



# УНИВЕРЗИТЕТ У НОВОМ САДУ ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА



УНИВЕРЗИТЕТ У НОВОМ САДУ  
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА  
НОВИ САД  
Департман за рачунарство и аутоматику  
Одсек за рачунарску технику и рачунарске комуникације

## ЗАВРШНИ (BACHELOR) РАД

Кандидат: Лана Салаи  
Број индекса: РА45/2014

Тема рада: Развој *widget* апликација за Андроид

Ментор рада: проф. др Иштван Пап

Нови Сад, јул, 2018



## КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, РБР:			
Идентификациони број, ИБР:			
Тип документације, ТД:	Монографска документација		
Тип записа, ТЗ:	Текстуални штампани материјал		
Врста рада, ВР:	Завршни (Bachelor) рад		
Аутор, АУ:	Лана Салаи		
Ментор, МН:	проф. др Иштван Пап		
Наслов рада, НР:	Развој <i>widget</i> апликација за Андроид		
Језик публикације, ЈП:	Српски / латиница		
Језик извода, ЈИ:	Српски		
Земља публиковања, ЗП:	Република Србија		
Уже географско подручје, УГП:	Војводина		
Година, ГО:	2018.		
Издавач, ИЗ:	Ауторски репринт		
Место и адреса, МА:	Нови Сад; трг Доситеја Обрадовића 6		
Физички опис рада, ФО: (поглавља/страна/ цитата/табела/слика/графика/прилога)	7/33/15/3/25/0/0		
Научна област, НО:	Електротехника и рачунарство		
Научна дисциплина, НД:	Рачунарска техника		
Предметна одредница/Клучне речи, ПО:	паметна кућа, зона, уређај, контролер, сензор, <i>widget</i> , сервис		
УДК			
Чува се, ЧУ:	У библиотеци Факултета техничких наука, Нови Сад		
Важна напомена, ВН:			
Извод, ИЗ:	У овом раду описано је једно решење клијентске <i>widget</i> апликације засноване на Андроид платформи. Апликација служи за праћење стања контролера унутар зона паметне куће. Реализована је коришћењем <i>Android Studio</i> развојног окружења.		
Датум прихватања теме, ДП:			
Датум одбране, ДО:			
Чланови комисије, КО:	Председник:	проф. др Илија Башичевић	
	Члан:	проф. др Небојша Ђевалица	Потпис ментора
	Члан, ментор:	проф. др Иштван Пап	



## KEY WORDS DOCUMENTATION

Accession number, <b>ANO:</b>		
Identification number, <b>INO:</b>		
Document type, <b>DT:</b>	Monographic publication	
Type of record, <b>TR:</b>	Textual printed material	
Contents code, <b>CC:</b>	Bachelor Thesis	
Author, <b>AU:</b>	Lana Salai	
Mentor, <b>MN:</b>	Istvan Papp, PhD	
Title, <b>TI:</b>	The development of Android widget applications	
Language of text, <b>LT:</b>	Serbian	
Language of abstract, <b>LA:</b>	Serbian	
Country of publication, <b>CP:</b>	Republic of Serbia	
Locality of publication, <b>LP:</b>	Vojvodina	
Publication year, <b>PY:</b>	2018.	
Publisher, <b>PB:</b>	Author's reprint	
Publication place, <b>PP:</b>	Novi Sad, Dositeja Obradovica sq. 6	
Physical description, <b>PD:</b> (chapters/pages/ref./tables/pictures/graphs/appendices)	7/33/15/3/25/0/0	
Scientific field, <b>SF:</b>	Electrical Engineering	
Scientific discipline, <b>SD:</b>	Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, <b>S/KW:</b>	smart home, zone, device, controller, sensor, widget, service	
<b>UC</b>		
Holding data, <b>HD:</b>	The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, <b>N:</b>		
Abstract, <b>AB:</b>	In this paper, one implementation of the client Android widget application is proposed. The implemented application is used for tracking states of controllers inside the zones of a smart home. The application is implemented using Android Studio development environment.	
Accepted by the Scientific Board on, <b>ASB:</b>		
Defended on, <b>DE:</b>		
Defended Board, <b>DB:</b>	President:	Ilija Basicevic, PhD
	Member:	Nebojsa Pjevalica, PhD
	Member, Mentor:	Istvan Papp, PhD
		Menthor's sign

## **Zahvalnost**

Zahvaljujem se mentoru, prof. dr Ištvanu Papu, na stručnoj pomoći i savetima prilikom izrade ovog rada.

Posebno se zahvaljujem Dušanu Janoševiću, Mihailu Lukiću i Dušanu Davidovu na pruženoj pomoći, savetima i strpljenju tokom izrade rada.

Na kraju, zahvaljujem se svojoj porodici i prijateljima koji su uvek bili uz mene i pružali mi podršku tokom čitavog dosadašnjeg školovanja.

## SADRŽAJ

1.	Uvod .....	1
2.	Teorijske osnove .....	3
2.1	OBLO sistem .....	3
2.1.1	Centralni uređaj .....	4
2.1.2	<i>Cloud</i> servis .....	4
2.1.3	Klijentske aplikacije .....	4
2.1.4	AZC ( <i>Active Zone Control</i> ) .....	4
2.1.4.1	AZC kontroleri .....	6
2.2	MQTT protokol .....	6
2.3	<i>Applicaton Widget</i> .....	7
3.	Koncept rešenja .....	9
3.1	Ideja realizacije .....	9
3.1.1	<i>Message payload</i> .....	13
3.2	Arhitektura aplikacije .....	13
3.2.1	<i>MVP pattern</i> .....	13
4.	Programsko rešenje .....	15
4.1	Primer MVP šablona .....	15
4.2	<i>RxJava</i> .....	17
4.3	Komunikacija sa <i>cloud</i> servisom .....	19
4.4	Komunikacija sa centralnim uređajem .....	20
4.5	Servisi .....	21
4.6	<i>Widget</i> .....	23
4.6.1	<i>AppWidgetProvider</i> klasa .....	23
4.6.2	<i>RemoteViewsService</i> i <i>RemoteViewsFactory</i> klase .....	25
5.	Testiranje i rezultati .....	27
5.1	Opis testnog sistema .....	27
5.2	Funkcionalno testiranje .....	28
5.2.1	Prijava na korisnički nalog .....	28
5.2.2	Kreiranje novog korisničkog naloga .....	28

5.2.3	Lista centralnih uređaja .....	28
5.2.4	Konekcija i sinhronizacija .....	29
5.2.5	Instalacija <i>widget</i> komponente.....	29
5.2.6	Prikaz podataka na <i>widget</i> -u i ažuriranje.....	29
5.3	Rezultati .....	29
6.	Zaključak .....	32
7.	Literatura .....	33

## SPISAK SLIKA

Slika 2.1 OBLO sistem -----	3
Slika 2.2 Hijerarhijska organizacija zona -----	5
Slika 2.3 Struktura MQTT poruke-----	7
Slika 3.1 Izgled <i>widget-a</i> (slučaj kad je <i>gateway</i> aktivan i slučaj kad nije) -----	10
Slika 3.2 <i>Widget</i> sa podacima dobijenim od kontrolera zone-----	10
Slika 3.3 <i>Login screen</i> -----	11
Slika 3.4 Lista centralnih uređaja -----	11
Slika 3.5 U toku je konekcija sa centralnim uređajem -----	12
Slika 3.6 U toku je sinhronizacija sa centralnim uređajem-----	12
Slika 3.7 MVP šablon-----	14
Slika 4.1 <i>HomeContract</i> klasa-----	15
Slika 4.2 <i>GatewayListContract</i> klasa -----	16
Slika 4.3 <i>GatewayListPresenter</i> : <i>start</i> metoda -----	18
Slika 4.4 <i>subscribeToGatewayEvents</i> metoda-----	18
Slika 4.5 <i>UseCase</i> : pretplata na događaje centralnog uređaja -----	18
Slika 4.6 <i>UseCase</i> : <i>execute</i> metoda -----	19
Slika 4.7 Prototip preplatnika ( <i>Observer</i> , odnosno <i>Subscriber</i> )-----	19
Slika 4.8 Preplatnik na <i>gateway</i> događaje-----	19
Slika 4.9 Primer zahteva poslatog u vidu JSON objekta -----	21
Slika 4.10 Primer <i>onBind</i> metode u <i>MqttService</i> klasi-----	22
Slika 4.11 <i>onServiceConnected</i> metoda -----	22

Slika 4.12 <i>onEnabled</i> metoda u <i>widget</i> klasi -----	22
Slika 4.13 <i>onUpdate</i> metoda: postavlja se <i>PendingIntent</i> za određeno dugme-----	24
Slika 4.14 Kreiranje <i>PendingIntent-a</i> za datu akciju -----	24
Slika 4.15 <i>onReceive</i> metoda: definisanje akcije-----	25
Slika 5.1 MSC dijagram testnog sistema -----	27

## SPISAK TABELA

Tabela 4.1 Metode za komunikaciju sa <i>cloud</i> servisom-----	20
Tabela 4.2 Metode za komunikaciju sa centralnim uređajem -----	20
Tabela 5.1 Funkcionalni testovi-----	31

## SKRAĆENICE

**MQTT** – Message Queuing Telemetry Transport, komunikacioni protokol

**TCP/IP** – Transmision Control Protocol/Internet Protocol, mrežni protokol

**QoS** – Quality of Service, kvalitet usluge

**BLE** – Bluetooth Low Energy, komunikacioni protokol

**API** – Application Programming Interface, aplikativna programska sprega

**JSON** – JavaScript Object Notation, Java skript objektna notacija

**MVP** – Model View Presenter

**RxJava** – Reactive Extensions for Java

**XML** – eXtensible Markup Language

**SDK** – Software Development Kit

**MSC** – Message Sequence Chart

## 1. Uvod

Pametne kuće stiču sve veću popularnost u poslednjih nekoliko godina. One predstavljaju sistem u kom su svakodnevno korišćeni uređaji u domaćinstvu povezani u jednu mrežu, preko interneta ili lokalno. Time se postiže intuitivnija kontrola tih uređaja. Stalni razvoj tehničkih rešenja, zajedno sa konceptom uređaja povezanih na internet, doživljavaju procvat na polju kućne automatizacije. Iz dana u dan raste dostupnost električnih uređaja različitih proizvođača, a cena im je sve pristupačnija.

Važan aspekt sistema za kućnu automatizaciju je omogućavanje udaljene kontrole različitih uređaja sa jednog mesta, putem mobilne ili web aplikacije. Jedna od postojećih implementacija sistema za upravljanje pametnom kućom je i OBLO sistem.

