Sadržaj

Uvod

Ubrzanim razvojem Interneta i proizvodnjom velikog broja povezanih uređaja omogućen je razvoj koncepta Interneta stvari. Glavna ideja ovog koncepta je povezivanje uređaja koji omogućuju različita senzorska opažanja ili izvedbu određenih aktuatorskih radnji, s njihovim korisnicima putem javne internetske mreže.

Sustavi temeljeni na konceptu Interneta stvari imaju široku domenu primjene. Jedna od domena primjene su pametne zgrade i pametni dom. Uvođenjem koncepta interneta stvari u domove mogu se automatizirati svakodnevne radnje i uređaji. U sklopu ovog rada razvijen je sustav temeljen na konceptu Interneta stvari u domeni pametnog doma za automatizaciju provjere tko je pozvonio na pametno zvono.

Zadatak ovog završnog rada je povezivanje pametnog zvona s Android aplikacijom. Pametno zvono se sastoji od tipke i web-kamere koji su spojeni na mrežu te je prilikom zvonjenja potrebno poslati obavijest o tome da je netko pozvonio na Android aplikaciju. Otvaranjem primljene obavijesti korisniku se treba prikazati tko je zvonio, a to je potrebno ostvariti tako da se Android aplikacija poveže s postavljenom web-kamerom i dohvati trenutnu sliku kamere.

Za ostvarivanje opisanog scenarija koristit će se različite tehnologije kako bi se ostvarile sve tražene funkcionalnosti i kako bi se u potpunosti implementirala cjelokupna raspodijeljena arhitektura kakvu podrazumijevaju rješenja iz domene Interneta stvari.

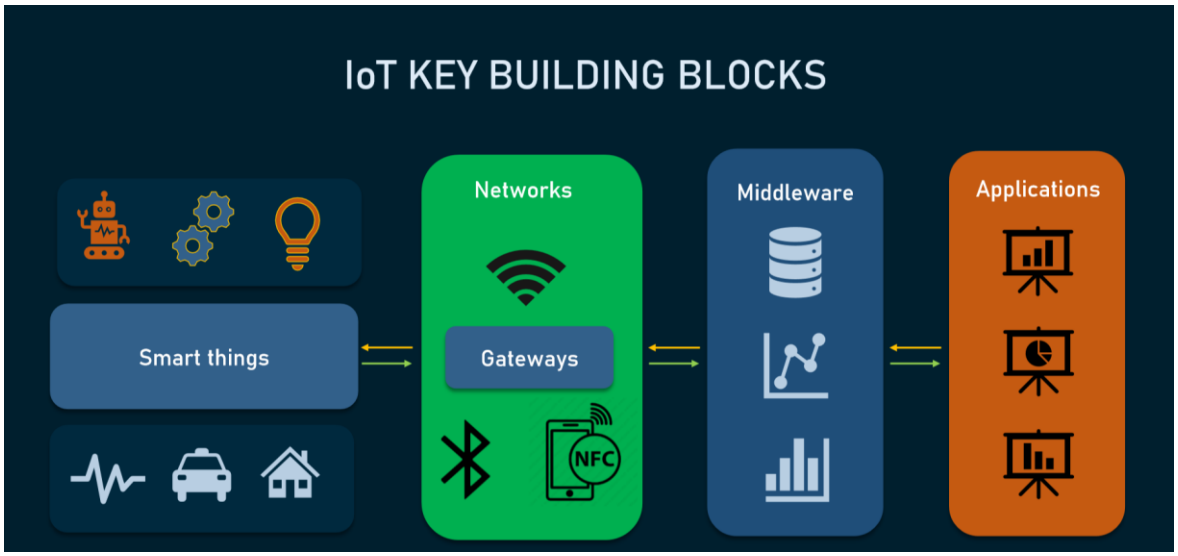
1. Internet stvari

Internet stvari (engl. *Internet of Things*) predstavlja mrežu stvari, odnosno fizičkih uređaja sa senzorima, *software*-om i drugim tehnologijama koje služe u svrhu njihovog međusobnog povezivanja i razmjene podataka, te povezivanja s krajnjim korisnicima kroz internetsku mrežu. Tehnologija Interneta stvari postaje jedna od najvažnijih tehnologija danas kada možemo svakodnevne predmete povezati na Internet i komunicirati s njima te ih upravljati. [1]

Razvoju Interneta stvari doprinio je pristup jeftinim senzorskim tehnologijama s niskom potrošnjom električne energije. Jednostavni internetski protokoli su olakšali učinkovit prijenos podataka od senzorskih čvorova ograničenih računalnih resursa do poslužitelja, a razvoj umjetne inteligencije za prepoznavanje govora približio je digitalne osobne pomoćnike korištenju u kućanstvu. [1]

1.1. Arhitektura usluga Internata stvari

Pojednostavljen pogleda na arhitektu usluga Interneta stvari sastojao bi se od tri najvažnije komponente a to su uređaji, mrežni prilaz (engl. *gateway*) i usluge u „oblaku“. Uređaji su stvari koje su povezane na mrežu, a to su najčešće neke vrste senzorskih uređaja koje koriste razne protokole za komunikaciju s mrežnim prilazom. Mrežni prilaz pruža funkcionalnosti pred procesiranja podataka i osiguravanje povezanosti s „oblakom“, najčešće korištenjem *websocket*-a. Sloj mrežnog prilaza bi također trebao dati zajednički pogled na uređaje što drugim slojevima sustava omogućava lakše upravljanje uređajima. Treći sloj arhitekture usluga Interneta svari je sloj usluge u „oblaku“. Usluga u „oblaku“ koristi baze podataka za spremanje senzorskih vrijednosti poslanih s povezanih uređaja te omogućava izvođenje brojnih akcija na osnovu zaprimljenih vrijednosti. [30]



Slika 1.1 Skica arhitekture usluga Interneta stvari [31]

1.2. Primjer usluga Interneta stvari

Usluge Interneta stvari mogu se primijeniti u gotovo svim područjima života kao što su pametni gradovi, pametne kuće, briga za starije i nemoćne, medicina i zdravstvo, strojna proizvodnja, poljoprivredna proizvodnja i slično. Ljudima najbliža primjena je ona u pametnim domovima. Usluga Interneta stvari može se primijeniti na koncepte za automatizaciju doma što može uključivati pametna svijetla kojima se upravlja putem Interneta, upravljanje grijanjem, sigurnosni sustav kamera te niz drugih pametnih uređaja. [30]

Osim za izgradnju pametnih domova, Internet stvari sve veću primjenu nalazi i rješenjima za pametne gradove. Tako se pojavljuju projekti pametne ulične rasvjete, pametnog rukovanja komunalnim otpadom, pomoć za pronalazak slobodnog parkirnog mjesta i slično. U takvim projektima primjenjuju se principi i tehnologije koncepta Interneta stvari za optimizaciju iskorištenja energije i podizanje kvalitete življenja u gradovima.

2. Opis studijskog slučaja

U sklopu ovog rada razvijen je sustav s pametnim zvonom koji se povezuje na aplikaciju za operacijski sustav Android te preko web-kamere prikazuje tko je pozvonio. Sustav se sastoji od sljedećih komponenti:

- pametno zvono sačinjeno do *Arduino Yun* pločice i web-kamere,
- instanca *Home Assistant* platforme,
- poslužitelj za komunikaciju s *Home Assistant* platformom,
- *Firebase Cloud Messaging* servis i
- Android aplikacija za prikaz slike s web-kamere i primanje obavijesti o zvonjenju

2.1. Funkcionalni zahtjevi

Pametno zvono treba omogućiti:

- obradu događaje pritiska tipke na zvonu,
- slanje *MQTT* poruke instanci *Home Assistant*-a,
- slanje slike trenutnog prikaza web-kamere *HTTP*-om do konkretnih pametnih uređaja.

Home Assistant treba omogućiti:

- pretplaćivanje na *MQTT* temu i primanje poruka objavljenih na toj temi,
- slanje *HTTP POST* zahtjeva s tijelom „*push*“ obavijesti na poslužitelj.

Poslužitelj treba omogućiti:

- primanje *HTTP POST* zahtjeva s tijelom i temom „*push*“ obavijesti,
- objavljivanje „*push*“ obavijesti na zadanu temu na *Firebase Cloud Messaging* servisu.

Android aplikacija treba omogućiti:

- registriranje pametnog zvona s web-kamerom,

- brisanje registriranog pametnog zvona,
- pretplaćivanje na primanje obavijesti o zvonjenju,
- objava preplate na primanje obavijesti o zvonjenju,
- dohvaćanje slike web-kamere.

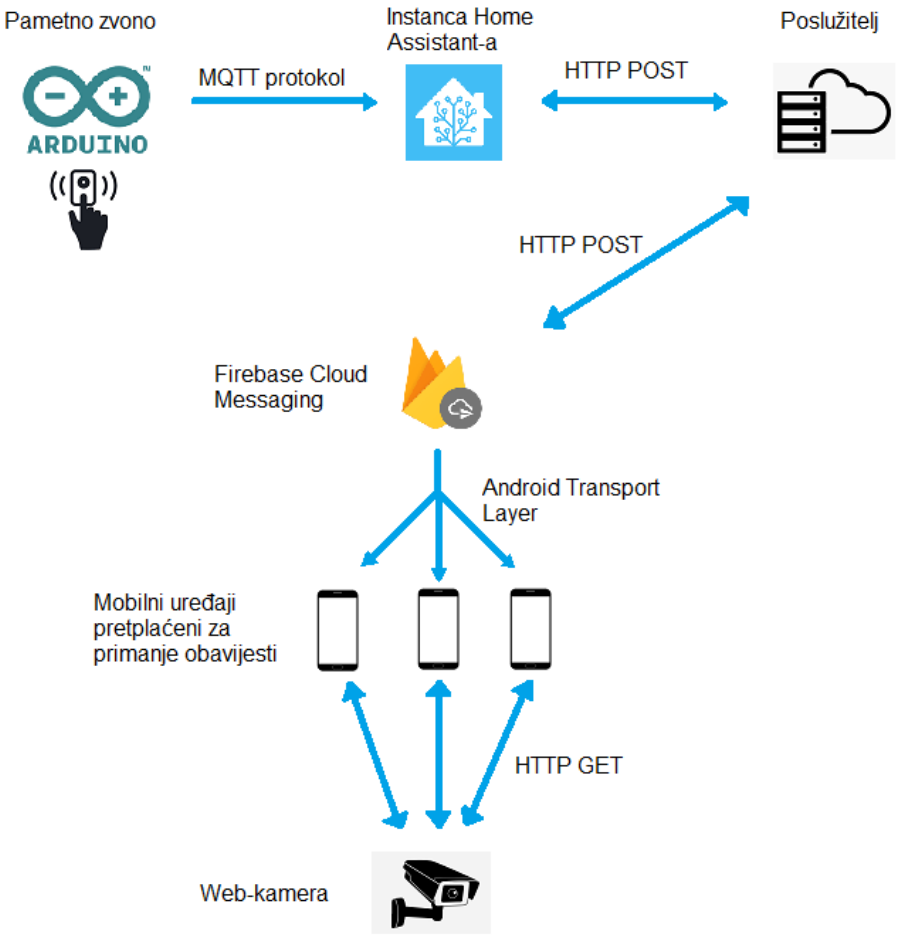
2.2. Arhitektura sustava

Razvijeni sustav se sastoji od više komponenata koje međusobno komuniciraju različitim protokolima putem interneta. Pametno zvono nakon pritiska na tipku uspostavlja komunikaciju s instancom *Home Assistant*-a objavljivanjem poruke putem *MQTT* protokola. *Home Assistant* je potrebno konfigurirati tako da bude putem *MQTT* brokera preplaćen na određenu temu kroz koju će primati nove poruke.

Zaprimanjem poruke o zvonjenju *Home Assistant* mora obavijestiti poslužitelja o tom događaju. Komunikacija između *Home Assistant*-a i poslužitelja odvija se putem *HTTP* protokola. *Home Assistant* šalje *POST* zahtjev poslužitelju s tijelom poruke koju ćemo prikazati korisniku putem „*push*“ obavijesti te temu pomoću koje će *Firebase Cloud Messaging* servis znati koje uređaje mora obavijestiti o zvonjenju.

