



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

РЕАЛИЗАЦИЈА ПЛАТФОРМЕ ЗА ПРИКУПЉАЊЕ ПОДАТАКА ИЗ ИоТ СИСТЕМА

Кандидат: Сандра Ивановић

Број индекса: РА 46/2013

Тема рада: Реализација платформе за прикупљање података из IoT система

Ментор рада: Проф. др Иштван Пап

Нови Сад, јул, 2017



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:	Сандра Ивановић		
Ментор, МН:	проф. др Иштван Пап		
Наслов рада, НР:	Реализација платформе за прикупљање података из ИоТ система		
Језик публикације, ЈП:	Српски / латиница		
Језик извода, ЈИ:	Српски		
Земља публиковања, ЗП:	Република Србија		
Уже географско подручје, УГП:	Војводина		
Година, ГО:	2017.		
Издавач, ИЗ:	Ауторски репрингт		
Место и адреса, МА:	Нови Сад; трг Доситеја Обрадовића 6		
Физички опис рада, ФО: (поглавља/страница/цитата/табела/слика/графика/прилога)	7/31/0/0/14/0/0		
Научна област, НО:	Електротехника и рачунарство		
Научна дисциплина, НД:	Рачунарска техника		
Предметна одредница/Кључне речи, ПО:	ИоТ, базе података, паметна кућа		
УДК			
Чува се, ЧУ:	У библиотеци Факултета техничких наука, Нови Сад		
Важна напомена, ВН:			
Извод, ИЗ:	У овом раду описана је имплементација платформе за прикупљање података у оквиру постојећег система за кућну аутоматизацију „Обло“.		
Датум прихватања теме, ДП:			
Датум одбране, ДО:			
Чланови комисије, КО:	Председник:	доц. др Немања Лукић	
	Члан:	доц. др Иван Каштелан	Потпис ментора
	Члан, ментор:	проф. др Иштван Пап	



KEY WORDS DOCUMENTATION

Accession number, ANO:		
Identification number, INO:		
Document type, DT:	Monographic publication	
Type of record, TR:	Textual printed material	
Contents code, CC:	Bachelor Thesis	
Author, AU:	Sandra Ivanović	
Mentor, MN:	Ištván Papp, PhD	
Title, TI:	Implementation of data collection platform in IoT system	
Language of text, LT:	Serbian	
Language of abstract, LA:	Serbian	
Country of publication, CP:	Republic of Serbia	
Locality of publication, LP:	Vojvodina	
Publication year, PY:	2017.	
Publisher, PB:	Author's reprint	
Publication place, PP:	Novi Sad, Dositeja Obradovica sq. 6	
Physical description, PD: <small>(chapters/pages/ref./tables/pictures/graphs/appendices)</small>	7/31/0/0/14/0/0	
Scientific field, SF:	Electrical Engineering	
Scientific discipline, SD:	Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, S/KW:	IoT, Big Data, smart home	
UC		
Holding data, HD:	The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, N:		
Abstract, AB:	This paper is a description of an implementation of data collection platform into an existing smart home system "Oblo".	
Accepted by the Scientific Board on, ASB:		
Defended on, DE:		
Defended Board, DB:	President:	Nemanja Lukić, PhD
	Member:	Ivan Kaštelan, PhD
	Member, Mentor:	Ištván Papp, PhD
		Menthor's sign

Zahvalnost

Zahvalujem se mentorima prof. dr Pap Ištvanu i dr Mariji Antić na saradnji, savetima i stručnoj podršci prilikom nastanka ovog rada. Takođe se zahvalujem članovima tima Aisha na uspešnoj saradnji tokom rada na projektu.

Posebnu zahvalnost bih izrazila prema mojoj porodici i prijateljima na bezrezervnoj podršci i veri u mene tokom studija.