Periferni uređaji su u OBLO sistemu povezani na centralni uređaj (eng. *gateway*) čija je uloga kontrola i/ili nadzor ovih uređaja. Korisničke aplikacije takođe komuniciraju sa centralnim uređajem. Komunikacija može da se odvija direktno sa centralnim uređajem, preko mrežnih rutera u lokalnoj mreži, ili sa udaljene lokacije pomoću *cloud* servisa.

Kako se, zahvaljujući ovom tehnološkom napretku, omogućila brža, jednostavnija i efikasnija kontrola okruženja sopstvenog doma, tako se ubrzo javila potreba da se i vreme utrošeno na zadavanje komandi željenim uređajima, putem aplikacija, svede na minimum. Kod mobilnih aplikacija problem je moguće rešiti dodavanjem *widget* komponente u aplikaciju. *Widget* nam omogućava brz pregled najbitnijih stavki koje aplikacija pruža.

U ovom radu opisano je jedno takvo rešenje. Oslanjajući se na postojeću OBLO aplikaciju, podržanu na Android platformi, implementirana je jednostavnija aplikacija koja

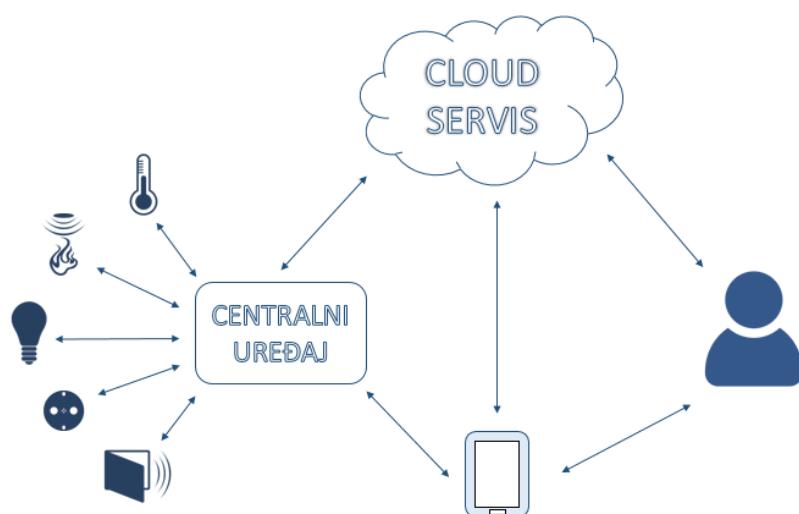
sadrži *widget* na početnom ekranu. U nastavku će, nakon opisa postojećeg OBLO sistema kućne automatizacije i svih njegovih komponenti, biti detaljno opisana implementacija rešenja.

## 2. Teorijske osnove

Na početku ovog poglavlja opisan je OBLO sistem sa svim komponentama koje ga sačinjavaju. Dat je opis protokola po kom uređaji komuniciraju sa centralnom jedinicom (*gateway*). Predstavljena je ideja nove implementacije - AZC (eng. *Active Zone Control*), koja predstavlja način rasporeda i kontrole uređaja u sistemu kućne automatizacije. Na samom kraju, opisan je koncept *widget*-a i prednosti njegovog korišćenja u mobilnim aplikacijama.

### 2.1 OBLO sistem

OBLO [1] rešenje za kućnu automatizaciju čini više distribuiranih komponenti [2], od kojih je glavna komponenta centralni uređaj. Centralni uređaj je posrednik u komunikaciji između klijentskih aplikacija i perifernih uređaja. Koristeći klijentsku aplikaciju, korisnik ima mogućnost da u lokalnoj mreži komunicira sa centralnim uređajem, posredstvom mrežnih ruter, ili sa udaljene lokacije preko *cloud-a*.



Slika 2.1 OBLO sistem

## 2.1.1 Centralni uređaj

Uloga centralnog uređaja je da perifernim uređajima prosledi komande zadate od strane korisnika, što se odvija putem MQTT protokola [3]. S obzirom na to da je komunikacija dvosmerna, informacije sa perifernih uređaja i eventualne promene njihovih stanja šalju se nazad ka korisniku.

U OBLO sistemu uređaji mogu biti senzorski i aktuatorski [4]. Senzori su uređaji kojima ne može da se upravlja. Služe za detekciju i prijavljivanje važnih karakteristika kuće kao što su temperatura, vlažnost vazduha, detekcija požara, poplave, stanje vrata ili prozora (otvoreno/zatvoreno) itd. Sa druge strane, postoje aktuatori kojima je moguće upravljati. Njima korisnik može upravljati slanjem komandi centralnom uređaju. U aktuatore spadaju pametne utičnice, prekidači, sijalice itd, odnosno svi uređaji na čije ponašanje korisnik može da utiče.

Kako bi komunikacija sa uređajima različitog tipa bila uspešna, s obzirom da svaki uređaj ima jedinstveno ponašanje i koristi tačno određen protokol za komunikaciju, centralni uređaj podržava više različitih komunikacionih protokola, kao što su *ZigBee*, *Z-Wave*, *IP* i *BLE* [5].

## 2.1.2 Cloud servis

U situaciji kada se korisnik nalazi van svoje kuće, a potrebno mu je da pristupi centralnom uređaju, rešenje problema se svodi na korišćenje *cloud* servisa. On je konstantno dostupan na globalnoj mreži i na raspolaganju je svim korisnicima, odnosno zajednički je za sve korisnike i centralne uređaje. Svaki centralni uređaj poseduje identifikacioni broj koji ga povezuje sa korisničkim nalogom. Prijavom na svoj nalog, korisniku je omogućena kontrola pametne kuće.

Korisno svojstvo *cloud*-a je i to što administratoru pruža mogućnost da održava celokupan sistem, upravlja bazom korisnika i kontrolera, ažurira programsku podršku kontrolera i pruža podršku krajnjim korisnicima [6].

## 2.1.3 Klijentske aplikacije

Pored direktnog kontrolisanja pametne kuće preko *cloud*-a, korisnik se sa sistemom može povezati i pomoću klijentske aplikacije. Klijentska aplikacija komunicira sa centralnim uređajima u lokalnoj mreži, a konekciju je moguće uspostaviti takođe putem *cloud* servisa. Pomoću aplikacije korisnik dobavlja listu dostupnih centralnih uređaja i odlučuje sa kojim uređajem želi da se poveže kako bi se omogućila dalja kontrola. Mobilna aplikacija je podržana od strane Android i iOS platforme.

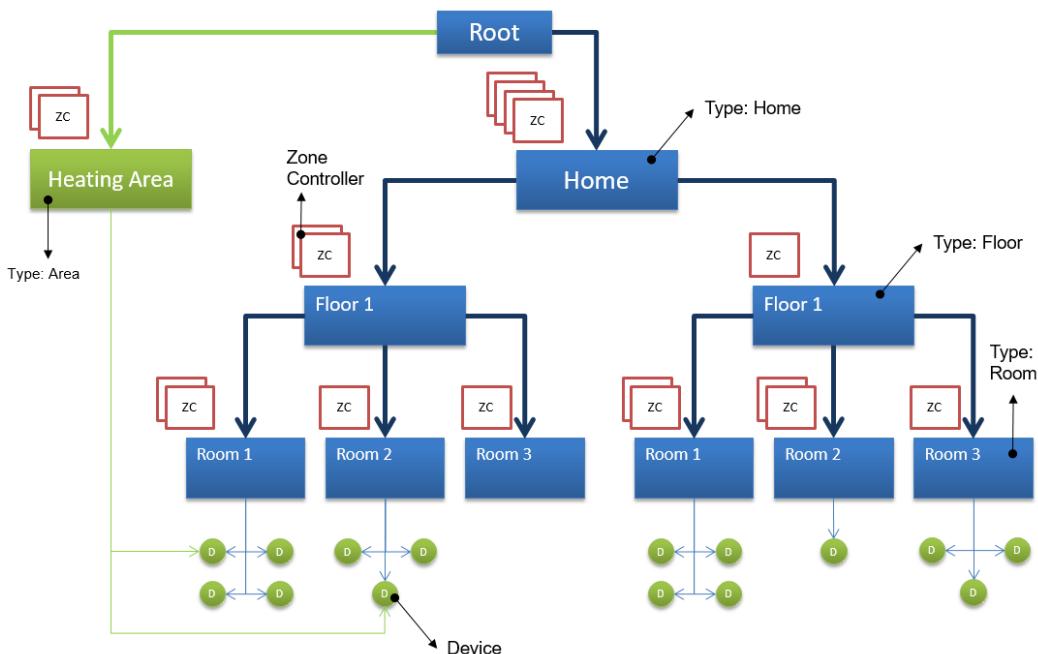
## 2.1.4 AZC (*Active Zone Control*)

U postojećoj implementaciji sistema, uređaji su raspoređeni po prostorijama, koje su dalje raspoređene po spratovima. Ideja nove implementacije (AZC) je raspored uređaja po zonama,

umesto po prostorijama i spratovima. Samim tim, uređaje je moguće grupisati na način koji najviše odgovara korisniku i koji mu omogućava intuitivno upravljanje pametnom kućom.

Ovo rešenje omogućava da se svi uređaji iste ili slične namene povežu u jednu celinu kako bi se mogli grupno kontrolisati, bez potrebe slanja komande svakom pojedinačnom uređaju. Na ovaj način korisnik može brzo i jednostavno da uključi sva svetla u kući ili da postavi željenu temperaturu na nivou prostorije ili čitave kuće. Postiže se lakši pristup senzorskim informacijama kao što je prosečna vlažnost vazduha, temperatura itd.

Organizacija zona u pametnoj kući je hijerarhijska (Slika 2.2). Njihov raspored ima strukturu stabla, sa jednom glavnom, korenskom zonom (eng. *root*) na vrhu arhitekture. Ona je sakrivena od korisnika i ne može joj se pristupiti. Svaka zona može pripadati jednoj ili više drugih zona, a takođe može da sadrži jednu ili više zona. Tako pored korenske zone, postoji još četiri različita tipa zona. Zona čitave kuće (eng. *home*) je korenska zona u smislu rasporeda kuće. Postoji samo jedna instance ove zone i ona grupiše zone tipa sprat (eng. *floor*). Zona kuće pripada prethodno pomenutoj korenskoj zoni. Sprat je zona koja grupiše zone sobe (eng. *room*) u kojima se nalaze uređaji. Pored ovih zona, koje se u fizičkom smislu baziraju na početnom rasporedu sistema pametne kuće, postoji i generički tip zone, koja može biti podzona bilo koje druge zone ili može da grupiše ostale zone. Ovakva zona predstavlja oblast (eng. *area*). Oblast je takođe član glavne, korenske zone.