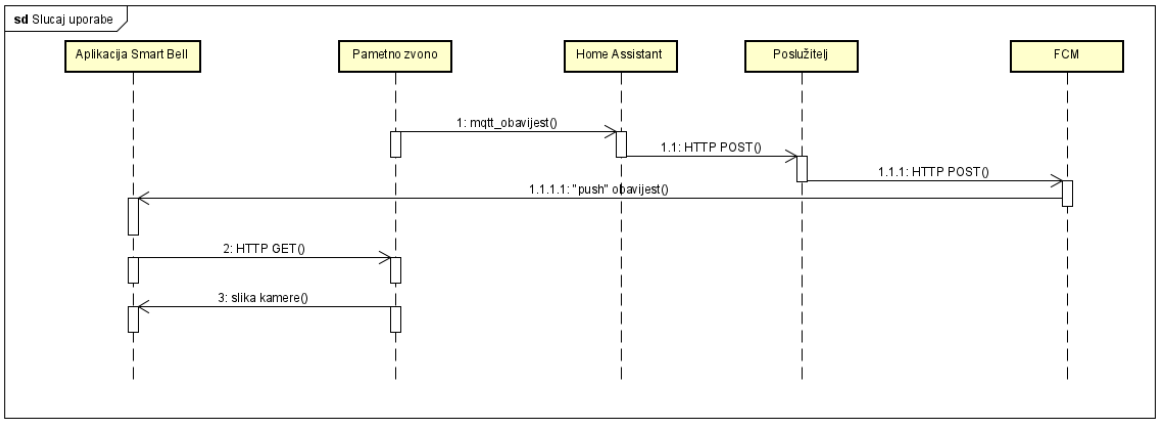
Obavijesti o zvonjenju na *Firebase Cloud Messaging* servis se šalje s poslužitelja *HTTP* protokolom slanjem *POST* zahtjeva. Iz poslanih parametara *Firebase Cloud Messaging* servis dalje šalje obavijesti na konkretne uređaje korištenjem *Android Transport Layer*-a. [2][3]

Primitkom obavijesti o zvonjenju, Android aplikacija *HTTP* protokolom dohvaća trenutnu sliku kamere. Aplikacija šalje *GET* zahtjev web-kameri te kao odgovor dobiva sliku trenutnog stanja.



Slika 2.1 Arhitektura sustava

2.3. Opis slučaja uporabe



Slika 2.2 Sekvencijski dijagram slučaja uporabe

Događaj zvonjenja počinje pritiskom na tipku zvona. Nakon pritiska na tipku zvona *Arduino* se povezuje na adresu *Home Assistant*-a te objavljuje *MQTT* poruku na specifičnu temu. Navedena radnja je u sekvencijskom dijagramu prikazana komponentom „1:mqtt_obavijest“. *Home Assistant* pretplaćen je na temu na koju *Arduino* objavljuje obavijest. Na primitak obavijesti o zvonjenju *Home Assistant* šalje *HTTP POST* zahtjev na poslužitelj. U tijelu *POST* zahtjeva postavlja se naslov i sadržaj obavijesti koja će se prikazati korisniku te tema na kojoj će se obavijest objaviti na *Firebase Cloud Messaging* servisu. Komponenta „1.1:*HTTP POST*“ na sekvencijskom dijagramu prikazuje opisani način komunikacije *Home Assistant*-a i poslužitelj. Poslužitelj nakon što primi *HTTP POST* zahtjev kreira obavijest na temelju dobivenih parametara. Kreiranu obavijest poslužitelj objavljuje na *Firebase Cloud Messaging* servis putem *HTTP POST* zahtjeva što na dijagramu prikazuje komponenta „1.1.1:*HTTP POST*“. *Firebase Cloud Messaging* servis zaprimljene poruke razašalje do uređaja koji su pretplaćeni na temu na kojoj je poruka objavljenja. Komponenta „1.1.1.1:*'push'* obavijest“ prikazuje slanje obavijesti na mobilni uređaj. Kada korisnik otvori pristiglu „*push*“ obavijest aplikacija šalje *HTTP GET* zahtjev na *link* web-kamere koja je postavljena kao dio pametnog zvona što pokazuje komponenta „2:*HTTP GET*“. Kao odgovor na *GET* zahtjev web-kamera šalje snimku trenutnog stanja pred kamerom, odnosno sliku onog tko je pozvonio. Slanje slike web-kamere prikazuje komponenta „3:slika kamere“.

3. Implementacija studijskog slučaja

3.1. Android aplikacija

U konceptu Interneta stvari raznim senzorskim tehnologijama se prate vrijednosti određenih parametara u odnosu na vrijednosti zabilježene senzorima. Kada zabilježene vrijednosti zadovolje prethodno zadane uvjete potrebno je učiniti određene akcije. Ovisno o domeni primjene Interneta stvari, vrijednosti koje se prate senzorima variraju. U slučaju pametnog doma te vrijednosti mogu opisivati temperaturu u sobi, vlažnost zraka sobe, kvaliteta zraka u sobi, ali i nalazi li se netko u sobi u kojoj je upaljeno svjetlo ili jesu li vrata hladnjaka korektno zatvorena. U navedenim situacijama poželjno je obavijestiti korisnika o trenutnom stanju mjerene vrijednosti, a u današnje vrijeme to je najefikasnije učiniti slanjem obavijesti na mobilni uređaj. Aplikacija *Smart Bell* razvijena je upravu u svrhu obavještavanja korisnika o tome da se dogodio neki događaj koji zahtjeva njegovu akciju, a to je događaj zvonjenja na ulazna vrata.

Aplikacija *Smart Bell* razvijena je za mobilne uređaje s operacijskim sustavom Android s osnovnom zadaćom povezivanja pametnog zvona s Android aplikacijom. Pametno zvono je povezano s aplikacijom *Smart Bell* putem „*push*“ obavijesti koje aplikacija prima kada netko pozvoni te na otvaranje pristigle obavijesti aplikacija dohvaća trenutnu sliku web-kamere pametnog zvona.

Primanje „*push*“ obavijesti ostvareno je putem *Firebase Cloud Messaging* servisa. Android aplikacija registrirana je na *Firebase* putem *Firebase Console* te su u aplikaciju ugrađene potrebne ovisnosti kako bi aplikacija mogla primati „*push*“ obavijesti. Za uspješno primanje obavijesti također je potrebno pretplatiti konkretni fizički uređaj, na kojemu je aplikacija instalirana, na određenu *Firebase Cloud Messaging* temu. Kada je uređaj pretplaćen na specifičnu temu, automatski će dobivati „*push*“ obavijesti o novim objavama u njoj.

Aplikacija je razvijena tako da svako pametno zvono predstavlja jednu temu za primanje obavijesti. Prilikom registracije zvona u aplikaciji potrebno je unijeti *URL* kamere na koji će se slati *HTTP GET* zahtjevi za dohvaćanje trenutne slika koju web-kamera zvona prikazuje.

3.1.1. Opis arhitekture aplikacije

Aplikacija *Smart Bell* razvijena je u domeni Interneta stvari zbog čega je za njen isparavan rad potrebno da mobilni uređaj na kojemu je ona instalirana bude stalno povezan na mrežu. Ako uređaj nije spojen na mrežu obavijesti o zvonjenju će stići tek kada se sljedeći put uspostavi veza s mrežom. Dohvaćanje slike s web-kamere također neće biti moguće ako uređaj nije spojen na mrežu.

Aplikacija za pamćenje registriranih pametnih zvona koristi *SQLite* bazu podataka, a za dohvaćanje i prikazivanje podataka koristi arhitekturu oblikovnog obrasca *Model-View-ViewModel* (*MVVM*). Prema *MVVM* obrascu podatci se odvajaju u *Model* komponentu u kojoj se dohvaćaju iz lokane baze podataka ili iz udaljene baze podataka putem Interneta. [19]

U Aplikaciji *Smart Bell* podatci se dohvaćaju iz lokane baze podataka u razred modela *CameraModel*. *CameraModel* predstavlja jedan entitet u bazi podataka. Članske varijable razreda *CameraModel* predstavljaju atribute entiteta u bazi podataka.

Komponenta *ViewModel* pruža podatke za specifičnu komponentu korisničkog sučelja kao što su fragmenti ili aktivnosti. *ViewModel* sadrži svu potrebnu logiku za rukovanje podatcima i komuniciranje s *Model*-om. Tip podataka *ViewModel* i razredi koji ga nasljeđuju mogu pozivati druge komponente za učitavanje podataka ili prosljeđivati korisnikove zahtjeve za izmjenu podataka. [19]

```
class CameraViewModel(private val databaseHandler: DatabaseHandler) : ViewModel() {

    val camerasList: MutableLiveData<ArrayList<CameraModel>> = MutableLiveData()

    fun getCameras() {
        camerasList.value = databaseHandler.getCameras()
    }
```

Slika 3.1 Dohvaćanje podataka u *ViewModel*

U razredu *CameraViewModel*, podatci dohvaćeni iz baze podataka u modele *CameraModel* predaju se kao vrijednost tipa podataka *MutableLiveData*. *MutableLiveData* nasljeđuje razred *LiveData* i omogućuje izmjenu vrijednosti podataka koje sadrži. [20]

Tip podataka *LiveData*, i izvedeni tipovi, omogućuje promatranje podataka koje sadrži, ali za razliku od standardnih tipova promatrača, *LiveData* prati životni ciklus komponenata

aplikacije kao što su aktivnosti, fragmenti ili servisi, što omogućuje obavještavanje promatrača koji su u aktivnom stanju životnog ciklusa. [21]

Komponenta *View* brine se o prezentacija podataka iz *ViewModel*-a. *View* promatra podatke koje *ViewModel* sadrži te na izmjenu podataka osvježava korisničko sučelje. Komponenata *View* također zaprima korisnikove zahtjeve za izmjenom podataka, ako je to moguće. Iz tog razloga komponente korisničkog grafičkog sučelja imaju promatrače koji prate kada je ta komponenta pritisnuta i pozivaju funkcije *ViewModel*-a koje će izmijeniti podatke na željeni način. [19]

Za prezentaciju liste podataka iz baze podataka aplikacija *Smart Bell* koristi *RecyclerView* koji omogućuje prezentaciju velikog skupa podataka na ograničenim veličinama zaslona mobilnog uređaja. Za isparavan rad *RecyclerView*-a potrebno mu je zadati *LayoutManager* i *Adapter*. [22]

LayoutManager će se brinuti o pozicioniranju objekata unutar *RecyclerView*-a i određivati kada objekti više nisu vidljivi, te se mogu „reciklirati“ za prikaz drugih objekata. [23]

Adapter se brine o povezivanju predanih podataka i grafičkih komponenata korisničkog sučelja koje *RecyclerView* prikazuje. [24]

```
val listOfCamerasImageView = findViewById<RecyclerView>(R.id.cameraRecycleView)
listOfCamerasImageView.layoutManager = LinearLayoutManager(applicationContext)
```

Slika 3.2 Dodjeljivanje *LayoutManager*-a

```
cameraListAdapter = CameraListAdapter(viewModel)
listOfCamerasImageView.adapter = cameraListAdapter
```

Slika 3.3 Dodjeljivanje *Adapter*-a

Za izgled grafičkih komponenata u *RecyclerView*-u zadužen je *ViewHolder*. Razred koji nasljeđuje razred *ViewHolder* zadaje se kao ugniježdena klasa razreda koji nasljeđuje razred *Adapter*. *ViewHolder* određuje koji će se podatak povezati s kojom komponentom korisničkog grafičkog sučelja. [25]

```
class CameraAdapter(cameraViewModel: CameraViewModel) : RecyclerView.Adapter<CameraAdapter.ViewHolder>() {

    private var cameraViewModel : CameraViewModel = cameraViewModel

    inner class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
```

Slika 3.4 Implementacija *Adapter*-a i *ViewHolder*-a

```
fun bind (camera: CameraModel) {
    cameraId?.text = camera.id.toString()
    subscribedSwitch?.text = camera.name
    val subscriptionStatus = subscribedSwitch?.isChecked
    if (subscriptionStatus != camera.subscribed) {
        subscribedSwitch?.isChecked = camera.subscribed
    }
    topicName?.text = camera.topic
}
```