УНИВЕРЗИТЕТ У НОВОМ САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА



SADRŽAJ

1.	Uvod	1
2.	Teorijske osnove.....	4
2.1	Sistem kućne automatizacije „Oblo“	4
2.1.1	Centralni uređaj	4
2.1.2	Servis u oblaku	5
2.1.3	Scene i akcije.....	6
2.1.4	MQTT protokol	6
2.2	Modul za glasovnu korisničku spregu	6
2.2.1	Amazon Aleksa	7
2.2.2	Programska podrška za Aleksa veština (eng. <i>Skill Service</i>).....	7
2.2.3	Aleksa korisnička sprega (eng. <i>Alexa Skill Interface</i>).....	8
2.3	Kaa platforma.....	9
2.4	Kassandra (eng. <i>Cassandra</i>).....	10
2.4.1	Kassandra jezik za upite (CQL)	10
3.	Koncept rešenja	11
3.1	Koncept dizajna sistema za prikupljanja podataka	11
3.1.1	Kaa aplikacija i okruženje za razvoj.....	11
3.1.2	Konfigurisanje i generisanje skupa alata za razvoj programske podrške	13
3.2	Koncept dizajna sistema za slanje podataka	16
3.2.1	Klijentska aplikacija za prikupljanje podataka	17
3.2.2	Kaa operativni servis	17
3.2.3	Kaa kontrolni servis.....	17
3.2.4	Kaa servis za komunikaciju.....	18

3.2.5 Baze podataka.....	18
4. Programsko rešenje.....	19
4.1 Programsko rešenje modula za glasovnu korisničku spregu	19
4.2 Programsko rešenje klijentske aplikacije za prikupljanje podataka	20
4.2.1 Node.js dodaci (eng. <i>Addons</i>).....	21
5. Rezultati.....	25
6. Zaključak	29
7. Literatura	31

SPISAK SLIKA

Slika 2.1 – Arhitektura „Oblo“ sistema	5
Slika 2.2 – Arhitektura modula za prepoznavanje glasa.....	8
Slika 3.1 – Pokretanje Sandbox-a u okviru virtuelne mašine i kreiranje Kaa aplikacije u web okruženju.....	12
Slika 3.2 – Konfiguraciona šema i prikaz u okviru Sandbox okruženja.....	14
Slika 3.3 – Šema podataka i prikaz u okviru Sandbox okruženja.....	15
Slika 3.4 – Dizajn izgleda Kasandra tabele	15
Slika 3.5 – Konfigurisanje log appendera.....	16
Slika 3.6 – Generisanje SDK-a	16
Slika 4.1 – Odsečak programskog koda koji sadrži logiku slanja podataka i prevezivanja C++ funkcija u JavaScript	22
Slika 4.2 – Izgled Binding.gyp datoteke.....	23
Slika 5.1 – Izgled prazne baze pri inicijalizaciji	25
Slika 5.2 – Simulacija glasovne komande za pribavljanje vrednosti sa senzora korisnika u Alexa web testnom okruženju.....	26
Slika 5.3 – Prikaz programskog koda modula za glasovnu korisničku spregu odakle se presreću tražene vrednosti	27
Slika 5.4 – Prikaz Kasandra tabele za čuvanje korisičkih komandi i zahteva	28

SKRAĆENICE

- IoT** – Pojam povezanih uređaja na Internet (eng. Internet of Things)
- AVS** – Amazon servis za raspoznavanje glasa (eng. Alexa Voice Service)
- AWS** – Amazon Internet Servis (eng. Amazon Web Service)
- SDK** – Skup alata za razvoj programske podrške (eng. Software Development Kit)
- API** – Interfejs za definisanje programske aplikativne sprege (eng. Application Programming Interface)
- CQL** – Kasandra jezik za upite (eng. Cassandra Query Language)
- RAM** – Radna memorija računara (eng. Random Access Memory)
- JSON** – Format Java Script objekata (eng. JavaScript Object Notation)

1. Uvod

Pojam velikih i kompleksnih setova podataka (eng. *Big Data*) je sve interesantniji u ovoj novoj tehnološkoj eri, gde onaj koji raspolaže adekvatnim podacima ima prevlast u industriji. Pod ovim pojmom se podrazumevaju podaci koji nisu podložni obradi od strane standardnih alata za obradu podataka, pre svega zbog svog obima, ali i zbog svoje nestruktuiranosti. Svuda oko nas se neprekidno generiše velika količina podataka. Svaki digitalni procesor, senzor ili mobilni telefon predstavljaju izvor velike količine podataka. Dakle, pravi izazov leži najpre u diferenciranju relevantnih informacija, a potom i u njihovom adekvatnom prikupljanju i analiziranju.