Slika 2.2 Hijerarhijska organizacija zona

Zone ne kontrolišu uređaje direktno, već koriste kontrolere zona (eng. *Zone Controller*, ZC), što znači da zone mogu da sadrže druge zone, uređaje i svoje kontrolere.

#### 2.1.4.1 AZC kontroleri

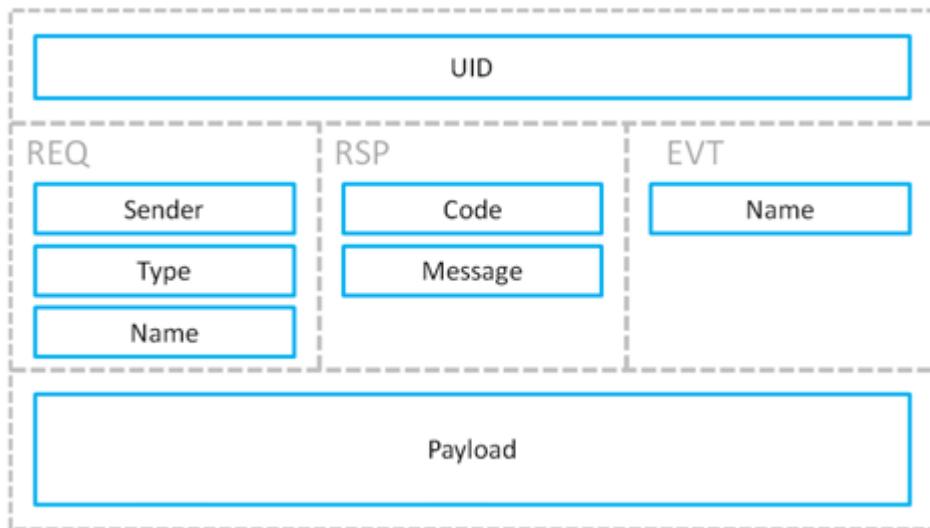
Kontroleri zona su virtuelne komponente koje implementiraju logiku iza AZC koncepta, uključujući grupnu kontrolu uređaja u zoni u kojoj se oni nalaze i/ili uređaja iz zona koje pripadaju njihovoj zoni. Implementiraju funkcionalnosti za neke određene svrhe kao što su grejanje, osvetljenje, hlađenje, itd. Dostupni su i podložni kontrolisanju kao i svi ostali uređaji, budući da sadrže servise [7] i njihove osobine. U zavisnosti od implementacije, kontroleri zona mogu da sadrže i dodatne servise potrebne za grupnu kontrolu. Vrednosti servisa u zoni predstavljaju stanja svih uređaja u zoni. S tim u vezi, postavljanjem nove vrednosti servisa, odnosno neke njegove osobine (eng. *property*), ta nova vrednost se dodeljuje svim uređajima ili zonama sa dostupnim odgovarajućim servisom. Ukoliko je nova vrednost postavljena od strane kontrolera koji postoji u još nekim od podzona, zone sa tim istim kontrolerom neće biti izmenjene. Zone koje imaju minimalno jedan uređaj ili jednu zonu imaju osnovni kontroler (eng. *Basic Zone Controller*, BZC).

## 2.2 MQTT protokol

Komunikacija centralnog uređaja sa *cloud* servisom, kao i centralnog uređaja i klijentske aplikacije je bazirana na MQTT protokolu.

MQTT (eng. *Message Queuing Telemetry Transport*) je komunikacioni protokol koji radi po principu pretplate i objave (eng. *publish/subscribe*). Oslanja se na TCP/IP protokol. Lagan je i jednostavan za implementaciju, stoga se koristi u sistemima čiji je mrežni propusni opseg limitiran, a uređaji poseduju ograničene mogućnosti procesiranja [8]. Ovo ga čini idealnim rešenjem za komunikaciju u sistemima kućne automatizacije.

Klijent može da bude primalac (eng. *subscriber*) ili pošiljalac (eng. *publisher*), a posrednik u njihovoj komunikaciji se naziva broker. On je centralna komponenta mreže. Osnovni pojam u MQTT protokolu je „tema“ (eng. *topic*). Kada je klijent zainteresovan za određenu temu, on će se na istu pretplatiti kako bi primio poruke od interesa. Pošiljalac na određenu temu šalje poruke, koje pristižu preplaćenim klijentima. Sadržaj poruke se nalazi unutar „*payload*“ sekcije u okviru MQTT *publish* zahteva. Model MQTT poruke u OBLO sistemu prikazan je na slici ispod (Slika 2.3). Sve poruke dolaze do brokera koji ih dalje preusmerava preplaćenim klijentima. Ovim se postiže razdvajanje slanja poruka od isporuke. Omogućen je mehanizam pretplate na više od jedne teme, tako što će se one razdvojiti pomoću kose crte „/“, a brokeru će biti poslat samo jedan zahtev [9].



Slika 2.3 Struktura MQTT poruke

MQTT uvodi tri tipa kvaliteta usluge (eng. *Quality of Service*, QoS):

1. QoS 0: poruka se šalje samo jednom od strane brokera i gubitak može da se desi,
2. QoS 1: garantuje da će poruka biti dostavljena bar jednom, duplikati su mogući,
3. QoS 2: garantuje da će poruka biti dostavljena samo jednom.

Ova osobina je vrlo bitna, posebno u slučaju korišćenja mobilnih uređaja. S obzirom da su ovi uređaji pokretljivi, veoma je važno da se izbegne nepotrebna razmena poruka, uslovljena čestim prekidima veze [9].

## 2.3 *Applicaton Widget*

U ovom poglavljju su navedene neke od prednosti korišćenja *widget-a* u Android aplikacijama.

Kao prvi primer data je aplikacija za podsetnik. Kakva je svrha podsetnika ukoliko se ne nalazi na vidnom mestu? Ne može da izvrši svoju funkciju na najbolji način ukoliko je sakriven iza ikonice. Postojanje *widget-a* na početnom ekranu osigurava da informacija neće promaći. Brz pristup informacijama predstavlja značajan napredak za korisničko iskustvo (eng. *user experience*).

*Widget-i* za vreme, datum i drugi slične namene se ažuriraju čak i ukoliko u tom trenutku nisu u prvom planu, što može biti veoma korisno.

Druga prednost je to što utiču na štednju baterije. Pregršt *widget-a* pruža mogućnost da se onemogući rad nekih funkcionalnosti kojima nije lako pristupiti iz podešavanja ili trake sa obaveštenjima, i na taj način je pogodno sačuvati bateriju. Korišćenje *widget-a* podrazumeva kratkotrajan boravak na početnom ekranu, a samim tim kraću upotrebu ekrana uopšte, a aktiviran ekran je komponenta koja najviše crpi energiju mobilnog uređaja. Naravno, sa striktno tehničke

---

tačke gledišta, *widget-i* neminovno troše bateriju koristeći pozadinske procese, ali kada su informacije poput važnih mejlova, tekstualne poruke, zakazani sastanci, status baterije i slično dostupni na početnom ekranu, ta potrošnja je neuporedivo manja od ostvarene uštede.

Pored navedenog, ušteda vremena korisnika je jedna od najbitnijih stavki koje se tiču korisničkog iskustva. Uz pomoć *widget-a*, olakšan je pristup informacijama kao i kontrola određenih procesa u sastavu aplikacije kojoj *widget* pripada.

Postoje nekoliko različitih tipova *widget-a*:

- *Widget-i* koji pružaju informacije korisniku,
- Oni koji sadrže kolekcije podataka,
- *Widget-i* za kontrolu i
- Hibridni *widget-i*, kao kombinacija prethodnih [10].

## 3. Koncept rešenja

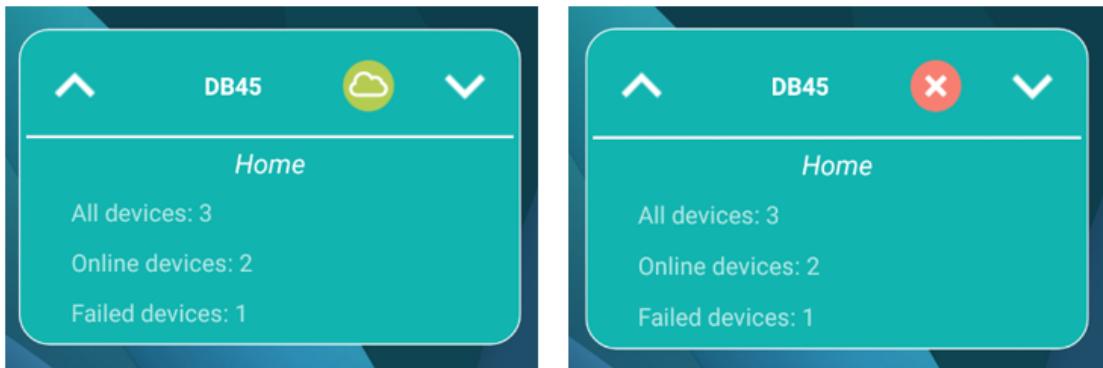
U ovom poglavlju je dat opis teme kao i kratak opis realizacije problema. Projekat se zasniva na razvoju Android *widget* aplikacije koja će podržavati neke od funkcionalnosti AZC-a na što jednostavniji način. Implementacija je odrđena nad postojećim OBLO *Home Automation* sistemom i oslanja se na trenutne API funkcionalnosti implementirane u datom sistemu.

### 3.1 Ideja realizacije

Aplikacija zahteva konfiguracioni ekran koji omogućava prijavu na korisnički nalog i odabir jednog od centralnih uređaja, koji pripadaju tom nalogu, za koji je potrebno prikazati AZC. Glavna uloga aplikacije je praćenje stanja kontrolera zone. Potrebno je implementirati prikaz svih zona definisanih na posmatranom centralnom uređaju i omogućiti njihovo listanje kako bi se za svaku od njih prikazali kontroleri sa svojim svojstvima. Stanje kontrolera bi trebalo da bude ažurno u svakom momentu, odnosno da se, ukoliko neki od uređaja promeni svoj status, novo stanje automatski prikaže na *widget*-u. S obzirom da na centralnom uređaju nije završena implementacija reagovanja na promenu nekog od svojstava kontrolera zone, nije moguće implementirati traženu ažurnost. Međutim, na drugi način je prikazano da je ažuriranje *widget*-a, pošto se desio određeni događaj, moguće.