Slika 3.5 Povezivanje podataka i komponenata grafičkog korisničkog sučelja

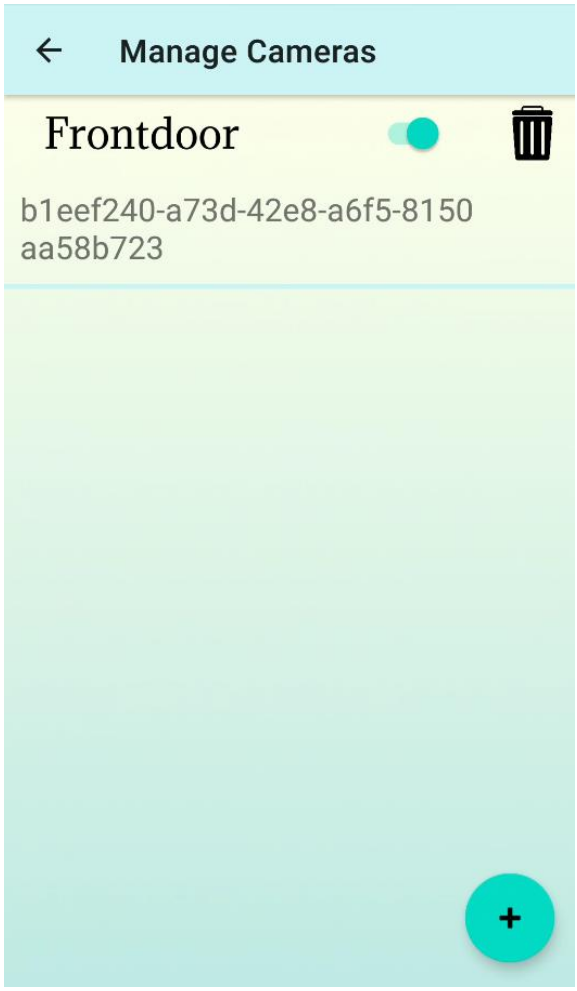
```
deleteCamera.setOnClickListener {  it: View!
    val id = cameraId.text.toString().toInt()
    val cameraModel = cameraViewModel.getCameraForId(id)
    if (cameraModel != null) {
        cameraViewModel.deleteCamera(cameraModel)
    }
}
```

Slika 3.6 Izmjena podataka u *ViewModel*-u na korisnički zahtjev

3.1.2. Opis glavnih funkcionalnosti i korištenih tehnologija

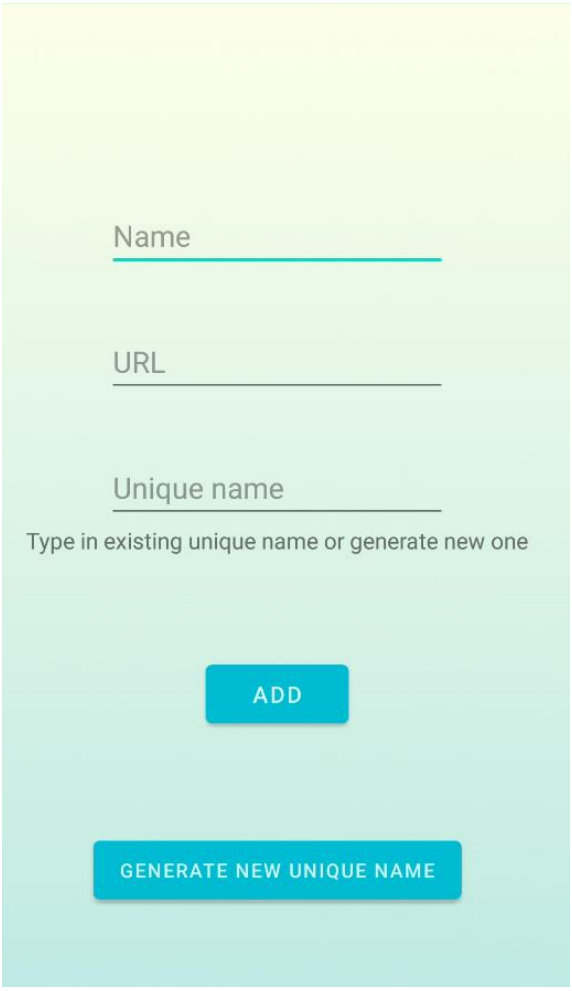
Otvaranjem aplikacije *Smart Bell* otvara se početni zaslon aplikacije s kojeg je moguće klikom na sliku kućice otvoriti pregled web-kamera registriranog pametnog zvona ili klikom na sliku kotačića prijeći u postavke registriranog pametnog zvona.

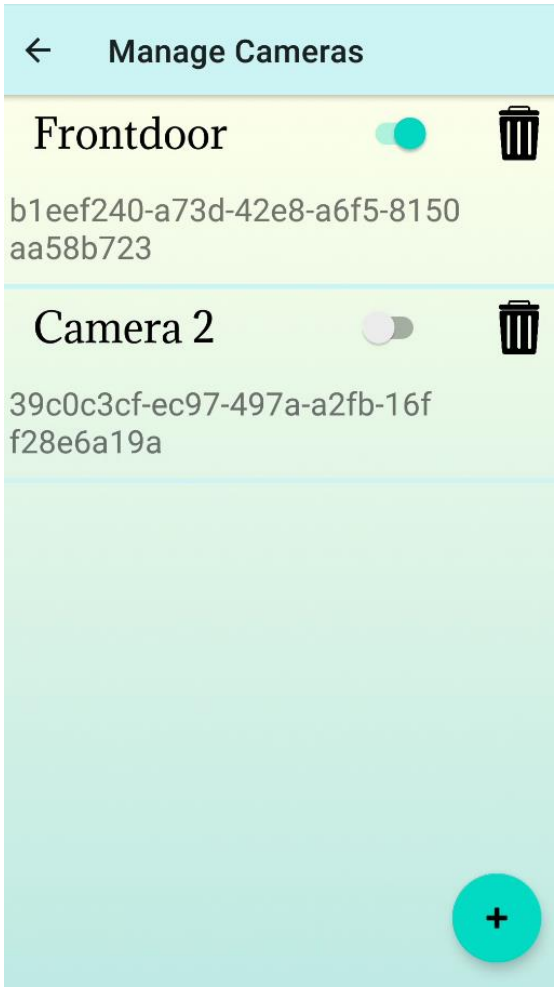
Slika 3.7 Početni zaslon aplikacije



Slika 3.8 Lista registriranih pametnih zvona

Prilikom prvog pokretanja aplikacije potrebno je otvoriti postavke registriranja pametnog zvona. U postavkama treba pritisnuti gumb sa znakom „+“ za dodavanje novog pametnog zvona. Pametnom zvonu treba dodijeliti naziv, upisati *URL* web-kamere koja prikazuje tko je pozvonio te *Firebase Cloud Messaging* temu na koju će pametno zvono objavljivati događaje zvonjenja. Radi sigurnosti, aplikacija nudi generiranje globalno jedinstvene teme, ali je moguće i upisivanje proizvoljnog naziva teme. Kada su uneseni svi potrebni podatci potrebno je stisnuti tipku „*ADD*“ koja će pohraniti zvono u bazu podataka aplikacije.



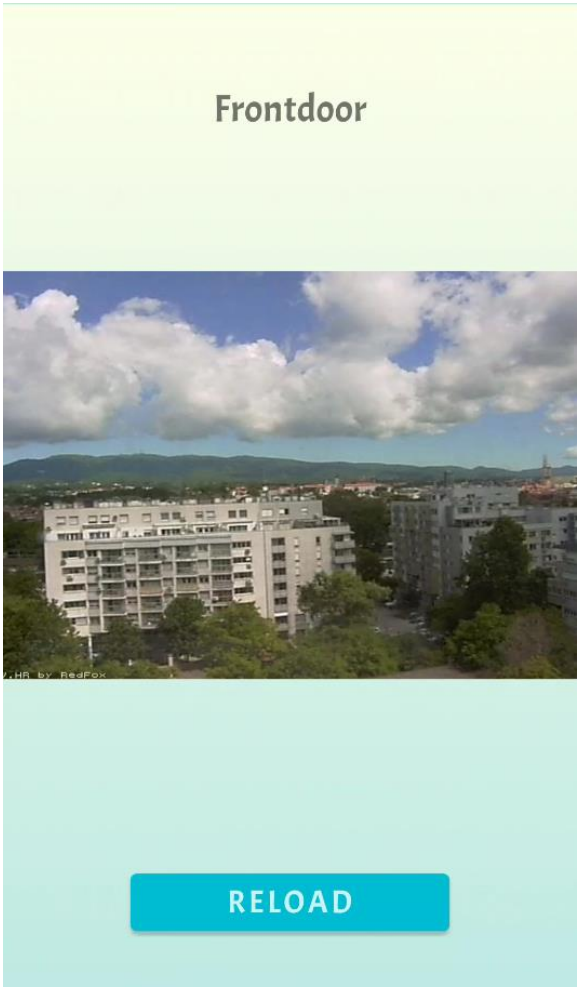


Slika 3.9 Dodavanje novog pametnog zvona

Slika 3.10 Prikaz dodanog novog zvona

Jednom dodano pametno zvono može se vidjeti u listi registriranih pametnih zvona. Mobilni uređaj je moguće pretplatiti na primanje obavijesti zvonjenja registriranog pametnog zvona ili odjaviti pretplatu na primanje obavijesti od pametnog zvona. Registrirano pametno zvono u svakom trenutku moguće je izbrisati i dodati novo zvono.

Kada je registrirano barem jedno pametno zvono moguće je otvoriti i prikaz slike s web-kamere zvona. Otvaranjem aktivnosti za pregled web-kamera otvara se lista koja prikazuje naziv zvona i trenutnu sliku koju aplikacija dohvaća s web-kamere te gumb za ponovno dohvaćanje slike.



Slika 3.11 Prikaz slike web-kamere

3.1.2.1 SQLite baza podataka

SQLite je relacijska baza podataka za pohranu podataka na mobilnom uređaju. Uređaji s operacijskim sustavom Android dolaze s ugrađenom implementacijom SQLite baze podataka. [26]

U aplikaciji *Smart Bell* baza podataka se koristi za zapisivanje vrijednosti varijabli koje su potrebne za ispravno prikazivanje naziva pametnog zvona, slike web-kamere pametnog zvona, teme na koju pametno zvono objavljuje obavijesti zvonjenja te za prikaz je li mobilni uređaj pretplaćen na obavijesti pojedinog zvona.

Baza podataka sastoji se od samo jednog entiteta *CameraTable* s atributima:

- *_id* – cijeli broj koji se automatski povećava za svaki novi zapis u bazi podataka
- *name* – naziv pametnog zvona
- *url* – *URL* web-kamere pametnog zvona

- *subscribed* – zastavica koja govori je li uređaj pretplaćen na obavijesti zvona
- *topic* – tema na koju zvono objavljuje obavijesti

Za rukovanje s SQLite bazom podataka potrebno je izraditi razred koja nasljeđuje razred *SQLiteOpenHelper*. *SQLiteOpenHelper* je apstraktni razred s apstraktnim metodama *onCreate(SQLiteDatabase)* i *onUpgrade(SQLiteDatabase, int, int)* koje je potrebno nadjačati u izvedenom razredu. Izvedeni razred brine se o otvaranju baze podataka, ako ona postoji, stvaranjem baze podataka ako ne postoji i nadogradnjom baze podataka ako je to potrebno. [27]

U razredu izvedenom iz klase *SQLiteOpenHelper* još se nalaze i funkcije za dodavanje, dohvaćanje i brisanje pametnog zvona iz baze podataka, funkcija za promjenu statusa pretplate na obavijesti zvona i funkciju koja vraća entitet *CameraModel* čiji *id* odgovara predanom *id*-u.

3.1.2.2 Firebase Cloud Messaging

Aplikacija *Smart Bell* razvijena je za primanje obavijesti od pametnog zvona. Obavijesti se šalju putem *Firebase Cloud Messaging* servisa. *Firebase Cloud Messaging* je dio *Firebase* platforme za razvoj mobilnih aplikacija. Aplikaciju je potrebno registrirati kao novi projekt putem *Firebase* konzole na poveznici: https://console.firebase.google.com/u/0/. U aplikaciju je potrebno dodati konfiguracijsku datoteku google-services.json koju je potrebno preuzeti za vrijeme registracije aplikacije na *Firebase* konzoli i potrebno je dodati ovisnosti za *Firebase Cloud Messaging* servis i *Google Messaging Servis*. [28]

Detaljnije informacije o dodavanju ovisnosti mogu se pronaći na poveznici: https://firebase.google.com/docs/cloud-messaging/android/client

Kada je *Firebase Cloud Messaging* ispravno dodan u projekt aplikacije, potrebno je dobadati novi razred koji će naslijediti *FirebaseMessagingService.* U tom razredu moguće je rukovati zadatcima koji se trebaju izvesti kada se zaprimi nova obavijest, ali za ostvarenje funkcionalnih zahtjeva sustava to neće biti potrebno. Potrebno je navedeni razred dodati u *AndroidManifest* kao novi *service* s *intent-filter*-om koji će prihvaćati obavijesti poslane mobilnom uređaju. [28]

```
<service android:name=".services.FirebaseService"
    android:exported="false">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
        <action android:name="com.google.firebase.INSTANCE_ID_EVENT" />
    </intent-filter>
</service>
```

Slika 3.12 *FirebaseService* dodan u *AndroidManifest*

Kako bi se na otvaranje primljene obavijesti otvorila aktivnost koja prikazuje sliku web-kamere pametnog zvona potrebno je u *AndroidManifest*-u i toj aktivnosti dodati *intent-filter* koji će prihvaćati obavijesti poslane od *Firebase Cloud Messaging* servisa.

```
<activity android:name=".activities.CameraViewActivity">
    <intent-filter>
        <action android:name="MainActivity" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Slika 3.13 *intent-filter* aktivnosti za prikaz slike web-kamere pametnog zvona

U aplikaciji *Smart Bell* potrebno je omogućiti pretplaćivanje i odjavljivanje pretplate na određenu temu na koju će se objavljivati poruke o zvonjenju. Funkcionalnost pretplaćivanja i odjave pretplate ostvaruje se pozivanjem instance razreda *FirebaseMessaging* te pozivom njegove funkcije *subscribeToTopic(topic)* odnosno *unsubscribeFromTopic(topic)*. Navedene funkcije su asinkrone funkcije čiji se kraj izvođenja može osluškivati, te se po njihovoj izvedbi na osnovu povratne vrijednosti (uspjeh ili neuspjeh) može obaviti neka akcija.

