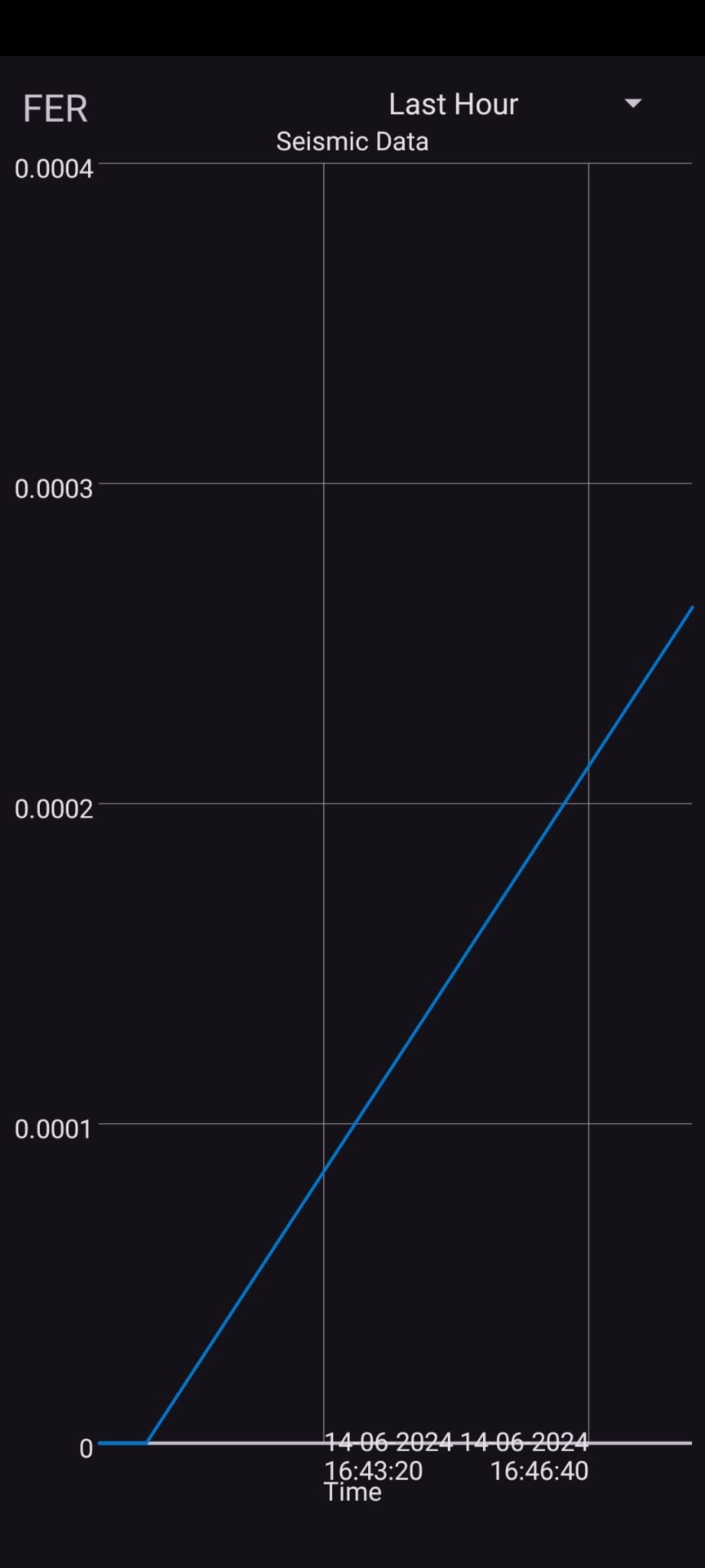
}

Nakon pritiska na određeni marker centrira se na njega te se pokaže i njegovo ime. Također nakon dva uzastopna pritiska na isti marker nudi se mogućnost otvaranja grafa za dotični marker. Prikaz mape kao i lokacije markera i korisnika dano je na sljedećoj slici (*Slika 16*.):



*Slika 16.* Prikaz zaslona mape koja prikazuje lokaciju markera s odgovarajućim bojama

Zadnji prikaz je grafički prikaz podataka koji su mjereni u povijesti. Po defaultu je postavljeno na zadnjih sat vremena, a moguće je i postaviti na zadnjih 12 sati, zadnji dan i sva prijašnja mjerenja. Graf je moguće i zumirati kako bi se dio mjerenja mogao preciznije vidjeti.