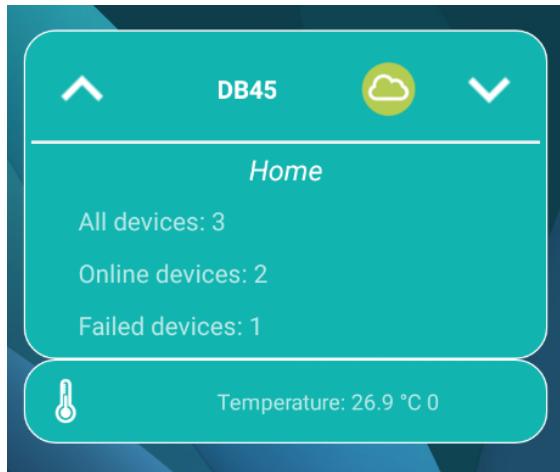
Kako bi korisnik imao brži uvid u informacije od značaja, navedene funkcionalnosti su implementirane tako da krajnji rezultati budu prikazani na *widget*-u. Korisničko iskustvo je poboljšano koristeći *widget*, s obzirom da su značajne informacije prikazane na glavnom ekranu mobilnog uređaja, te ne postoji potreba stelnog aktiviranja aplikacije.

Kao što je prikazano (Slika 3.1), na *widget*-u se nalazi ime centralnog uređaja sa kojim se korisnik povezao.



Slika 3.1 Izgled *widget-a* (slučaj kad je *gateway* aktivan i slučaj kad nije)

Pored imena nalazi se predefinisani indikator o statusu centralne jedinice. Indikator prikazuje stanja *online* ili *offline*, koja označavaju dostupnost centralnog uređaja. Postojeće zone mogu da se listaju strelicama ka gore i dole. Ispod naziva centralne jedinice, prikazana je zona, sa svojim imenom i brojem uređaja koje sadrži, uključujući i njihov status (*online* – ukoliko je uređaj aktivan i *failed* – ukoliko uređaj nije aktivan, ili se ne nalazi u dometu centralnog uređaja). Klikom na trenutno prikazanu zonu, otvara se dodatni prozor (eng. *view*) u kom su prikazani podaci dobijeni od kontrolera (Slika 3.2). U podatke spadaju sve očitane vrednosti senzora trenutne zone.

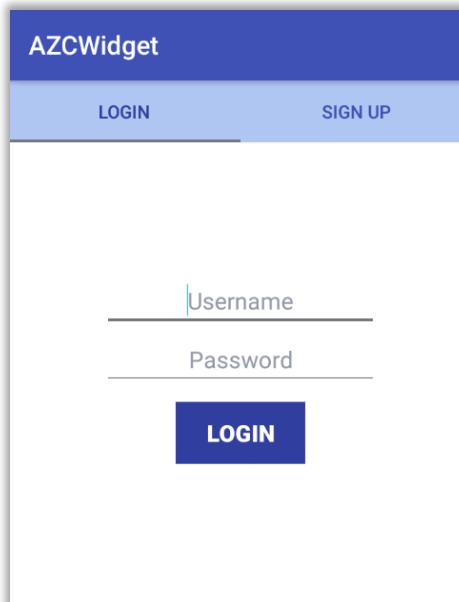


Slika 3.2 *Widget* sa podacima dobijenim od kontrolera zone

Nakon instalacije aplikacije, prilikom prvog pokretanja, korisniku se nudi mogućnost prijavljivanja u sistem (Slika 3.3). Prijavljanje se vrši unošenjem korisničkog imena i lozinke, zatim slanjem zahteva *cloud* servisu klikom na *login* dugme. Prijavljanje je neuspešno ukoliko korisnik pokuša da se prijavi na nepostojeći nalog ili ukoliko unese pogrešnu lozinku. Naravno, konekcija sa internetom mora da bude uspostavljena kako bi prijava bila uspešna.

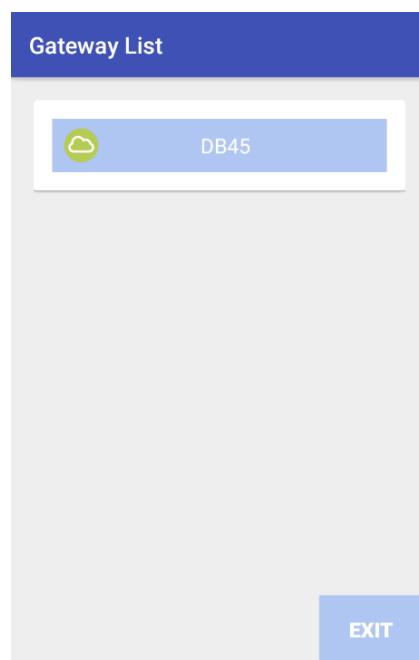
Ukoliko korisnik nema nalog, data mu je mogućnosti kreiranja novog naloga. Prilikom kreiranja novog naloga, neka od obaveznih polja koja je potrebno popuniti su: korisničko ime i

lozinka, pol, ime i prezime, saglasnost sa uslovima korišćenja i polisom privatnosti itd. Ukoliko su sva ta polja popunjena u odgovarajućem formatu, a korisnik sa datim korisničkim imenom ne postoji već u bazi korisnika, kreiranje se izvršava uspešno. Nakon toga potrebno je aktivirati nalog. Međutim, nakon pravljenja i aktivacije naloga, sve ostale kompleksnije operacije, kao što je npr. dodavanje centralnih uređaja, potrebno je obaviti putem postojeće OBLO aplikacije.



Slika 3.3 *Login screen*

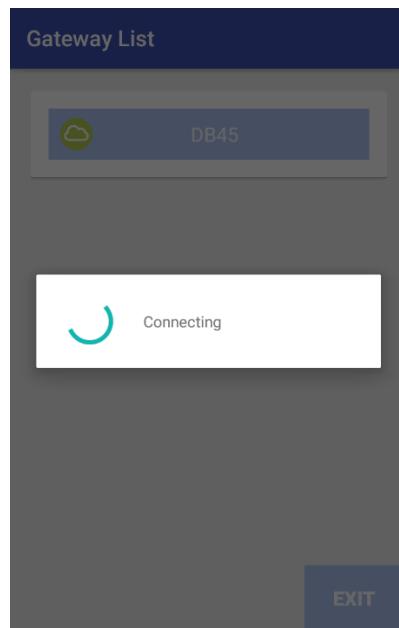
Nakon uspešnog prijavljivanja, korisniku je predstavljena lista dostupnih centralnih uređaja povezanih sa njegovim nalogom. Uređaji mogu imati status *online* ili *offline* i on je predstavljen indikacijom kao na slici (Slika 3.4).



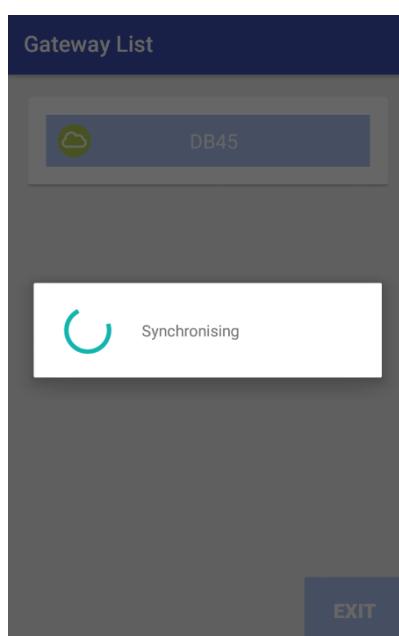
Slika 3.4 Lista centralnih uređaja

Klikom na jednu od ponuđenih centralnih jedinica uspostavlja se konekcija i sinhronizacija sa istom (Slika 3.5 i Slika 3.6), samo ukoliko je centralna jedinica aktivna. Konekcija podrazumeva povezivanje sa centralnim uređajem, ali ne uključuje dobavljanje informacija od istog. Sinhronizacija obuhvata dobavljanje svih članova zona (zone, uređaji i kontrolери), kao i potrebnih podataka o rasporedu uređaja po zonama, njihovim statusima i ostalim funkcionalnostima na centralnom uređaju.

Komunikacija sa centralnom jedinicom se izvršava korišćenjem MQTT protokola, slanjem zahteva (eng. *request*) i prijemom odgovora (eng. *response*). Zahtevi i odgovori se nalaze u *payload* segmentu MQTT poruke.



Slika 3.5 U toku je konekcija sa centralnim uređajem



Slika 3.6 U toku je sinhronizacija sa centralnim uređajem

### 3.1.1 ***Message payload***

Unutar *payload* segmenta se nalaze sve informacije namenjene jednoj ili drugoj strani (centralnoj jedinici ili mobilnoj aplikaciji). Ne postoji ograničenje po pitanju tipa podataka koji se može slati, MQTT prihvata niz bajta, dužine do 256 megabajta [9]. U slučaju AZC-a, kroz *payload* se prenose informacije kao što su očitana stanja kontrolera i podaci koje oni sami šalju. Pristigle poruke su definisane upotreboom tekstualnog sadržaja u JSON formatu, koji se zatim parsira i organizuje tako da prikazuje sve potrebne podatke na UI (eng. *User Interface*) *widget* aplikacije.

## 3.2 Arhitektura aplikacije

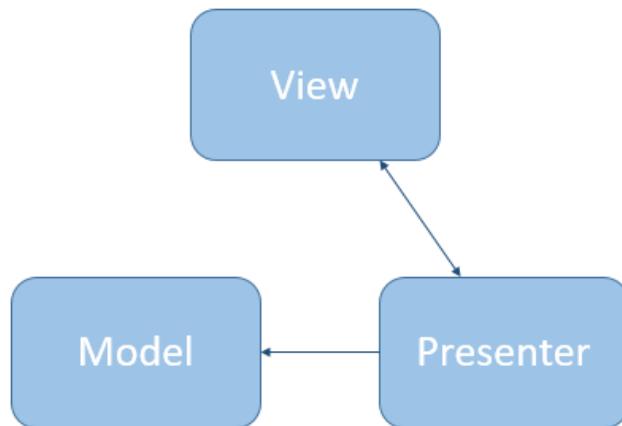
Aplikacija je podeljena u nekoliko logičkih celina, što olakšava razvoj funkcionalnosti i razumevanje implementacije. Logičke celine koje čine aplikaciju su prezentacioni sloj (eng. *presentation layer*), u kom je implementirana interakcija sa korisnikom prilikom pokretanja aplikacije, prijavljivanje na korisnički nalog i sinhronizacija sa centralnom jedinicom. Druga logička celina – *backend* sloj, predstavlja deo u kom je realizovana logika aplikacije i ona nije relevantna za korisnika. U ovoj celini su implementirani servisi, modeli u sistemu, kao i klase u kojima se pozivaju postojeće funkcije iz *third-party* biblioteka, realizovanih u okviru OBLO sistema. Neke od njih su funkcija za prijavljivanje korisnika na sopstveni nalog, kreiranje novog naloga, konekcija sa centralnim uređajem, sinhronizacija i sl. U treću celinu smeštena je realizacija widgeta.