```
private fun subscribeToTopic(topic: String, context: Context) {
    FirebaseMessaging.getInstance().subscribeToTopic(topic)
        .addOnCompleteListener { task ->
            if (!task.isSuccessful) {
                Toast.makeText(context, text: "Failed to subscribe", Toast.LENGTH_SHORT).show()
            }
        }
}
```

Slika 3.14 Pretplata na temu

3.1.2.3 Prikaz slika bibliotekom Glide

Aplikacija *Smart Bell* mora omogućiti dohvaćanje slike web-kamere sa zadanog *URL*-a. Biblioteka *Glide* je projekt otvorenog koda koji na jednostavna način omogućuje dohvaćanje medija putem interneta. *Glide* koristi prilagođeni *HttpUrlConnection* za dohvaćanje podataka. Za rad s *Glide*-om potrebno je dodati ovisnosti za dohvaćanje projekta. [29]

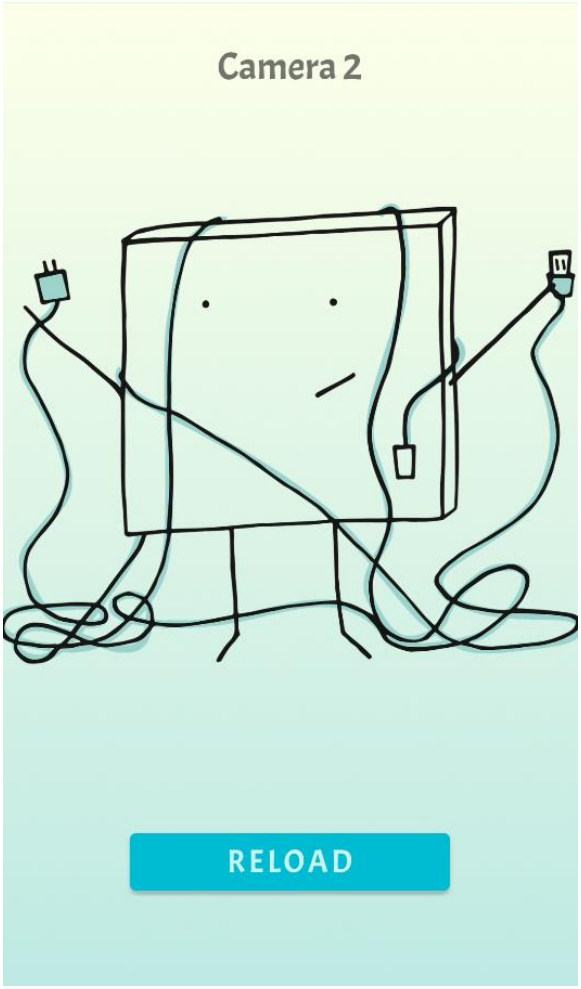
```
//Glide
implementation 'com.github.bumptech.glide:glide:4.12.0'
annotationProcessor 'com.github.bumptech.glide:compiler:4.12.0'
```

Slika 3.15 Ovisnosti za *Glide*

Za dohvaćanje slike s web-kamere *Glide*-u je dovoljno predati kontekst aktivnosti u kojoj će se slika prikazivati, *URL* s koje je potrebno dohvatiti sliku i grafičku komponentu tipa *ImageView* u koju će se slika učitati. *Glide* resurse dohvaća putem Interneta što može rezultirati neuspjehom dohvaćanje resursa ili kašnjenjem zatraženog resursa. U slučaju kašnjenja resursa *Glide* omogućuje postavljenje *placeholder*-a, to je slika koja će se prikazati korisniku dok zatraženi resurs ne stigne do Android aplikacije. U slučaju pogreške, kada je nemoguće dohvatiti zatraženi resurs, (npr. kada mobilni uređaj nije povezan na mrežu) *Glide* omogućava postavljanje slike koja će se prikazivati umjesto zatraženog resursa. *Glide* također omogućava i spremanje dohvaćenog resursa u privremenu memoriju te učitavanje traženog resursa iz privremene memorije, ali za domenu primjene u ovom sustavu potrebno je tu opciju isključiti kako bi se uvijek dohvaćala najnovija slika web-kamere pametnog zvona.

```
private fun reloadCamera(url: String, context: Context, imageView: ImageView) {
    Glide.with(context)
        .load(url)
        .placeholder(R.drawable.security_cam)
        .error(R.drawable.error_image)
        .transition(DrawableTransitionOptions.withCrossFade( duration: 500))
        .diskCacheStrategy(DiskCacheStrategy.NONE)
        .skipMemoryCache( skip: true)
        .into(imageView)
}
```

Slika 3.16 Dohvaćanje slike web-kamere bibliotekom *Glide*



Slika 3.17 *placeholder* za vrijeme učitavanja slike Slika 3.18 pogreška prilikom dohvaćanja slike

3.2. Firebase Cloud Messaging i poslužitelj

Firebase Cloud Messaging (FCM) je servis za pouzdano slanje poruka korisnicima. Implementiranje slanja i primanja poruka pomoću *Firebase Cloud Messaging* servisa sastoji se od dvije glavne komponente:

- pouzdano okruženje poput usluge u „oblaku“ za *Firebase* ili poslužitelj aplikacije
- iOS, Android ili web-aplikacija, klijentska aplikacija [4]

Poslužitelj aplikacije stvara poruku koja će se poslati i određuje odredište na koje će se obavijest poslati. Klijentske aplikacije primaju obavijesti od *Firebase Cloud Messaging* servisa. Obavijesti se mogu slati pomoću alata *Firebase Admin SDK* ili *FCM server protocols*. *Firebase Admin SDK* rukuje autorizacijom zahtjeva koji se šalju na poslužitelj

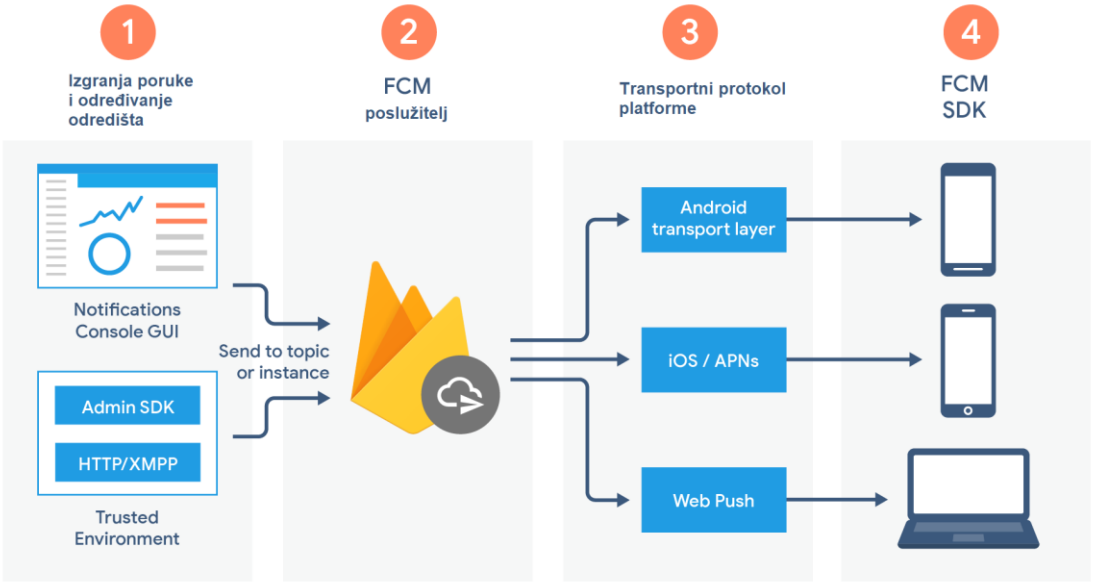
Firebase Cloud Messaging servisa, te upravlja pretplatama na teme. Pomoću *Firebase Admin SDK* moguće je:

- slati poruke pojedinom uređaju,
- slati poruke na specifične teme na koje su uređaji pretplaćeni,
- pretplatiti i odjaviti pretplatu uređaja na specifičnu temu i
- konstruirati poruke s podatcima prilagođene ciljanoj platformi [4]

Zbog navedenih mogućnosti za implementaciju slanja obavijesti na Android aplikaciju *Smart Bell* koristit će se *Firebase Admin SDK.*

Slanje obavijesti putem *Firebase Cloud Messaging* servisa odvija se u 4 koraka

- izgradnja poruke i odredište poruke u povjerljivom okruženju,
- poslužitelj *Firebase Cloud Messaging* servisa zaprima poruku i dodaje joj meta podatke,
- transportni protokol na razini specifične platforme prosljeđuje poruku do odredišta i
- FCM SDK (*software development kit*) na korisnikovom uređaju prikazuje obavijest [3]



Slika 3.19 Dijagram slanja obavijesti putem *Firebase Cloud Messaging* servisa [3]

Poslužitelj je razvijen kako bi objavljivao obavijesti na *Firebase Cloud Messaging* servisu. Kako bi poslužitelj objavio obavijest, treba zaprimiti *HTTP POST* zahtjev u tijelu kojega će biti definiran naslov obavijesti koju treba objaviti, poruka koju želimo poslati krajnjem korisniku te tema na koju obavijest treba objaviti. Zaprimanjem tako definirane obavijesti, poslužitelj će na *Firebase Cloud Messaging* servis poslati autorizirani *HTTP POST* zahtjev kako bi *Firebase* proslijedio obavijest do registriranih Android aplikacija.

3.2.1. Opis rada poslužitelja

Za slanje obavijesti putem *Firebase Cloud Messaging* servisa potrebno je implementirati pouzdani poslužitelj koji će generirati obavijest koju treba poslati korisniku, te odrediti kojem korisniku treba poslati generiranu poruku. Za ostvarenje potrebnih funkcionalnosti slanja obavijesti o zvonjenju od pametnog zvona do Android aplikacije razvijen je poslužitelj koji prima *HTTP POST* zahtjeve te na osnovu primljenih parametara u tijelu zahtjeva generira obavijesti.

Poslužitelj sluša na otvorenom *portu* i kada zaprimi *POST* zahtjev na putanji „*notification/topic*“ pokreće slanje obavijesti na *Firebase Cloud Messaging* servis. Obavijesti se objavljuju na *Firebase Cloud Messaging* servis pomoću gotovih *Firebase*-ovih razreda i funkcija. Parametri poruke (naslov, tijelo poruke i tema) se izvlače iz pristiglog *POST* zahtjeva te se dodaju poruci. Poruci je potrebno zadati i konfiguracije na temelju platforme uređaja na kojoj je Aplikacija *Smart Bell* instalirana. Za kreiranje i slanje obavijesti stvorena su dva servisa koji će baratati *Firebase*-ovim funkcijama za konfiguriranje obavijesti. Neki od dodatnih parametara koji se predaju pri kreiranju obavijesti su zvuk obavijesti, boja obavijesti, ali i *ClickAction* koji određuje koja aktivnost aplikacije će se otvoriti kada korisnik klikne na „push“ obavijest.