*Slika 17.* Prikaz zaslona grafa određenog senzora na FERu

Podaci se dohvaćaju pomoći HTTP zahtjeva prema platformi prema sljedećem principu:

String url = "https://djx.entlab.hr/m2m/data?res=" + device1.getDeviceId();

Request request = new Request.Builder()

.url(url)

.addHeader("Authorization", credentials)

.addHeader("Accept", "application/vnd.ericsson.m2m.output+json;version=1.1")

.build();

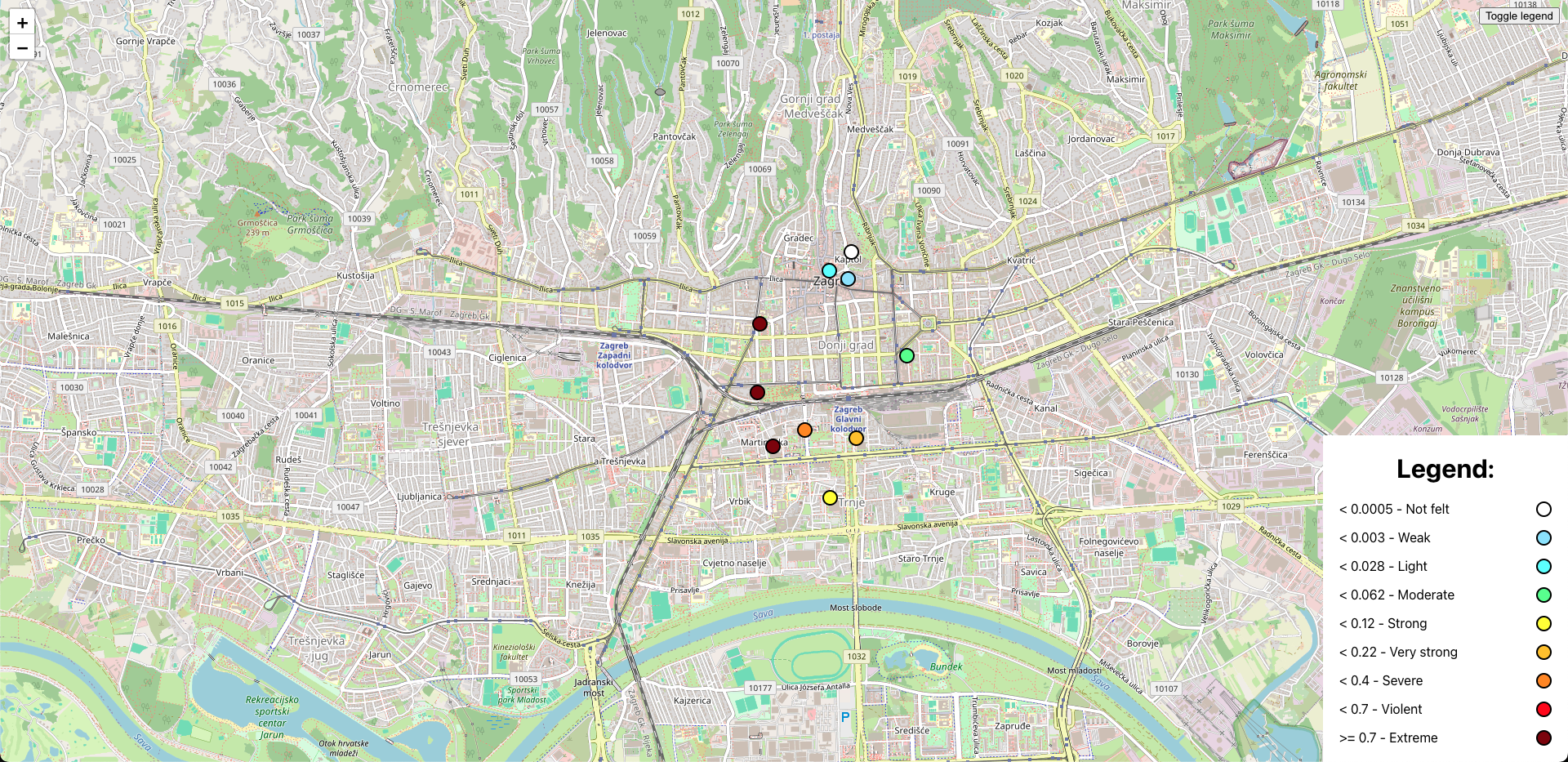
client.newCall(request).enqueue(new Callback() {

Važno je napomenuti kako je potrebno definirati određeni oblik JSON-a pošto se ne koristi predefinirani u upitima prema DataJediX platformi. Slanjem zahtjeva na različite endpointove se dohvaćaju relevantni podaci potrebni za izradu aplikacije i dohvaćanje podataka. Za autorizaciju se koristi Basic Auth.

* 1. **Web aplikacija**

Za razvoj web aplikacije koristila se React knjižnica za izradu aplikacija i programski jezik TypeScript. Aplikacija prikazuje kartu na kojoj se nalaze točke koje prikazuju senzore, a klikom na senzor prikazuje se popup sa grafičkim prikazom očitanja pojedinog senzora. Pored karte se također nalazi legenda iz koje korisnik može očitati jačinu potresa (*Slika 18.*).