Arhitektura aplikacije, koja ne podrazumeva i implementaciju *widget*-a, bazirana je na principu MVP šablonu.

### 3.2.1 ***MVP pattern***

*Model-View-Presenter* obrazac (Slika 3.7) se koristi u većini današnjih aplikacija i omogućava, kako pregledan i čist kod, tako i jasnu podelu logičkih celina u kodu [11]. Najviše se koristi za kreiranje korisničkog interfejsa.

- *Model* je komponenta u kojoj je implementirana poslovna (eng. *bussines*) logika aplikacije. On takođe sadrži podatke koje je potrebno prikazati korisniku.
- *View* je odgovoran za predstavljanje podataka i prihvatanje korisničkih naredbi, poželjno je da u sebi sadrži što manje logike i da predstavi podatke dobijene od prezentera. Njega najčešće implementiraju aktivnosti i fragmenti aplikacije.
- *Presenter* zahteva podatke od modela i vrši transformacije nad njima kako bi bili pogodni za prikaz korisniku. Za svaki *view* postoji *presenter*.



Slika 3.7 MVP šablon

U datom rešenju **model** aplikacije predstavljaju klase kao što su:

- Korisnik
- Centralni uređaj
- Periferni uređaj
- Zona
- Zonski kontroler
- Servis uređaja ili kontrolera
- Svojstvo servisa (*property*), itd.

**View** aplikacije sadrži logiku vezanu za implementaciju korisničke sprege, odnosno u njemu je smeštena sva interakcija sa korisnikom. S obzirom da *view* klasu implementiraju uglavnom aktivnosti i fragmenti, u aplikaciji su realizovane dve ovakve aktivnosti. Prva, koja predstavlja *login* aktivnost i druga, koja predstavlja *gateway list* aktivnost. Prvi *view* služi da se *presenter* klasi šalje korisnički zahtev za prijavu na nalog (pritiskom na *login* dugme), a potom se obaveštava korisnik o uspešnosti prijave ili eventualnim greškama prilikom iste. U drugoj aktivnosti se korisniku prikazuje lista dostupnih centralnih uređaja. Nakon toga, šalje se komanda *presenter* klasi radi konekcije i sinhronizacije sa centralnim uređajem usled pritiska na jedan od ponuđenih uređaja. Takođe se korisniku prikazuje tok pomenutih akcija.

**Presenter** aplikacije ne sme da ima dodir sa logikom korisničke sprege. Njegova uloga je dobavljanje podataka od modela, koje je potrebno prikazati korisniku. U dатој aplikацији он комуникара са сервисима преко којих добија податке које пружа model.

## 4. Programsко rešenje

Rešenje je realizovano korišćenjem *Java* programskog jezika u *Android Studio* razvojnog okruženju.

### 4.1 Primer MVP šablon

MVP šablon se u datom rešenju koristi radi kreiranja korisničkog interfejsa. Pokretanjem aplikacije, interakcija sa korisnikom se svodi na prijavu na korisnički nalog, a potom na odabir centralnog uređaja. Klase potrebne za ovaku implementaciju su *Contract*, *Activity* i *Presenter*.

```
class HomeContract {  
    interface View {  
        void completeLoginView();  
        void userExists(boolean exist);  
        void userCreated();  
        void showErrorMsg(String msg);  
        void loginError(Throwable e);  
    }  
    interface Presenter {  
        void setView(HomeContract.View view);  
        void loginUser(String username, String password);  
        void checkIfUserExists(String username);  
        void createUser(ObloUser obloUser);  
    }  
}
```

Slika 4.1 *HomeContract* klasa

*Contract* klasa služi da spoji interfejs od *view-a* i interfejs od *presenter-a* u jedno, kao što je prikazano na primeru *HomeContract* klase (Slika 4.1).

U klasi *HomeActivity* su implementirane sve metode iz *view* interfejsa. Njihova uloga je da obaveste korisnika da li je nalog kreiran uspešno ili ne (u slučaju pravljenja novog naloga), takođe da li je prijavljivanje na nalog uspešno obavljen ili prikaz poruke, ukoliko je došlo do neke greške.

U *HomePresenter* klasi implementirane su metode u kojima se komunicira sa poslovnom logikom aplikacije. Neke od metoda su *loginUser*, *createUser* i *checkIfUserExists*, koje šalju zahtev za prijavu na nalog, kreiraju novi nalog ili proveravaju da li korisnik pod tim imenom već poseduje nalog, respektivno.

Koristeći ovaj šablon, realizovan je prikaz liste centralnih uređaja i odabir jednog od njih. *GatewayListContract* klasa data je na slici ispod.

```
class GatewayListContract {

    interface View {

        void renderGatewayListView(List<ObloGateway> gateways);

        void gatewayIsConnected();

        void gatewayConnecting();

        void gatewaySynchronising();
    }

    interface Presenter {

        void setView(GatewayListContract.View view);

        void refreshGatewayList();

        void connectGateway(ObloGateway obloGateway);

        void start();

        void stop();
    }
}
```

Slika 4.2 *GatewayListContract* klasa

Metoda *renderGatewayList* služi da prikaže listu centralnih uređaja povezanih sa trenutnim nalogom, *gatewayConnecting* i *gatewaySynchronizing* obaveštavaju korisnika da je u toku konekcija i sinhronizacija sa centralnim uređajem. Nakon poziva metode *gatewayIsConnected* završava se operacija sinhronizacije, i gasi se trenutno prikazana

aktivnost, nakon čega je korisnik spremjan da prikaže *widget* na ekranu svog mobilnog uređaja. Sve ove metode implementirane su u klasi *GatewayListActivity*.

U metodama *start* i *stop* se pretplaćuje (eng. *subscribe*) na pristizanje događaja, koji su od značaja prilikom konekcije i sinhronizacije, od strane centralnog uređaja, odnosno prestaje dobavljanje istih. Metoda *refreshGatewayList* osvežava listu prikazanih centralnih jedinica, a *connectGateway* šalje zahtev za povezivanje sa centralnom jedinicom. Ove metode takođe služe za interakciju sa poslovnom logikom i implementirane su u *GatewayListPresenter* klasi.

Internim događajima u aplikaciji se rukuje koristeći *RxJava* biblioteku o kojoj će biti reči u sledećem poglavlju. Interni događaji su u principu odgovori na zahteve poslate ka *business* logici.

## 4.2 RxJava

*RxJava* je biblioteka koja olakšava implementiranje principa reaktivnog programiranja u Androidu [12]. Reaktivno programiranje je paradigma programiranja, isto kao što je funkcionalno ili imperativno programiranje. Bazira se na kreiranju, transformisanju i reagovanju na tok podataka te propagiranju promena koje uzrokuje tok podataka [13]. Tok podataka opisuje stanje u kojem se program nalazi. Zahvaljujući ovome omogućeno je asinhrono programiranje na mnogo jednostavniji način nego uz korišćenje *callback* funkcija, jer gomilanje takvih funkcija dovodi do teškog razumevanja napisanog koda.

Tok podataka možemo zamisliti kao polje koje je, za razliku od uobičajenog polja čiji su elementi razdvojeni memorijom, razdvojeno vremenom. Sam tok će tokom vremena emitovati više vrednosti. Direktna implementacija toka podataka jeste klasa *Observable*, koja se može shvatiti kao entitet koji se posmatra tokom vremena i koji odašilje vrednosti toka podataka. Te vrednosti se šalju entitetima koji su pretplaćeni na njega. Oni se nazivaju *Observer*-i, i definišu metode *onNext*, *onError* i *onComplete*, koje se pozivaju u zavisnosti od toga kako *Observable* emituje podatke [13].

Više *Observable*-a je moguće ulančavati, praveći lanac istih, a to se postiže koristeći operatore koji su implementirani unutar *RxJava* biblioteke. Operator je funkcija kojom je omogućeno funkcionalno programiranje nad tokom podataka. Svaki operator prima ulazni *Observable* i vraća modifikovani izlazni *Observable*. Neki od najčešće korišćenih operatora su *map* i *filter* [13].

Kao što je već rečeno, *RxJava* biblioteka je korišćena kako bi se reagovalo na interne događaje unutar aplikacije. Ti događaji nastaju kao rezultati različitih slučajeva korišćenja (eng. *use case*) kao što su:

- Prijava na nalog,

- Kreiranje novog naloga,
- Provera da li korisnik sa datim nalogom postoji,
- Dobavljanje liste raspoloživih centralnih uređaja,
- Reagovanje na promene na centralnom uređaju,
- Reagovanje na promene na perifernim uređajima.

U nastavku su prikazane klase i metode koje realizuju slučaj korišćenja – reagovanje na promene na centralnom uređaju, ujedno je i prikazan primer upotrebe *RxJava* biblioteke.

```
    @Override
    public void start() {
        Log.d(TAG, "start: ");
        getAndUpdateGatewayList_();
        subscribeToGatewayEvents();
    }
}
```

Slika 4.3 *GatewayListPresenter*: *start* metoda

U *start* metodi klase *GatewayListPresenter* poziva se funkcija *subscribeToGatewayEvents* (Slika 4.3). U njoj se kreira željeni slučaj korišćenja (Slika 4.4 i Slika 4.5) nad kojim se poziva metoda *execute* (Slika 4.6) koja kao parametar prima *Observer* objekat (Slika 4.8) koji se pretplaćuje na emitovanje događaja na centralnom uređaju. U klasi koja predstavlja taj objekat se implementira metoda *onNext*, kako bi se u datom momentu mogao obraditi svaki događaj koji pristigne sa centralnog uređaja.

```
private void subscribeToGatewayEvents() {
    subscribeToGatewayEventsUC = new SubscribeToGatewayEvents();
    subscribeToGatewayEventsUC.execute(new GatewayEventsSubscriber());
}
```

Slika 4.4 *subscribeToGatewayEvents* metoda

```
public class SubscribeToGatewayEvents extends UseCase<GatewayEvent> {

    @Override
    protected Observable<GatewayEvent> buildUseCaseObservable() {
        return EventManager.getInstance().observeEvents(GatewayEvent.class);
    }
}
```

Slika 4.5 *UseCase*: pretplata na događaje centralnog uređaja

```

    ...
    public UseCase<T> execute(ObloDefaultSubscriber<T> useCaseSubscriber) {
        this.mSubscription = this.buildUseCaseObservable()
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribeWith(useCaseSubscriber);
        return this;
    }
}

```

Slika 4.6 *UseCase: execute* metoda

Događaji emitovani od strane centralnog uređaja u ovom slučaju predstavljaju tok podataka, *Observable*. Neki od mogućih događaja su: *connecting*, *synchronising*, *connected\_cloud* i *connection\_lost*.