```
@RestController
public class PushNotificationController {
    private final PushNotificationService pushNotificationService;
    public PushNotificationController(PushNotificationService pushNotificationService) {
        this.pushNotificationService = pushNotificationService;
    }
    @PostMapping("/notification/topic")
    public ResponseEntity sendNotification(@RequestBody PushNotificationRequest request) {
        pushNotificationService.sendPushNotificationWithoutData(request);
        return new ResponseEntity<>(new PushNotificationResponse(
                HttpStatus.OK.value(), message: "Notification has been sent."), HttpStatus.OK);
    }
```

Slika 3.20 Zaprimanje *POST* zahtjeva

3.2.2. Povezivanje poslužitelja i FCM servisa

Na početku razvoja poslužitelja koji će objavljivati obavijesti na *Firebase Cloud Messaging* servisu potrebno je integrirati *Firebase Admin SDK* u projekt. Za integraciju *Firebase Admin SDK*-a u projekt potrebno je generirati privatni ključ za autorizaciju na *Firebase Cloud Messaging* servis putem *Firebase* konzole. [17]

Generiranu datoteku potrebno je dodati u projekt. Dodatno potrebno je dodati ovisnosti za *firebase-admin* u projekt.

Kada je *Firebase Admin SDK* dodan u projekt potrebno je inicijalizirati Android aplikaciju pomoću generiranog privatnog ključa. [18]

```
@Service
public class FCMInitializer {
    @Value("${app.firebase-configuration-file}")
    private String firebaseConfigPath;

FirebaseOptions options = new FirebaseOptions.Builder()
        .setCredentials(GoogleCredentials
                .fromStream(new ClassPathResource(firebaseConfigPath).getInputStream()))
        .build();
if (FirebaseApp.getApps().isEmpty()) {
    FirebaseApp.initializeApp(options);
    logger.info("Firebase application has been initialized");
}
```

Slika 3.21 Isječak bitnih linija koda iz razreda *FCMInicializer*

Nakon inicijalizacije aplikacije potrebno je pripremiti servis koji će obaviti objavljivanje poruke na *Firebase Cloud Messaging* servis. U navedenom servisu se kreira poruka na osnovu parametara primljenih u tijelu *HTTP POST* zahtjeva. Poruka se kreira pomoću gotovih *Firebase*-ovih razreda uvezenih u projekt. Stvorenoj poruci se predaju zaprimljeni podatci i ostale potrebne konfiguracije za prikazivanje obavijesti na mobilnom uređaju. Kada je poruka kreirana poziva se *Firebase*-ova funkcija za objavljivanje obavijesti kojoj se stvorena poruka predaje putem argumenta funkcije. [18]

```
public void sendMessage(Map<String, String> data, PushNotificationRequest request)
        throws InterruptedException, ExecutionException {
    Message message = getPreconfiguredMessageWithData(data, request);
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    String jsonOutput = gson.toJson(message);
    String response = sendAndGetResponse(message);
    logger.info("Sent message with data. Topic: "
            + request.getTopic() + ", " + response + " msg " + jsonOutput);
}
```

Slika 3.22 Slanje obavijesti na *Firebase Cloud Messaging* servis

3.3. Home Assistant

Home Assistant je platforma otvorenog koda, razvijena za automatizaciju doma odnosno centralnu kontrolu nad svim pametnim uređajima koje registriramo na platformu te se može njima upravljati putem Interneta. [12]

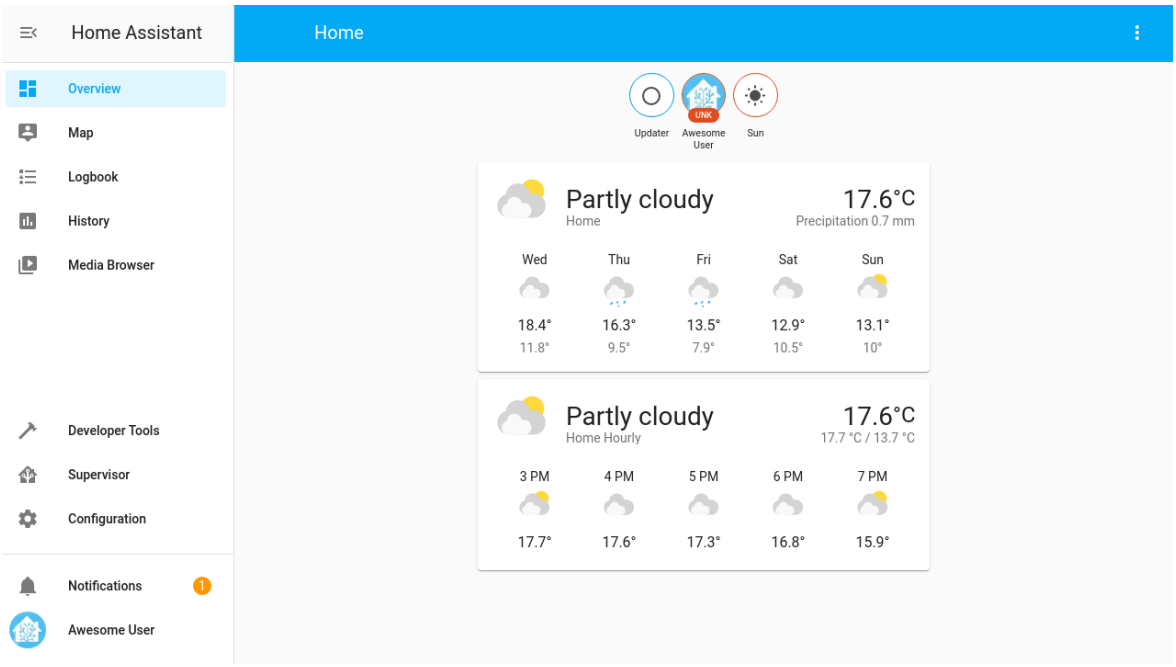
U ovom radu, platforma *Home Assistant*, korištena je za prihvaćanje poruke s pametnog zvona, odnosno *Arduino* pločice povezane na Internet te za prosljeđivanje informacije o zvonjenju do poslužitelja.

Za komunikaciju s pametnim zvonom u *Home Assistant*-u potrebno je konfigurirati *MQTT brokera* koji osluškuje ima li novih obavijesti objavljenih na nekoj od tema. Kada pametno zvono objavi novu poruku na neku od tema, odnosno kada je netko pozvonio, *Home Assistant* šalje *HTTP POST* zahtjev na poslužitelj. U tijelu *POST* zahtjeva definiran je naslov obavijesti koju želimo poslati krajnjem korisniku, tijelo poruke koju želimo poslati i tema na koju je pretplaćena Android aplikacija.

3.3.1. Konfiguracija Home Assistant-a

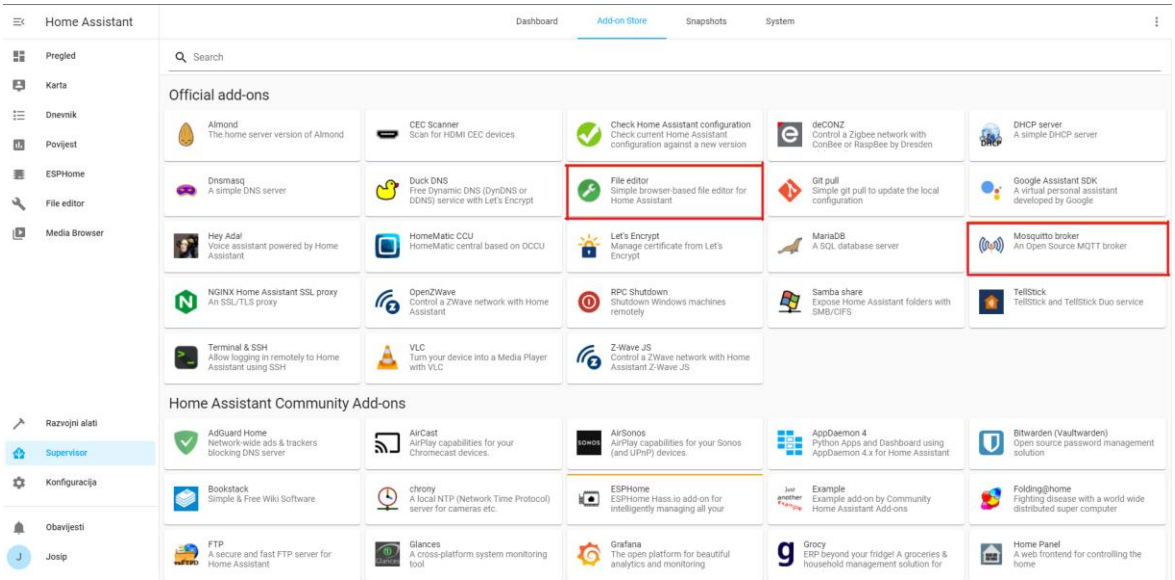
Upute za instalaciju *Home Assistant*-a mogu se naći na ovoj poveznici: https://www.home-assistant.io/installation/ na kojoj je potrebno odabrati metodu instalacije s obzirom na uređaj na koji će se *Home Assistant* instalirati. Za potrebe ovog rada odabrana je instalacija za operacijski sustav *Windows* putem virtualnog stroja uz pomoć alata *VirtualBox*.

Nakon uspješne instalacije *Home Assistant*-ovom korisničkom sučelju može se pristupiti preko web-preglednika na poveznici http://homeassistant:8123/ te je potrebno stvoriti korisnički račun unosom imena, korisničkog imena, lozinke i lokacije nakon čega će se otvoriti početna stranica korisničkog sučelja.



Slika 3.23 Početna stanica korisničkog sučelja *Home Assistant*-a [32]

Za početak konfiguracije instalirane instance *Home Assistant*-a potrebno je u izborniku s lijeve strane odabrati opciju „*Supervisor*“. Nakon otvaranja izbornika „*Supervisor*“ potrebno je otvoriti odjeljak „*Add-on Store*“ te u odjeljku „*Official add-ons*“ odabrati, instalirati i pokrenuti „*File editor*“ i „*Mosquitto broker*“ dodatke.



Slika 3.24 *Add-ons Store*

3.3.2. Automatizacija slanja „push“ obavijesti

Za automatizaciju slanja „push“ obavijesti potreban je dodatak „*File editor*“. U alatnoj traci s lijeve strane potrebno je odabrati opciju „*File editor*“ u strukturi direktorija i

datoteka potrebno je pronaći i odabrati datoteku *configuration.yaml*. Izmjene u datoteci *configuration.yaml* omogućuju naprednu konfiguraciju *Home Assistant*-a.

Kako bi se poruka objavila na *Firebase Cloud Messaging* servis potrebno je poslati *HTTP POST* zahtjev na poslužitelj. Za slanje *HTTP* zahtjeva pomoću *Home Assistant*-a potrebno je stvoriti novu *REST* naredbu. Naredbi je obavezno dodijeliti naziv i *URL* na koji će se *HTTP* zahtjev slati, a za uspješno ostvarivanje potrebnih funkcionalnost potrebno je definirati i parametre *method, payload* i *content_type*. [13]