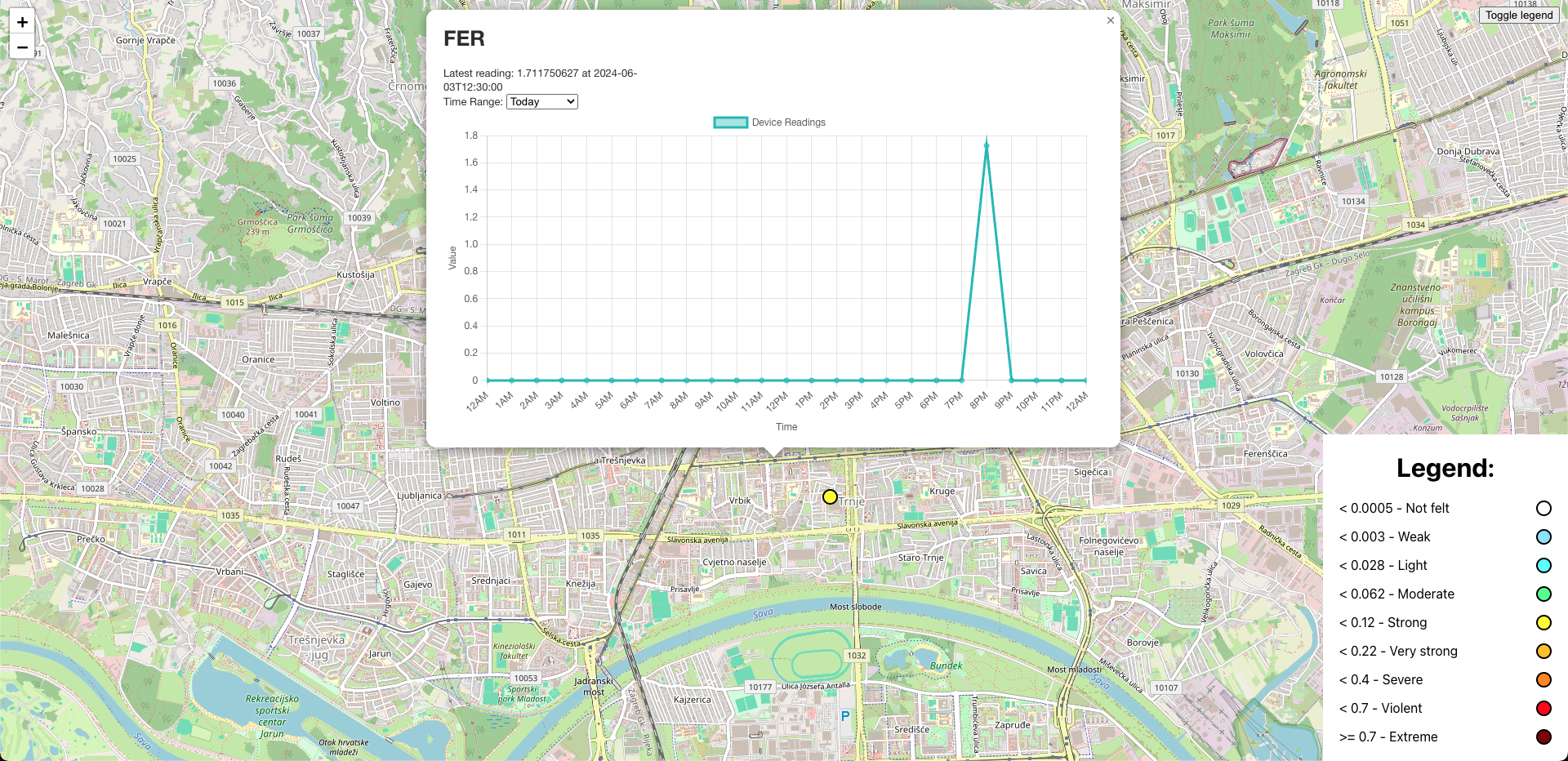
Za prikaz karte koristi se *leaflet* paket i OpenStreetMap karta. Senzori se prikazuju pomoću njihovih koordinata, a boje točaka ovise o zadnjoj vrijednosti koju je senzor primio. Imena svih senzora dobivamo slanjem GET upita na https://djx.entlab.hr/m2m/provisioning/sensor/SiezmoStatAccelerometer/resource, njegovu lokaciju slanjem GET upita na https://djx.entlab.hr/m2m/provisioning/resource/${deviceId}/attribute, a očitanja na https://djx.entlab.hr/m2m/data?res=${deviceId}.



*Slika 18.* Karta s prikazom senzora

Klikom na senzor otvara se popup sa povijesnim prikazom očitanja tog senzora (*Slika 19.*). Ime senzora je prikazano na vrhu, a ispod njega vrijednost i vrijeme posljednjeg očitanja. Korisnik može birati koliko daleko unazad želi pregledati podatke te zoomirati pomoću kotačića na mišu.

Za prikaz povijesnih podataka koristi se *chart.js* paket. Ovisno o odabranom rasponu vremena (Today, Last week, Last month ili All time) tablica prikazuje vrijednosti u drugačijim periodima (sat, dan, tjedan ili godina). Ukoliko u periodu nije bilo očitanja vrijednost je postavljena na 0.



*Slika 19.* Detalji senzora