Prototip klase *Observer*-a data je na slici ispod:

```

    ...
    public abstract class ObloDefaultSubscriber<T> extends DisposableObserver<T> {

        @Override public void onComplete() {

        }

        @Override public void onError(Throwable e) {

        }

        @Override public void onNext(T t) {

        }
    }
}

```

Slika 4.7 Prototip preplatnika (*Observer*, odnosno *Subscriber*)

```

private class GatewayEventsSubscriber extends ObloDefaultSubscriber<GatewayEvent> {
    @Override
    public void onNext(GatewayEvent gatewayEvent) {
        processGatewayEvent(gatewayEvent);
    }
}

```

Slika 4.8 Preplatnik na *gateway* događaje

### 4.3 Komunikacija sa *cloud servisom*

U tabeli ispod (Tabela 4.1) prikazane su važne metode pomoću kojih se komunicira sa *cloud servisom*, koje su takođe implementirane korišćenjem *RxJava* biblioteke, nalaze se u klasi *CloudManager*, a povratna vrednost im je tipa *Observable*, te su za njih implementirani slučajevi korišćenja i preplatnici na emitovan tok podataka.

Metoda:	Objašnjenje:
<code>loginUser(String username, String password)</code>	Šalje zahtev <i>cloud</i> servisu za prijavu na korisnički nalog
<code>validateUser(String username)</code>	Proverava na <i>cloud</i> -u da li korisnik sa datim korisničkim imenom već postoji
<code>createUser(ObloUser obloUser)</code>	Kreira novog korisnika
<code>getGatewayListFromCloud()</code>	Dobavlja listu centralnih uređaja <i>cloud</i> -a za dati korisnički nalog

Tabela 4.1 Metode za komunikaciju sa *cloud* servisom

#### 4.4 Komunikacija sa centralnim uređajem

Metode u kojima je realizovana komunikacija sa centralnom jedinicom, preko MQTT protokola, implementirane su u klasi *MqttManager* i prikazane su ispod (Tabela 4.2).

Metoda:	Objašnjenje:
<code>connectAndSynchronize(ObloGateway gateway)</code>	Šalje zahtev centralnom uređaju za konekciju i sinhronizaciju sa istim. Centralni uređaj prelazi u stanje <i>connecting</i> , postavlja se aktivan <i>gateway</i> i tip konekcije. Po okidanju <i>callback</i> funkcije <i>onConnected</i> , <i>gateway</i> prelazi u stanje <i>connected</i> i pozivaju se metode za sinhronizaciju, nakon čega je u stanju <i>synchronized</i> .
<code>getZoneLayout()</code>	Metoda koja se poziva prilikom sinhronizacije. Predstavlja dobavljanje svih članova zona (zone, uređaji i kontroleri).
<code>getDeviceDetailedList()</code>	Dobavlja listu svih uređaja povezanih sa centralnim uređajem.
<code>getControllerServiceDetailed(int controllerID)</code>	Za kontroler, čiji je identifikacioni broj prosleđen kao parameter, dobavlja se lista svih servisa (temperaturni servis, servis koji detektuje pokret, vlažnost itd).

Tabela 4.2 Metode za komunikaciju sa centralnim uređajem

Podaci koji se razmenjuju i sa *cloud* servisom i sa centralnim uređajem, šalju se u JSON formatu [14]. JSON format podataka koristi tekstualne datoteke za prenos objekata. Objekti mogu da poseduju dve strukture:

- Skup parova ime-vrednost: koristi se za prenos objekata realizovanih kao struktura, heš tabela, lista s ključevima, itd.
- Uređena lista vrednosti: koristi se za prenos objekata realizovanih kao niz, lista, sekvenca, itd.

U primeru JSON datoteke (Slika 4.9) prikazan je zahtev da se od centralne jedinice dobavi skup svih servisa koji pripadaju kontroleru sa identifikacionim brojem 1031.

```
/MqttMessenger: PUBLISHED message :
{
    "name": "get_service_list_detailed",
    "sender": "cli/867811026014029",
    "type": "command",
    "params": {
        "id": 1031
    },
    "uid": 490673938
}
```

Slika 4.9 Primer zahteva poslatog u vidu JSON objekta

Objekat dobijen u JSON formatu je potrebno prebaciti u *custom* objekat, tj. izvršiti parsiranje, u zavisnosti od toga koji tip objekta je stigao. Takođe je potrebno svaki objekat koji se šalje, prebaciti u JSON objekat.

## 4.5 Servisi

Kako bi pristup podacima, odnosno reagovanje na promene stanja centralnog i perifernih uređaja, bilo moguće i nakon zatvaranja aktivnosti aplikacije, u datom rešenju su implementirani servisi koji će nakon pokretanja aplikacije biti konstantno aktivni, i u njima će se izvršavati svi pozivi funkcija usmereni ka centralnom uređaju ili *cloud*-u. Servis je komponenta aplikacije koja ne nudi korisnički interfejs i služi za izvršavanje vremenski dužih operacija kao pozadinskih procesa [15].

U aplikaciji postoje dva servisa, *MqttService* i *StratusRestService* i implementirani su tako da se komponente mogu povezati na njih, što znači da će da budu aktivni dok god su aktivne te komponente. U ovom slučaju *widget* će, pored aktivnosti aplikacije, da se poveže na date servise.

U servisu je potrebno implementirati *callback* metodu *onBind*, koja vraća *IBinder* objekat (Slika 4.10) koji definiše programski interfejs koji klijent koristi kako bi komunicirao sa servisom. Potrebno je i napraviti klasu, koja nasleđuje *Binder*, unutar servisa u kojoj se obavlja implementacija metoda.

```
@Nullable
@Override
public IBinder onBind(Intent intent) {
    Log.d(TAG, msg: "onBind: ");
    return new MqttBinder();
}
```

Slika 4.10 Primer *onBind* metode u *MqttService* klasi

Klase u kojoj se vrši povezivanje na servis implementira interfejs *ServiceConnection*, u čijoj *callback* metodi *onServiceConnected* dobijeni parametar tipa *IBinder* predstavlja *binder* objekat servisa sa kojim se vrši povezivanje (Slika 4.11). Pomoću tog objekta je omogućena komunikacija sa servisom i pozivanje njegovih metoda.

```
@Override
public void onServiceConnected(ComponentName componentName, IBinder iBinder) {
    if (componentName.getClassName().contains(OBLO_MQTT_SERVICE)) {
        mMqttBinder = (ObloMqttService.MqttBinder)iBinder;
    } else if (componentName.getClassName().contains(STRATUS_REST_SERVICE)) {
        mStratusRestBinder = (StratusRestService.StratusRestBinder)iBinder;
    }
}
```

Slika 4.11 *onServiceConnected* metoda

*Widget* komponenta se povezuje na servise u svojoj *onEnabled* *callback* metodi, koja se poziva pri postavljanju prve instance *widget*-a na početni ekran (Slika 4.12).

```
@Override
public void onEnabled(Context context) {
    Log.d(TAG, msg: "onEnabled: ");
    WidgetSingleton.getInstance().setContext(context);
    WidgetSingleton.getInstance().bindToServices();
    super.onEnabled(context);
}
```

Slika 4.12 *onEnabled* metoda u *widget* klasi

S obzirom da je *widget* objekat tipa *AppWidgetProvider*, što predstavlja klasu koja nasleđuje *BroadcastReceiver*, nije ga moguće direktno povezati na servise. Zbog toga se uvodi *singleton* klasa, o kojoj će biti više reči u narednom poglavljju, kako bi se preko nje *widget* povezao na servise.

Kao što je u *GatewayListPresenter* klasi realizovana pretplata na događaje centralnog uređaja, tako je u *MqttService* odradena pretplata na događaje perifernih uređaja i centralnog uređaja, kako bi se na njih moglo reagovati i nakon gašenja aplikacije i prelaska na *widget*.

## 4.6 **Widget**

Da bi se kreirao *widget*, potreban je *AppWidgetProviderInfo* objekat, koji sadrži osnovne podatke kao što su *layout widget-a*, učestanost ažuriranja, *AppWidgetProvider* klasa itd [9]. Ovi podaci se definišu u XML datoteci. Potrebna je implementacija *AppWidgetProvider* klase, što predstavlja glavnu klasu koja definiše osnovne metode koje omogućavaju interakciju sa *widget-om* koja se bazira na emitovanju događaja, s obzirom da je *AppWidget BroadcastReceiver* komponenta. Pomoću ove klase reaguje se na događaje kao što su ažuriranje *widget-a* (eng. *onUpdated*), kreiranje njegove prve instance (eng. *onEnabled*), brisanje jedne od instanci (eng. *onDeleted*) i brisanje poslednje instance (eng. *onDisabled*). Potrebna je i definicija XML *layout* datoteke koja definiše izgled samog *widget-a*. U *AndroidManifest.xml* datoteci treba deklarisati novu *AppWidgetProvider* klasu kao *receiver* komponentu.

*Widget* u datom rešenju je hibridni *widget*, sadrži skup podataka koje se vremenom ažuriraju i daju neke informacije.

Kako bi samo jedna zona u datom momentu bila prikazana na početnom ekranu, kao kontejner zona koristi se *AdapterViewFlipper*, koji poput liste, steka i sl. služi za prikaz skupa podataka. *AdapterViewFlipper* koristi adapter koji služi za povezivanje individualnih elemenata neke kolekcije sa njima predodređenim elementima korisničkog interfejsa. U slučaju *AppWidget-a*, adapter je zamenjen sa *RemoteViewsFactory* klasom koja predstavlja omotač oko adapter interfejsa. Kada se zahteva prikaz određenog elementa iz skupa, *RemoteViewsFactory* kreira i vraća element kao *RemoteViews* objekat. U cilju dodavanja prikaza skupa podataka u *widget*, potrebno je još implementirati klasu *RemoteViewsService*, čija je svrha da dozvoli adapteru da zatraži *RemoteViews* objekat.

S obzirom da je zadatak da se za svaku zonu prikaže niz očitavanja jedne od osobina servisa za svaki od kontrolera, i ovi podaci moraju biti smešteni u neku listu, i u tu svrhu se koristi *ListView* kao skup elemenata. Za ovu listu je takođe neophodno implementirati *service* i *factory* klase.

### 4.6.1 **AppWidgetProvider** klasa

Predstavlja neophodnu klasu pri implementaciji *widget-a*. U datom rešenju to je klasa *AZCAppWidget*. Ona nasleđuje *AppWidgetProvider* klasu i implementira potrebne metode.