```
rest_command:
  my_request:
    url: http://8b88bd08433e.ngrok.io/notification/topic
    method: POST
    payload: '{"message":"Go check it out","title": "Someone has rung your doorbell", "topic": "b1eef240-a73d-42e8-a6f5-8150aa58b723"}'
    content_type: 'application/json'
```

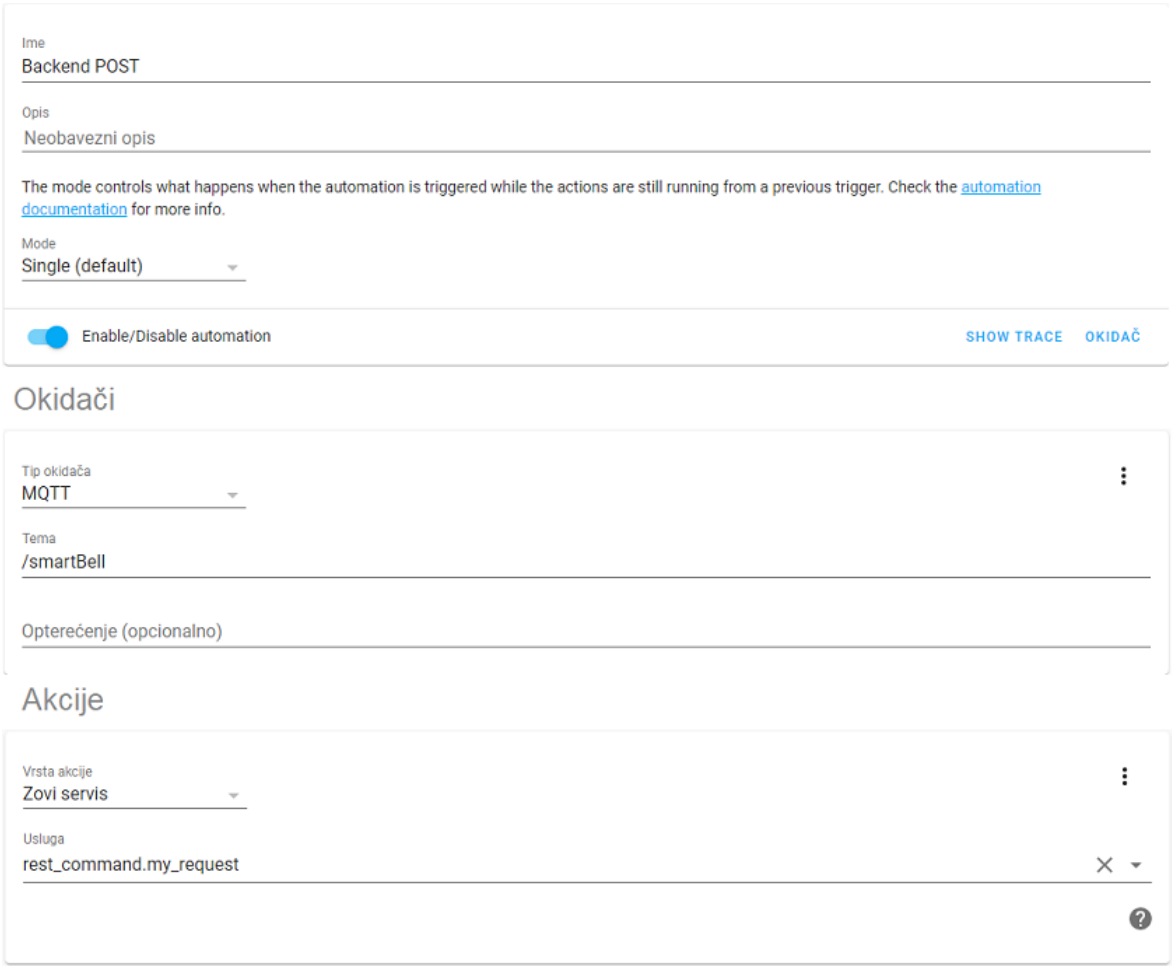
Slika 3.25 REST naredba

Za parametar *url* potrebno je postaviti *URL* na kojemu se može pristupiti poslužitelju (u ovom slučaju to je izvedeno alatom *ngrok* koji stvara javni *URL* poslužitelja pokrenutog na *localhost*-u). Parametar *method* potrebno je postaviti na „*POST*“. Kao *payload* potrebno je predati niz znakova koji čine valjani *JSON* objekt s varijablama *message, title* i *topic*. *Message* i *title* odnose se tijelo i naslov obavijesti koja će se poslati korisniku putem Android aplikacije, a *topic* je *Firebase Cloud Messaging* tema na koju se objavljuje poruka. *Content_type* postavlja se na *application/json.*

Kada je *REST* naredba uspješno dodana potrebno je napraviti automatizaciju koja će na zadani okidač pozivati tu naredbu. U alatnoj traci s lijeve strane potrebno je odabrati opciju „*Konfiguracija*“ i kada se otvori izbornik konfiguracija odabrati opciju „*Automatizacija*“ te kliknuti na gumb „*dodaj automatizaciju*“.

Automatizaciji je potrebno zadati naziv, okidač i akciju koja se automatski izvodi kada se pokrene okidač. [14]

Kao okidač potrebno je postaviti opciju „*MQTT*“ i zadati *MQTT* temu na koju će instanca *Home Assistant*-a biti pretplaćena putem *MQTT brokera*. Kao akciju potrebno je postaviti opciju „*Zovi servis*“ te kao servis koji će se zvati odabrati *REST* naredbu koja je dodana u *configuration.yaml* datoteku.



Slika 3.26 Dodavanje nove automatizacije

3.3.3. Protokol MQTT i MQTT broker

MQTT (*Message Queuing Telemetry Transport*) je protokol za slanje poruka koji se temelje na objavi-pretplati načinu komunikacije, zbog čega se često primjenjuje u okviru koncepta Interneta stvari. Protokol je dizajniran kao vrlo jednostavan (engl. *lightweight*) objavi-pretplati način prijenosa poruka, zbog čega je prikladan za povezivanje udaljenih uređaja s jednostavnom programskom logikom. Zbog svoje jednostavnosti korištenja, *MQTT* protokol je rasprostranjen u raznim domenama primjene Interneta stvari kao što je automobilska industrija, industrijska proizvodnja, prijevoz, pametni dom, i dr. Protokol omogućava dvosmjernu komunikaciju između uređaja i udaljenog poslužitelja što olakšava razašiljanje poruka grupi uređaja. [15]

MQTT klijenti su vrlo mali i zahtijevaju minimalnu količinu računalnih resursa što omogućava korištenje *MQTT* protokola na mikrokontrolerima. Klijent se uvijek spaja na poslužitelj i ima mogućnosti:

- objavljivanja poruka za koje bi drugi klijenti mogli biti zainteresirani,
- pretplatiti se na primanje poruka,
- odjaviti pretplatu na primanje poruka,
- prekid veze s poslužiteljem. [16]

MQTT poslužitelj odnosno *MQTT broker* je program ili uređaj koji obnaša dužnost posrednika između klijenata koji objavljuju poruke i klijenata koji su se pretplatili na primanje poruka. Poslužitelj pruža funkcionalnosti:

- prihvaćanje spajanje klijenata putem mreže,
- prihvaća poruke koje klijenti objavljuju,
- obrađuje zahtjeve za pretplatu i odjavu pretplate klijenata,
- prosljeđuje poruke koje odgovaraju pretplatama klijenta. [16]

Komunikacija *MQTT* protokolom se odvija razmjenom serije *MQTT Control* paketa. Paket se sastoji od najviše 3 dijela a to su:

- *Fixed header*, zaglavlje prisutno u svakom *MQTT Control* paketu,
- *Variable header*, zaglavlje prisutno u nekim *MQTT Control* paketima,
- *Payload*, prisutan u nekim MQTT *Control* paketima. [16]

Svaki *MQTT Control* paket sadrži *Fixed header* zaglavlje veličine dva *byte*-a. Prvi *byte* čine tip paketa i zastavice na bitovima od 7 do 4 i od 3 do 0, a drugi *byte* čini brojčana vrijednost koliko je *byte*-a preostalo u *Variable header* i *Payload*-u. [16]

Variable header se sastoji od dva *byte*-a koji predstavljaju identifikator paketa zapisan u *big-endian* formatu. Identifikator paketa zajednički je istim tipovima paketa. [16]

Payload je posljednji dio *MQTT* paketa koji sadržava tijelo poruke ako se ona šalje. [16]

Postoji više tipova *MQTT* paketa, u kontekstu ovog rada zanimljivi su paketi *PUBLISH* i *SUBSCRIBE*.

Klijenti šalju *PUBLISH* pakete *broker*-u kako bi se poruka proslijedila klijentima zainteresiranim za primanje te vrste poruke. *Broker* šalje *PUBLISH* pakete klijentima kada prosljeđuje objavljene poruke. *PUBLISH* paket se sastoji od sva tri dijela *MQTT* paketa.

Variable header sadrži ime teme na koju se objavljuje poruka i identifikator paketa, a *Payload* sadrži poruku koja se objavljuje. [16]

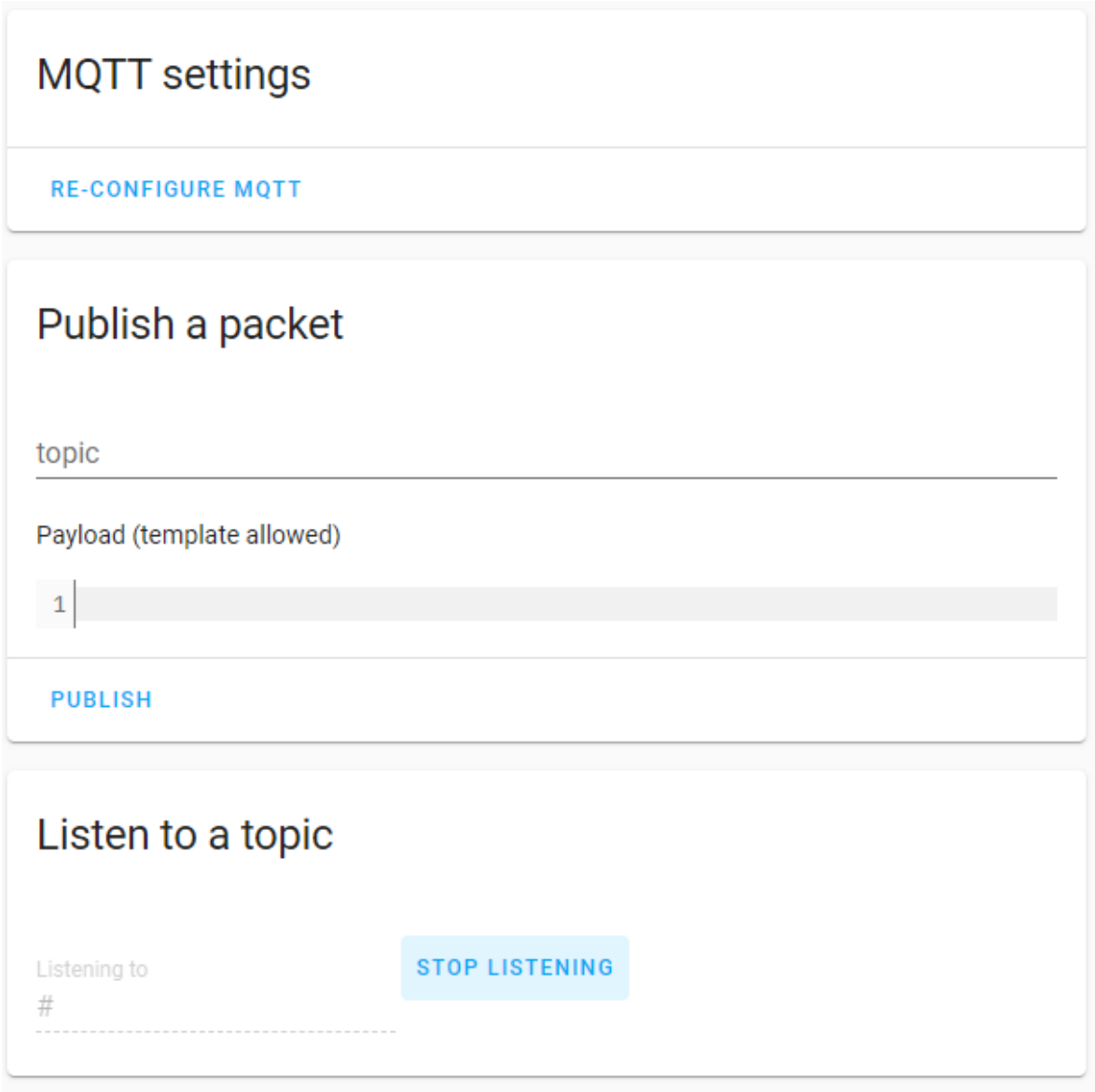
Klijenti šalju *SUBSCRIBE* paket *broker*-u kako bi se pretplatili na jednu ili više tema. *Broker* šalje *PUBLISH* pakete klijentima kako bi im proslijedio poruke koje odgovaraju njihovim pretplatama. *SUBSCRIBE* paket se sastoji od sva tri dijela *MQTT* paketa. *Variable header* sadrži identifikator paketa, a *Payload* sadrži listu tema na koje se klijent želi pretplatiti. [16]

3.3.3.1 Broker Mosquitto

Za uspješno primanje *MQTT* poruka potrebno je konfigurirati *MQTT broker* na instanci *Home Assistant*-a. Postoji više načina postavljanja *broker*-a na *Home Assistant*. Za potrebe ovog rada dodan je novi korisnik čije vjerodajnice su korištene za autorizaciju klijenata.

Novog korisnika se dodaje tako da se u alatnoj traci s lijeve strane odabere opcija „*Konfiguracija*“ te kada se otvori izbornik konfiguracije potrebno je odabrati opciju „*Korisnici*“ te pritisnuti gumb „*add user*“. Nakon pritiska na gumb za dodavanje korisnika otvara se dijalog u kojemu je potrebno upisati korisničko ime i lozinku novog korisnika.

Nakon stvaranja novog korisnika potrebno je konfigurirati *MQTT broker,* za to je potrebno u izborniku „*Konfiguracije*“ odabrati opciju „*Integracije*“ gdje bi se trebala nalaziti opcija za konfiguraciju *Mosquitto broker*-a. Potrebno je kliknuti na gumb „*configure*“ te zatim na gumb „*re-configure mqtt*“ i u dijalogu koji se otvori upisati korisničko ime i lozinku novog korisnika. Zadnji korak konfiguracije *broker*-a je pretplaćivanje na temu. U polje za upis teme na koju će *Home Assistant* biti preplaćen može se upisati ime teme na koju će pametno zvono objavljivati poruke ili se može unijeti „*wildcard*“ znak „#“ koji označava slušanje na svim temama.



Slika 3.27 Pretplaćivanje *Home Assistant*-a na sve teme

3.4. Pametno zvono

Pametno zvono se sastoji od *Arduino Yun* pločice i samostalne web-kamere. *Arduino Yun* pločica povezana je s *TinkerKit Sensor Shield*-om koji omogućava spajanje raznih senzora i aktuatora s *Arduino*-m. Tako je za potrebe ovog rada korišten *TinkerKit PushButton* koji predstavlja tipku zvona. Pritiskom na tipku povezanu s *Arduino*-m počinje radnja zvonjenja te se *Arduino* pločica povezuje s poslužiteljem na kojem je konfiguriran *MQTT* broker (u ovom slučaju to je instanca *Home Assistant*-a), a zatim se preko *MQTT* protokola objavljuje poruku na željenu temu. Web-kamera na primitak *HTTP GET* zahtjeva pošiljatelju (u ovom slučaju to je Android aplikacija) šalje snimak trenutne slike.

Za razvoj programske potpore za *Arduino* koristi varijacija programskog jezika *C++* u razvojnom okruženju *Arduino IDE*. Svaki program za *Arduino* sastoji se od najmanje dvije funkcije, *void setup()* i *void loop()*. U funkciji *setup()* zadaju se određeni parametri prije izvođenja samog programa kako bi se program ispravno izvodio. Funkcija *loop()* je svojevrsni ekvivalent *main()* funkciji u standardnim *C++* programima samo što se funkcija *loop()* beskonačno ponovno poziva sve dok je *Arduino* priključen na izvor napona. Funkcija *loop()* izvodi se nakon funkcije *setup()*.

3.4.1. TinkerKit PushButton za Arduino

TinkerKit PushButton je jednostavan senzor koji se spaja na *TinkerKit Sensor Shield* pločicu. *TinkerKit Sensor Shield* pločica omogućava spajanje senzora i aktuatora izravno na *Arduino*. Senzori se na *TinkerKit Sensor Shield* pločicu povezuju putem 3-polnih konektora. Na pločici postoji 12 konektora od kojih je 6 za analogne ulaze, a 6 za analogne izlaze. *PushButton* treba spojiti na jedan od konektora za ulaze. [5][6]

Razvoj programske potpore za *TinkerKit PushButton* je vrlo jednostavna. Potrebno je uključiti biblioteku za rad s *TinkerKit* senzorima *TinkerKit.h* te se nakon toga može inicijalizirati varijabla *button* tipa *TKButton* na vrijednost ulaznog konektora na koji je tipka spojena.

```
#include <TinkerKit.h>

TKButton button(I0);
```

Prije pristupanja varijabli *button* potrebno je omogućiti pristup digitalnim i analognim ulazima *Arduino*-a. Za to je potrebno u funkciji *setup()* pokrenuti *Bridge*.

```
Bridge.begin();
```

Za ostvarivanje funkcionalnosti zvonjenja potrebno je provjeravati stanje tipke te ako je tipka bila pritisnuta potrebne su daljnje akcije.

```
while(!button.pressed()){}
```

Navedena *while* petlja će blokirati daljnje izvođenje programa sve dok se tipka ne pritisne.

3.4.2. Povezivanje na Internet

Arduino Yun se može ponašati kao pristupna točka za Internet, ali može se spojiti i na postojeću javnu mrežu. Kako bi bilo moguće objavljivati *MQTT* poruke potrebno je *Arduino* spojiti na javnu mrežu. Kada se *Arduino Yun* prvi puta spoji na napajanje bit će u načinu rada pristupne točke koja će se zvati *Linino-XXXXXXXXXXXX*. Potrebno je spojiti

računalo ili pametni mobitel na tu mrežu te konfigurirati *Arduino* za spajanje na javnu mrežu preko web-aplikacije na adresi http://linino.local upisivanjem vjerodajnica potrebnih za spajanje na mrežu. Nakon konfiguracije *Arduino* će se ponovno pokrenuti i bit će spojen putem *Wi-Fi*-a na mrežu. [7]

3.4.2.1 YunClient

Programska potpora za komunikaciju putem Interneta na *Arduino Yun* pločici ostvaruje se pomoću *YunClient-a.* Za to je potrebno uključiti biblioteku *YunClient.h* i inicijalizirati varijablu tipa *YunClient.*

```
#include <YunClient.h>

YunClient yunClient;
```

YunClient je bazna klasa za sve klijentske funkcije *Yun* pločice.[8]

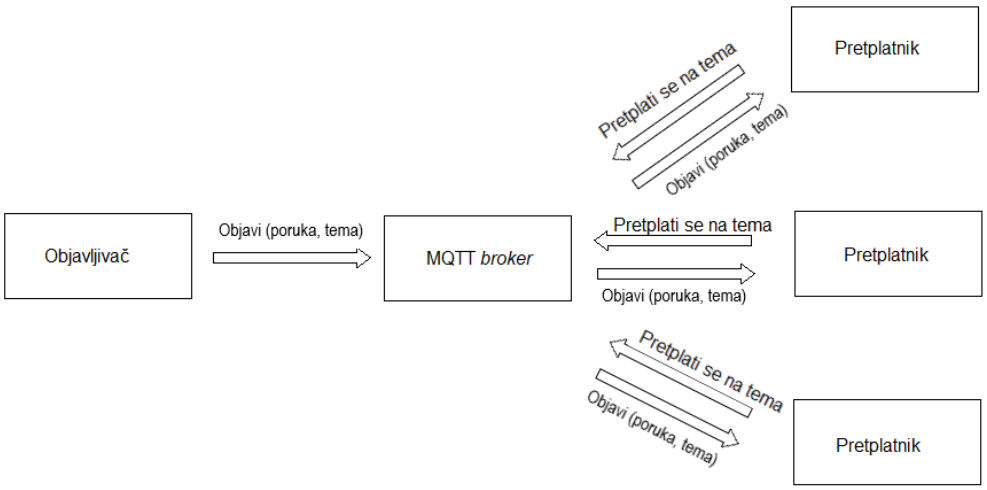
3.4.3. Objavi-pretplati obrazac

Zbog potencijalno velikog broja senzorskih uređaja s kojima je potrebno komunicirati preko Interneta, čija lokacija i ponašanje mogu varirati za vrijeme životnog ciklusa nekog *IoT* sustava, potrebno je uvesti jednostavan način komunikacije koji će odgovarati dinamici sustava. Korištenje *point-to-point* komunikacije u kojoj se senzorski uređaj izravno spaja na poslužitelj dovelo bi do razvoja krutog i glomaznog sustava. Kako bi se smanjio teret s razvoja sustava, povezivanje komponenata sustava se odvija preko posredničke infrastrukture s prikladnim komunikacijskim modelom. [9]

Objavi-pretplati obrazac pruža „*labavo povezanu*“ (engl. *loosely coupled*) ovisnost komponenata koja pojednostavljuje razvoj velikih sustava. Akteri ovog obrasca su objavljivači (engl. *publishers*) i pretplatnici (engl. *subscribers*). Pretplatnici mogu iskazati interes za neki događaj ili uzorak događaja te su obaviješteni o svim događajima, generiranih od objavljivača, koji odgovaraju njihovom interesu. Događaj se asinkrono prosljeđuje do svih pretplatnika koji su iskazali interes za primanje obavijesti za taj događaj. [9]

Objavi-pretplati obrazac ima nekoliko varijacija temeljeno na načinu na koji se pretplaćuje na događaje. Najčešće vrste pretplate su pretplata na temu, pretplata na sadržaj događaja i pretplata na tip događaja. Za ostvarivanje funkcionalnosti zvonjenja u ovom radu korišten je obrazac pretplate na temu. Kod *objavi-pretplati* obrasca temeljnog na pretplati na teme sudionici sustava:

- objavljuju događaje na pojedinačne teme
- pretplaćuju se na pojedinačne teme koje se identificiraju ključnom riječi. [9]



Slika 3.28 Skica *objavi-pretplati* obrasca

3.4.3.1 PubSubClient za Arduino

Razvoj programske potpore *objavi-pretplati* obrasca za *Arduino* ostvaruje se pomoću *PubSubClient.h* biblioteke. *PubSubClient.h* biblioteka pruža jednostavnog klijenata za slanje *objavi-pretplati* poruka putem *MQTT* protokola verzije 3.1.1. [10]

Kako bi se ostvarila funkcionalnost slanja poruka putem *MQTT* protokola potrebno je inicijalizirati klijenta tipa *PubSubClient* te mu kroz konstruktor predati ranije inicijaliziranog klijenta koji će obavljati komunikaciju putem Interneta, kod Arduino Yun pločice to je *YunClient.* [11]

```
#include<PubSubClient.h>

PubSubClient client(yunClient);
```

Za ispravno povezivanje s poslužiteljem *PubSubClient*-u je potrebno predati IP adresu poslužitelja i *port* na koji će se spojiti. IP adresa poslužitelja zadaje se pomoću tipa *IPAdress* koju je potrebno inicijalizirati listom od 4 *byte*-a, odvojena zarezom.

```
IPAdress server(10,1,217,95);
```

IP adresa 10.1.217.95 je adresa na kojoj se može pristupiti konfiguriranoj instanci *Home Assistant* platforme.

IP adresa i *port* se *PubSubClient*-u predaju pomoću funkcije *setServer (IPAddress server, int port)*. Standardni *MQTT port* je 1883.

```
client.setServer(server, MQTT_PORT);
```

Kada se pritisne tipka zvona, *PubSubClient* se spaja na postavljenog poslužitelja te objavljuje poruku na temu na koju je pretplaćena instanca *Home Assistant*-a. Za uspješno povezivanje s poslužiteljem klijentu je potrebno zadati i vjerodajnice konfiguriranog *MQTT brokera.*