Najvažnije metode su:

- *onUpdate* – poziva se pri kreiranju *widget-a* i nakon isteka vremenskog perioda nakon kog je potrebno ažurirati *widget*, što u datom rešenju nije navedeno, s obzirom da se *widget* ažurira nakon promene stanja uređaja, a ne u tačno određeno vreme,
- *onReceive* – poziva se svaki put kad se dogodi akcija na *widget-u* i u zavisnosti od iste, obavljaju se određene aktivnosti koje pripadaju toj akciji, bilo da je ažuriranje widgeta, pokretanje neke aktivnosti, itd.
- *onDelete* – poziva se svaki put kad se neka od instanci *widget-a* ukloni sa početnog ekrana,
- *onEnabled* – poziva se kad se prva instanca postavi na ekran,
- *onDisabled* – poziva se kad se poslednja instanca ukloni sa ekran-a.

U metodi *onUpdate* se za svaku instancu *widget-a* pozove funkcija *updateAppWidget* u kojoj se kreira korisnički interfejs za *widget*, postavlja adapter za listu zona i kreiraju se *PendingIntent*-i za pritisak na gore/dole strelice i elemenat liste zona. Postavljanje adaptera podrazumeva pokretanje *ZonesRVService* klase (*RemoteViewsService* klasa o kojoj je bilo reči u poglavlju 4.6). Pomoću *PendingIntent*-a se definiše akcija na neki od datih događaja i ta akcija se obrađuje u *onReceive* metodi. U nastavku su dati primeri za pritisak na strelicu ka dole kako bi se prikazala sledeća zona.

```
views.setOnClickListener(R.id.widget_next_button, showNextItemPendingIntent(context, appWidgetId));
```

Slika 4.13 *onUpdate* metoda: postavlja se *PendingIntent* za određeno dugme

```
private PendingIntent showNextItemPendingIntent(Context context, int id) {
    final Intent intent = new Intent(context, AZCAppWidget.class);
    intent.setAction(NEXT_ACTION);
    intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, id);
    return PendingIntent.getBroadcast(context, id, intent, PendingIntent.FLAG_UPDATE_CURRENT);
}
```

Slika 4.14 Kreiranje *PendingIntent*-a za datu akciju

```

String action = intent.getAction();
RemoteViews remoteViews = new RemoteViews(context.getPackageName(), R.layout.azcapp_widget);
int widgetID = intent.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, AppWidgetManager.INVALID_APPWIDGET_ID);
WidgetSingleton.getInstance().setWidgetID(widgetID);

switch (action) {
    case NEXT_ACTION:
        Log.d(TAG, msg: "onReceive: NEXT ");
        remoteViews.showNext(R.id.adapter_view_flipper);
        remoteViews.setViewVisibility(R.id.invisible_list, View.GONE);
        AppWidgetManager.getInstance(context).updateAppWidget(widgetID, remoteViews);
        break;
}

```

Slika 4.15 *onReceive* metoda: definisanje akcije

U *onEnabled* metodi se *widget* povezuje na *MqttService* i *StratusRestService* servise. S obzirom da je *widget BroadcastReceiver* komponenta, nije moguće da se direktno poveže na servis, jer postoji opcija da neće postojati dok se izvršavaju metode pozvane iz servisa. Stoga je neophodno napraviti *singleton* klasu koja će nakon kreiranja postojati uvek, dok god se aplikacija ne ugasi potpuno, zbog čega se u *onEnabled* metodi poziva metoda *bindToServices* iz *WidgetSingleton* klase. S obzirom da se u *singleton* klasi povezuje na servise, u njoj se dobavljuju *binder* objekti za svaki od servisa, preko kojih se *widget* klasa snabdeva podacima, kao što su lista zona, uređaji i kontroleri.

U *onDisabled* metodi se poziva funkcija za zaustavljanje pomenutih servisa koja je takođe definisana u *singleton* klasi. Neposredno pre zaustavljanja servisa, poziva se metoda *unsubscribeFromEvents* koja prekida pretplatu na događaje uređaja.

#### **4.6.2 *RemoteViewsService* i *RemoteViewsFactory* klase**

*Widget* pokreće *RemoteViewsService* servis onda kada je potrebno popuniti listu ili neku drugu kolekciju željenim elementima. *ZonesRVService* se pokreće pri kreiranju *widget-a*, a *ControllersRVService* se pokreće pritiskom na elemenat zone. U svojoj metodi *onGetViewFactory* servis kao povratnu vrednost ima objekat tipa *RemoteViewsFactory*, što je objašnjeno u poglavlju 4.6. U datom rešenju adapteri za liste zona i liste servisa kontrolera su predstavljeni klasama *ZonesRVFactory* i *ControllersRVFactory*. Metode koje je potrebno implementirati su:

- *onCreate* – što se tiče liste zona, u ovoj metodi se preko *singleton* klase dobavljuju lista zona i lista uređaja kako bi se odredilo koliko ima aktivnih, odnosno neaktivnih uređaja. U slučaju kreiranja liste servisa kontrolera u zoni, u ovoj metodi se inicijalizuju imena servisa svih kontrolera,
- *getCount* – metoda koja vraća broj elemenata datog skupa,
- *getViewAt* – za svaki element liste, u zavisnosti od toga na kojoj poziciji u listi se nalazi, postavlja podatke koji će biti prikazani korisniku. Za listu zona, u ovoj

metodi se određuje broj aktivnih i neaktivnih uređaja, i kreira se *Intent* koji zajedno sa *PendingIntent*-om iz *onUpdate* metode definiše akciju nakon pritiska na određeni element liste zona. Taj *Intent* je neophodan zato što nije dozvoljeno koristiti samo *PendingIntent* za pritisak na elemenat neke kolekcije, kao što je na primer dozvoljeno za pritisak na dugme. Ovaj novi *Intent* se definiše kao „dopunjajući“. Što se tiče liste servisa kontrolera, u ovoj metodi se za servis na određenoj poziciji postavlja ime njegovog svojstva (pod svojstvom se misli da li se radi o temperaturnom servisu, servisu koji meri nivo vlažnosti, itd.) zajedno sa vrednošću i odgovarajućom ikonicom koja predstavlja o kom svojstvu je reč,

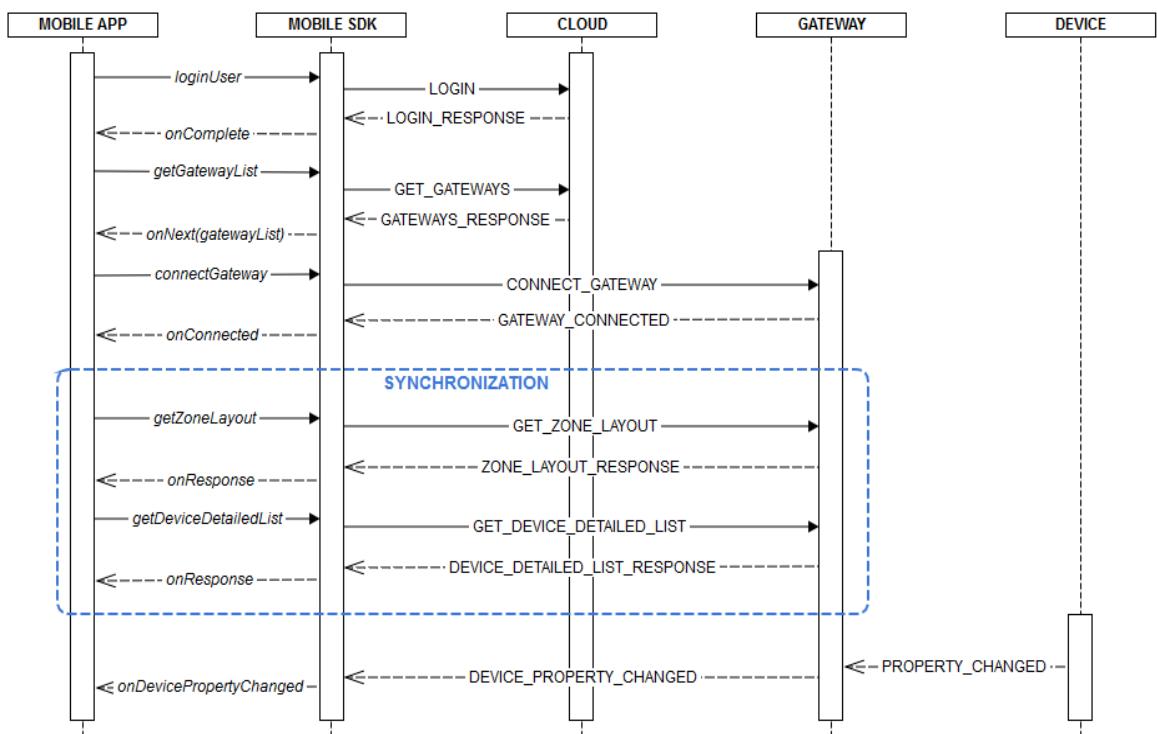
- *onDataSetChanged* – metoda koja je definisana samo za listu servisa kontrolera, s obzirom da oni predstavljaju podatke koje je neophodno ažurirati u toku vremena. U njoj se ponovo, za odgovarajuću zonu, dobavljaju kontroleri sa svim svojim servisima i njihovim svojstvima. Potom se ponovo poziva metoda *getViewAt* za svaki od kontrolera u listi, kako bi se prikaz korisniku promenio u zavisnosti od izmenjenih podataka.

## 5. Testiranje i rezultati

Testiranje aplikacije izvršeno je na Android mobilnom telefonu i tablet uređaju. Obavljen je samo funkcionalno testiranje, jer testiranje brzine izvršenja direktno zavisi od SDK-a koji se koristi za komunikaciju. Samim tim, sva merenja bi bila kompromitovana bilo kojim lošim ponašanjem SDK-a.

### 5.1 Opis testnog sistema

Na slici ispod (Slika 5.1) dat je MSC dijagram u kom je prikazan tok izvršavanja testirane aplikacije.



Slika 5.1 MSC dijagram testnog sistema

## 5.2 Funkcionalno testiranje

U slučajeve koje je bilo neophodno testirati spadaju:

- *Login*, odnosno prijava na korisnički nalog
- *Sign up*, odnosno kreiranje novog naloga
- Dobavljanje liste centralnih uređaja i prikaz iste
- Konekcija i sinhronizacija sa centralnim uređajem
- Instalacija *widget* komponente
- Prikaz željenih podataka na *widget*-u i ažuriranje istog

### 5.2.1 Prijava na korisnički nalog

Prilikom prijavljivanja na korisnički nalog, nakon pritiska *login* dugmeta, poziva se metoda *loginUser*. Testirani su slučajevi kada:

- 1) konekcija sa internetom nije uspostavljena
- 2) korisnik pokušava da se prijavi sa pogrešnom lozinkom ili pogrešnim korisničkim imenom
- 3) korisnik pokušava da se prijavi na nalog koji ne postoji
- 4) konekcija sa internetom je uspostavljena i korisnik pokušava da se prijavi sa validnim imenom i lozinkom

### 5.2.2 Kreiranje novog korisničkog naloga

Prilikom kreiranja novog naloga, potrebno je nakon popunjениh odgovarajućih polja pritisnuti *sign up* dugme, nakon čega sledi provera da li korisnik sa datim korisničkim imenom već postoji, verifikacija korisnika, a potom kreiranje naloga.

Testiranje kreiranja novog naloga obavljeno je u nekoliko koraka:

- 1) pokušaj kreiranja naloga sa već postojećim korisničkim imenom
- 2) pokušaj kreiranja naloga sa nevalidnim podacima (korisničko ime, lozinka itd nisu uneti po tačno definisanom šablonu)
- 3) pokušaj kreiranja ukoliko nisu uneta sva neophodna polja (korisničko ime, lozinka, saglasnost sa uslovima korišćenja i polisom privatnosti itd)
- 4) pokušaj kreiranja naloga sa validnim podacima i popunjениm svim neophodnim poljima

### 5.2.3 Lista centralnih uređaja

Centralni uređaji se ispravno prikazuju u listi nakon prijave na nalog. Poziv metode za dobavljanje uređaja sledi odmah nakon prijave na nalog, prilikom kreiranja nove aktivnosti.

### 5.2.4 Konekcija i sinhronizacija

Testiran je slučaj povezivanja sa uređajem i sinhronizacija sa istim dok je uspostavljena konekcija sa internetom.

### 5.2.5 Instalacija *widget* komponente

Testirano je ponašanje *widget*-a:

- 1) ukoliko se njegova instanca postavi na početni ekran pre pokretanja aplikacije
- 2) ukoliko se postavi nakon pokretanja aplikacije, odnosno nakon povezivanja i sinhronizacije sa centralnom jedinicom
- 3) ukoliko se ne obriše prethodna instanca, a aplikacija se pokrene ispočetka
- 4) ukoliko se postavi više instanci na početni ekran, tokom jedne sesije

### 5.2.6 Prikaz podataka na *widget*-u i ažuriranje

Prikaz podataka na *widget*-u je testiran u zavisnosti od toga kada je *widget* postavljen na početni ekran. Ažuriranje *widget*-a je implementirano na promenu stanja nekog od uređaja povezanih na centralnu jedinicu. Testiranje ažurnosti je sprovedeno promenom stanja uređaja. S obzirom da je samo jedna zona (*home* zona) posedovala jedan kontroler (sa temperaturnim servisom), manipulacija stanjem uređaja izvršavana je hladjenjem, odnosno zagrevanjem jednog uređaja u zoni.

## 5.3 Rezultati

Prijava na korisnički nalog uspešno je izvršena samo u četvrtom slučaju, navedenom u poglavlju 5.2.1. U svim ostalim slučajevima korisniku je prikazano upozorenje na ekranu u kom je naveden razlog zbog čega nije moguće izvršiti prijavu.

Kreiranje novog korisničkog naloga izvršava se uspešno takođe samo u četvrtom slučaju (poglavlje 5.2.2), kada su uneti svi validni podaci. U ostalim slučajevima korisniku je prikazano upozorenje zbog čega nije moguće kreirati nalog.

Dok je uspostavljena konekcija sa internetom, uspešno se izvršava povezivanje sa centralnim uređajem i sinhronizacija. Slučaj povezivanja kada veza sa internetom nije uspostavljena nije implementiran, pa je samim tim i neuspešan. Prilikom sinhronizacije dobavljeni su svi podaci od značaja koje je dalje potrebno prikazati na *widget*-u.

Ukoliko se instanca *widget*-a postavi na ekran nakon instalacije, a pre pokretanja aplikacije, na njemu neće biti prikazano ništa od podataka. Kako bi validni podaci bili prikazani, potrebno je prvo pokrenuti aplikaciju, obaviti prijavu, potom konekciju i sinhronizaciju sa centralnom jedinicom. Nakon toga je neophodno odabrati odgovarajući *widget* iz liste podržanih *widget*-a na mobilnom uređaju. Ukoliko se instanca *widget*-a ne ukloni sa početnog

ekrana, a aplikacija se pokrene ispočetka, na *widget*-u ostaju stari podaci. Pritom, centralna jedinica se prikazuje kao neaktivna, a pretplata na događaje perifernih uređaja prestaje. Potrebno je instancirati novu komponentu koja je funkcionalna. Ukoliko se tokom jedne sesije postavi više instanci na početni ekran, sve su funkcionalne i stanja im se menjaju u zavisnosti od toga kojom instancom se rukuje.

*Widget* reaguje na promenu stanja centralne jedinice. Ukoliko ista postane neaktivna, indikator o njenom statusu prikazuje stanje offline. Do ovoga može doći ukoliko se npr. izgubi konekcija sa internetom. Nije realizovan slučaj kada se konekcija uspostavi ponovo i uređaj postane aktivran. Tada je potrebno pokrenuti aplikaciju ispočetka.

Klikom na određenu zonu prikazuje se lista servisa kontrolera unutar nje. Po specifikaciji ovog rada, vrednost svojstava tih servisa bi trebala u svakom momentu biti ažurna, međutim ovu akciju nije bilo moguće implementirati s obzirom da na centralnom uređaju nije realizovano rukovanje događajima kontrolera. Umesto toga, kako bi se pokazalo ažuriranje *widgeta*, uvedeno je reagovanje na promene stanja uređaja i jedan brojač, koji se na *widget*-u prikazuje i njegova vrednost se povećava svaki put kad uređaj date zone promeni vrednost svojstva.

Prilikom uklanjanja poslednje instance *widget-a* sa početnog ekrana, zaustavljaju se servisi u aplikaciji i moguće je ponovo pokrenuti aplikaciju i ponoviti prijavu na određeni nalog.

Na Android verziji 5.0 nije potrebno dodeliti ovlašćenje za pristup stanju telefona prilikom prvog pokretanja aplikacije. Međutim, na verzijama iznad te je potrebno to učiniti u podešavanjima aplikacija.

U tabeli ispod (Tabela 5.1) dati su funkcionalni testovi na različitim uređajima.

Slučaj	Telefon (Android 5.0)	Telefon (Android 6.0)	Telefon (Android 7.0)	Tablet (Android 8.0)
Prijava na korisnički nalog	✓	✓	✓	✓
Kreiranje novog naloga	✓	✓	✓	✓
Dobavljanje i prikaz liste centralnih uređaja	✓	✓	✓	✓
Konekcija sa centralnim uređajem	✓	✓	✓	✓

Sinhronizacija sa centralnim uređajem	✓	✓	✓	✓
Instalacija <i>widget</i> komponente	✓	✓	✓	✓
Prikaz podataka na <i>widget</i> -u	✓	✓	✓	✓
Ažuriranje podataka	✓	✓	✓	✓

Tabela 5.1 Funkcionalni testovi

## 6. Zaključak

U ovom radu je opisana implementacija *widget* aplikacije za Android sa ciljem očitavanja stanja senzora u zonama pametne kuće. Prednost upotrebe *widget*-a je poboljšano korisničko iskustvo u cilju uštede vremena korišćenja aplikacije, jer su informacije direktno dostupne na početnom ekranu.

Korisnička aplikacija komunicira sa centralnim uređajem koji predstavlja mozak sistema pametne kuće i njegova uloga je nadzor i kontrola perifernih uređaja. U rešenju, oni komuniciraju posredstvom *cloud* servisa. Implementirano je da se korišćenjem mobilne aplikacije obavi sinhronizacija sa centralnim uređajem. Time se postiže dobavljanje informacija od značaja sa centralnog uređaja kao što su zone u sistemu pametne kuće, zonski kontroleri, kao i uređaji raspoređeni po tim zonama. Neke od ovih informacija prikazane su na *widget*-u.

Moguće unapređenje može da bude realizacija komunikacije između centralnog uređaja i aplikacije u lokalnoj mreži putem mrežnih rutera, s obzirom da je trenutni nedostatak to što se aplikacija uspešno izvršava samo dok je konekcija sa internetom uspostavljena.

Po završetku realizacije AZC koncepta, očekuje se rešenje ovog zadatka u potpunosti. Nakon što se na centralnoj jedinici implementiraju slučajevi promene stanja zonskih kontrolera, biće moguća reakcija na događaje tog tipa i ažuriranje informacija od značaja na *widget*-u aplikacije.

Nakon što to sve bude realizovano, u budućem radu, aplikacija može da se modifikuje tako što bi se pored očitavanja stanja senzora, moglo uvesti kontrolisanje uređaja po zonama kuće, slanjem komandi njihovim kontrolerima.

## 7. Literatura

- [1] OBLO zvanični sajt: <http://obloliving.com/pocetna/>, pristupano 28.6.2018.
- [2] Ivan Lazarević, *Realizacija pouzdanog mehanizma ažuriranja programske podrške na namjenskim platformama zasnovanim na Linuks operativnom sistemu*, Univerzitet u Novom Sadu, Fakultet tehničkih nauka, 2016
- [3] MQTT protokol, dostupno na: <https://hivemq.com/>, pristupano 28.6.2018.
- [4] Milica Matić, *Realizacija OAuth 2.0 modula u okviru IoT sistema*, Univerzitet u Novom Sadu, Fakultet tehničkih nauka, 2017
- [5] OBLO centralni uređaj, dostupno na: <http://obloliving.com/oblo-living-kontroler/>, pristupano 28.6.2018.
- [6] OBLO *cloud*, dostupno na: <http://obloliving.com/tehnologija/>, pristupano 28.6.2018.
- [7] I. Lazarević, M. Sekulić, M. Savić, V. Mihić, *Modular home automation software with uniform cross component interaction based on services*, ICCE Berlin 2015 IEEE 5th International Conference
- [8] MQTT *Quality of Service*, dostupno na: <https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/>, pristupano 28.6.2018.
- [9] Milan Tucić, *Protokol za razmenu poruka u sistemima pametnih kuća*, Univerzitet u Novom Sadu, Fakultet tehničkih nauka, 2016
- [10] Android *Application Widget*, dostupno na:  
<https://developer.android.com/guide/topics/appwidgets/>, pristupano 28.6.2018.
- [11] MVP šablon, dostupno na: <https://www.journaldev.com/14886/android-mvp>, pristupano 28.6.2018.
- [12] RxJava, dostupno na: <http://reactivex.io/RxJava/2.x/javadoc/>, pristupano 28.6.2018.
- [13] Jurica Maltar, *Angular*, Diplomski rad, Osijek, 2017

- [14] JSON format, dostupno na: <https://www.json.org/>, pristupano 28.6.2018.
- [15] Android servis, dostupno na:  
<https://developer.android.com/guide/components/services>, pristupano 28.6.2018.