```
while(!client.connect(
      "arduinoClient",
      brokerUser,
      brokerPass)){}
client.publish("/smartBell","doorbell");
```

Klijent s nazivom arduinoClient se spaja na *MQTT broker* naziva *brokerUser* i lozinke *brokerPass* i objavljuje poruku „*doorbell*“ na temu „*smartBell*“.

Zaključak

Razvojem tehnologije Interneta i jednostavnih protokola za komunikaciju među uređajima putem Interneta razvijen je koncept Interneta stvari. Internet stvari temelji se na jednostavnim senzorskim uređajima povezanim na mrežu. Povezni uređaji, odnosno „stvari“ komuniciraju s poslužiteljem koji zaprima senzorska mjerenja s povezanih uređaja. Na temelju pristiglih vrijednosti, poslužitelj može pokrenuti određenu akciju ili obavijestiti korisnika o zabilježenim vrijednostima, te ga upozorava kako je potrebna njegova intervencija. U ovom radu obrađen je scenarij sustava Interneta stvari iz domene pametnog doma. Sustav se temelji na pametnom zvonu povezanom s Android aplikacijom.

Za ostvarivanje raspodijeljene arhitekture sustava Interneta stvari razvijeno je više komponenata za komunikaciju s karaj na kraj, od senzorskih uređaja na pametnom zvonu do korisničke aplikacije. Razvijene komponente komuniciraju putem Interneta različitim protokolima, različite složenosti.

Sustav se sastoji od pametnog zvona koje je povezano s Android aplikacijom, te na pritisak tipke zvona, zvono šalje obavijest korisniku u obliku „*push*“ obavijesti kroz aplikaciju. Otvaranjem pristigle obavijesti, Android aplikacija se spaja na web-kameru pametnog zvona i dohvaća trenutnu sliku kamere. Za implementaciju slanja obavijesti korisniku razvijen je poslužitelj koji na pobudu pritiska tipke zvona kreira obavijest i određuje ciljne instance aplikacije kojima će se obavijest poslati.

Literatura

[1] Oracle – What is IoT https://www.oracle.com/internet-of-things/what-is-iot/ (23.5.2021.)

[2] Firebase – Build app server send request 1.6.2021. https://firebase.google.com/docs/cloud-messaging/send-message (4.6.2021.)

[3] Firebase – FCM Architetural Overview 1.6.2021. https://firebase.google.com/docs/cloud-messaging/fcm-architecture (4.6.2021.)

[4] Firebase – Firebase Cloud Messaging 1.6.2021. https://firebase.google.com/docs/cloud-messaging (4.6.2021.)

[5] TinkerKit Sensor Shield V.2 Overview http://www.farnell.com/datasheets/1581474.pdf (29.5.2021.)

[6] TinkerKit PushButton Overview https://www.mouser.com/catalog/specsheets/TinkerKitPushButton.pdf (29.5.2021)

[7] Arduino – Getting Started with the Arduino Yun 5.2.2018. https://www.arduino.cc/en/Guide/ArduinoYun (29.5.2021.)

[8] Arduino – YunClient https://www.arduino.cc/en/Reference/YunClientConstructor (29.5.2021.)

[9] P. T. Eugster, P. A. Felber, R. Guerraoui and A.-M. Kermarrec, "The many faces of publish/subscribe", ACM Computing Surveys (CSUR), vol. 35, no. 2, pp. 114-131, Jun. 2003.

[10] Arduino - PubSubClient https://www.arduino.cc/reference/en/libraries/pubsubclient/ (29.5.2021.)

[11] PubSubClient – API Documentation https://pubsubclient.knolleary.net/api#connect (29.5.2021.)

[12] Home Assistant https://www.home-assistant.io/ (29.5.2021.)

[13] Home Assistant – RESTful Command https://www.home-assistant.io/integrations/rest_command/ (29.5.2021.)

[14] Home Assistant – Automating Home Assistant https://www.home-assistant.io/getting-started/automation/ (29.5.2021.)

[15] MQTT https://mqtt.org/ (29.5.2021.)

[16] OASIS Standard MQTT Version 3.1.1 29.10.2014. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html (29.5.2021.)

[17] Firebase – Add the Firebase Admin SDK to your server 1.6.2021. https://firebase.google.com/docs/admin/setup (4.6.2021.)

[18] Rafal, Send push notifications from Spring Boot server-side application using FCM 5.6.2019. https://blog.mestwin.net/send-push-notifications-from-spring-boot-server-side-application-using-fcm/ (30.5.2021.)

[19] Developer Android – Guide to app architecture 29.3.2021. https://developer.android.com/jetpack/guide (30.5.2021.)

[20] Developer Android – MutableLiveData 24.2.2021 https://developer.android.com/reference/androidx/lifecycle/MutableLiveData (30.5.2021.)

[21] Developer Android – LiveData 24.2.2021. https://developer.android.com/topic/libraries/architecture/livedata (30.5.2021.)

[22] Developer Android – RecyclerView 5.5.2021. https://developer.android.com/reference/kotlin/androidx/recyclerview/widget/RecyclerView (30.5.2021.)

[23] Developer Android – LayoutManager 5.5.2021. https://developer.android.com/reference/kotlin/androidx/recyclerview/widget/RecyclerView.LayoutManager (30.5.2021.)

[24] Developer Android – Adapter 5.5.2021. https://developer.android.com/reference/kotlin/androidx/recyclerview/widget/RecyclerView.Adapter (30.5.2021.)

[25] Developer Android – ViewHolder 5.5.2021. https://developer.android.com/reference/kotlin/androidx/recyclerview/widget/RecyclerView.ViewHolder (30.5.2021.)

[26] Tutorialspoint – SQLite Database https://www.tutorialspoint.com/android/android_sqlite_database.htm (30.5.2021.)

[27] Developer Android – SQLiteOpenHelper 24.2.2021. https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper (30.5.2021.)

[28] Firebase – Set up an Android client 1.6.2021. https://firebase.google.com/docs/cloud-messaging/android/client (4.6.2021.)

[29] Glide https://bumptech.github.io/glide/ (30.5.2021.)

[30] Wikipedia - Internet Of Things 4.6.2021. https://en.wikipedia.org/wiki/Internet_of_things (4.6.2021.)

[31] Skica arhitekture usluga Interneta stvari https://www.altexsoft.com/media/2020/08/iot-architecture-building-blocks.png (4.6.2021.)

[32] Home Assistant – Onboarding Home Assistant https://www.home-assistant.io/getting-started/onboarding/ (6.6.2021.)

Sažetak

Sustavi temeljeni na konceptu Interneta stvari zasnivaju se na raspodijeljenoj arhitekturi. Arhitekturu takvih sustava čine senzorski uređaji koji se putem mreže spajaju na poslužitelj i šalju zabilježene vrijednosti. Poslužitelj zaprima i obrađuje senzorska mjerenja te ih prezentira korisniku. Na osnovu prezentiranih podataka korisnik može poduzimati određene akcije ili upravljati povezanim uređajima.

Zadatak sustava razvijenog u sklopu ovog završnog rada je povezivanje pametnog zvona s Android aplikacijom u vidu slanja „*push*“ obavijesti na korisnikov mobilni uređaj kada netko pozvoni na pametno zvono. Sustav se sastoji od pametnog zvona, instance *Home Assistant* platforme, poslužitelja za komunikaciju s *Home Assistant* platformom, *Firebase Cloud Messaging* servisa i Android aplikacije.

Pametno zvono sastoji se od *Arduino Yun* pločice i web-kamere. *Arduino* pločica je povezana s *TinkerKit Sensor Shield*-om preko kojeg je, na *Arduino* pločicu, povezan *TinkerKit PushButton. TinkerKit PushButton* je jednostavni senzorski uređaj na čiji pritisak kreće akcija zvonjenja. Akcija zvonjenja se sastoji od povezivanja s *Home Assistant* platformom i objavljivanja poruke putem *MQTT* protokola.

Home Assistant je konfiguriran tako da putem *MQTT broker*-a prima obavijesti o porukama koje pametno zvono objavljuje. Na primitak obavijesti o zvonjenju *Home Assistant* poslužitelju šalje *HTTP POST* zahtjev na temelju čijeg sadržaja će poslužitelj kreirati „*push*“ obavijest i odrediti primatelje kreirane obavijesti.

Iz zaprimljenog *HTTP POST* zahtjeva, poslužitelj izvlači naslov obavijesti, tijelo obavijesti i temu na koju će se obavijest objaviti. Kreiranu obavijest poslužitelj objavljuje na *Firebase Cloud Messaging* servis na zadanu temu. *Firebase Cloud Messaging* servis obavijest prosljeđuje uređajima koji su se putem Android aplikacije prijavili za primanje obavijesti s te teme.

Otvaranjem pristigle obavijesti, Android aplikacija se povezuje na web-kameru pametnog zvona i dohvaća trenutnu sliku kamere.

Summary

Systems based on Internet of Things concept rest upon distributed architecture. Architecture of such systems is made of sensor devices which are connected to server and they send values they record to server. Server accepts and processes measures received from sensors and presents that measures to user. Based on presented values user can take some actions or manage connected devices.

System developed for this final assignment is made to connect smart doorbell with Android application by sending push notifications to user's mobile phone when someone pushes button on smart doorbell. System is made of smart doorbell, instance of Home Assistant platform, server for communication with Home Assistant, Firebase Cloud Messaging service and Android application.

Smart doorbell consists of Arduino Yun board and web-cam. To Arduino board is connected TinkerKit Sensor Shield on which is connected TinkerKit PushButton. TinkerKit PushButton is simple sensor device, push on this button starts action of ringing doorbell. Ringing doorbell action contains two steps, connecting to Home Assistant platform and publishing message via MQTT protocol.

Home Assistant is configured to receive messages that smart doorbell publishes to MQTT broker. When Home Assistant receives message that someone have rung doorbell, it sends HTTP POST Request which body will be used by server to get parameters to create push notification and determine who to send created notification.

From received HTTP POST Request server gets title of notification, message of notification and topic on which to publish notification. Created notification, server publishes to Firebase Cloud Messaging service to specific topic. Firebase Cloud Messaging service forwards notification to mobile devices which are subscribed to that topic via Android application.

When user opens received push notification, Android application connects to smart doorbell's web-cam and fetches current image of web-cam.

Skraćenice

FCM	*Firebase Cloud Messaging*	razmjena poruka u oblaku Firebase
HTTP	*Hypertext Transfer Protocol*	protokol za prijenos hiperteksta
IDE	*Integrated Develpoment Environment*	integrirano razvojno okuženje
IoT	*Internet of Things*	Internet stvari
JSON	*JavaScript Object Notation*	JavaScript notacija objekata
MQTT	*Message Queuing Telemetry Transport*	red čekanja poruka za telemetrijski transport
MVVM	*Model-View-ViewModel*	model - pogled - pogled na model
REST	*Representational State Transfer*	reprezentacijski prijenos stanja
SDK	*Software Development Kit*	paket za razvoj programa
URL	*Uniform Resource Locator*	ujednačeni lokator sadržaja

Privitak

Instalacija programske podrške

Instalacija programske podrške se sastoji od četiri dijela:

- instalacija programske podrške za pometeno zvono,
- instalacija i konfiguracija instance *Home Assistant* platforme,
- instalacija programske podrške poslužitelja i
- instalacija Android aplikacije

Instalacija programske podrške za pametno zvono

Za instalaciju programske podrške za pametno zvono potrebno je preuzeti, instalirati i konfigurirati *Arduino IDE* minimalno verzije 1.8.13 sa sljedeće poveznice:

- https://www.arduino.cc/en/software

Nakon instalacije *Arduino IDE*-a potrebno je preuzeti kod koji će se izvoditi na *Arduino Yun* pločici. Kod se nalazi na poveznici:

- https://gitlab.tel.fer.hr/kusek-zr-2021/zr-lukacevic/-/blob/master/SOFTWARE/Arduino/smartBell/smartBell.ino

Preuzeti kod potrebno je kroz *Arduino IDE* učitati na *Arduino Yun* pločicu klikom na gumb „Prenesi“ u alatnoj traci *Arduino IDE*-a.

Za ostvarenje svih funkcionalnosti ovog rada korištena je i prethodno konfigurirana web-kamera koja na primitak *HTTP GET* zahtjeva vraća snimak trenutne slike.

Instalacija i konfiguracija instance Home Assistant platforme

Upute za instalaciju i konfiguraciju instance *Home Assistant* platforme za obavljanje funkcionalnosti potrebnih za ovaj rad su već opisane ranije u tekstu u poglavljima:

- 3.3.1 Konfiguracija Home Assistant-a,
- 3.3.2 Automatizacija slanja „push“ obavijesti i
- 3.3.3.1 Mosquitto broker

Instalacija programske podrške poslužitelja

Za instalaciju programske podrške poslužitelja potrebno je prethodno na računalo, na kojem će se kod poslužitelja izvoditi, preuzeti, instalirati i konfigurirati *Java JRE* ili *Java JDK* minimalne verzije 15. Najnoviju verziju *Java*-e moguće je preuzeti s poveznice:

- https://www.oracle.com/java/technologies/javase-downloads.html

Nakon instalacije navedenog alata potrebno je preuzeti datoteku *smartbell-0.0.1.jar* sa sljedeće poveznice:

- https://gitlab.tel.fer.hr/kusek-zr-2021/zr-lukacevic/-/blob/master/SOFTWARE/smartbell-0.0.1.jar

Nakon preuzimanja datoteke *smartbell-0.0.1.jar* potrebno ju je pokrenuti. Za pokretanje navedene datoteke potrebno je pozicionirati se kroz *Terminal* (ili *Command Prompt*) u direktorij u kojemu se datoteka nalazi i izvesti naredbu:

```
java --enable-preview -jar smartbell-0.0.1.jar
```

Izvođenjem navedene naredbe pokreće se poslužitelj na adresi *localhost:8080*, što je privatna adresa računala na kojemu se program izvodi. Za ispravan rad sustava i izvođenje komunikacije između instance *Home Assistant* platforme potrebno je stvoriti javnu adresu kojoj će se moći pristupiti s bilo kojeg uređaja.

Za stvaranje javne adrese može se koristiti alat *ngrok*, alat se može preuzeti sa sljedeće poveznice:

- https://ngrok.com/download

Kada je alat instaliran i konfiguriran potrebno ga je pokrenuti naredbom:

```
ngrok http 8080
```

Instalacija Android aplikacije

Android aplikaciju je moguće preuzeti sa sljedeće poveznice:

- https://gitlab.tel.fer.hr/kusek-zr-2021/zr-lukacevic/-/blob/master/SOFTWARE/app-debug.apk

Aplikaciju je potrebno preuzeti i instalirati na mobilnom uređaju s operacijskim sustavom Android minimalne verzije 6.0 (*API level 23*).