Koncept uređaja povezanih na internet (eng. *IoT*) trenutno doživljava procvat na polju kućne automatizacije. Poslednjih godina, kućna automatizacija postaje sve popularnija i rasprostranjenija i na tržištu je dostupno mnoštvo „pametnih“ uređaja od različitih proizvođača, kao i sistema koji omogućavaju povezivanje uređaja i njihovo kontrolisanje. Omogućavanje kontrole različitih uređaja sa jednog mesta je jedan od najznačajnijih aspekata sistema za kućnu automatizaciju. Korisnička sprega za kontrolu mora biti intuitivna, pozdana i laka za korišćenje. Većina ovih sistema je kontrolisana putem mobilne ili Internet aplikacije.

Aplikacije za kontrolu komuniciraju sa centralnim uređajem sistema za kućnu automatizaciju (eng. *gateway*). Ovaj uređaj je uveden kao rešenje problema neuniformnosti perifernih uređaja, u smislu komunikacijskih protokola kojima se služe. On omogućava objedinjavanje svih perifernih uređaja u jedinstvenu mrežu, čineći sistem centralizovanim i unificira komunikaciju sa pojedinačnim uređajima. Ova komunikacija se odvija u lokalnoj mreži, preko mrežnih ruteru direktno sa centralnim uređajem, ili van lokalne mreže posredstvom servisa u oblaku (eng. *cloud service*).

Prikupljanje i obrada podataka o načinu upotrebe sistema je od interesa na polju kućne automatizacije, jer predstavlja osnov za razvoj funkcionalnosti koje omogućavaju komfornej

korišćenje sistema, jednostavniju kontrolu i povećavaju ekonomičnost. Obradom informacija vezanih za svakog pojedinačnog korisnika, sistem uči o njegovim navikama, potrebama i rutinama i može se adekvatno prilagoditi korisničkim potrebama.

Platforma za prikupljanje podataka prestavlja skup međusobno zavisnih komponenti, čiji je krajnji cilj beleženja podataka o korisničkoj aktivnosti u okviru IoT sistema. Podaci se beleže u određenom formatu, čuvaju i kasnije obrađuju, a sve u cilju analiziranja ponašanja korisnika i dodavanja dodatnog stepena inteligencije u sistem. Prikladnom analizom ovako formatiranih podataka može se doći do uočavanja obrazaca u korisnikovim aktivnostima i stvaranja predikcija o korisničkim potrebama.

Primer uočene rutine u ponašanju bila bi, recimo, navika korisnika da svakog dana u poslepodnevnim časovima spušta roletnu, podešava osvetljenje i temperaturu na određeni nivo i pali televizor. Ako se uoči rutina u ponašanju, pronalazi se okidač ovih akcija. U pomenutom slučaju, okidač je određeno doba dana. Uočeni obrasci ponašanja mogu rezultirati akcijom od strane sistema na identifikovani okidač.

Takođe, može se omogućiti da na zahtev korisnika sistem reprodukuje određeni obrazac, bez obzira da li je on ustaljen ili ne (npr. izdavanje komande da se ponove aktivnosti sistema u toku nekog prethodnog perioda). U ovim slučajevima bitno je voditi računa o prirodi akcija koje su zabeležene i razlučiti koja je možda bila izuzetak u ponašanju, te stoga ne treba da se ponovi. Ovo je korisno za korisnike ne samo zbog mogućnosti da komfornije koriste sistem, već i u slučajevima kada napuštaju kuću na duže vreme, a iz različitih razloga imaju potrebu da se neke akcije u okviru sistema i dalje obavljaju.

Izazov koji implementacija ovakve platforme donosi jeste uočavanje pogodnog mesta za dodavanje komponente za prikupljanje podataka u sistem. Takođe, od interesa su brzina rada ove komponente, kao i zaštita podataka koji se čuvaju. Problem leži i u osobini podataka da su nestruktuirani, tako da posebnu pažnju treba posvetiti dizajnu i implementaciji formata u kome se podaci prikupljaju i beleže. Takođe, iz perspektive korisnika, uvođenje veštačke inteligencije u svakodnevni životni prostor je i dalje pomalo zastrašujuće. Stoga je bitno zadobiti poverenje korisnika ostavljanjem mogućnosti da se autonomne funkcionalnosti sistema isključe na zahtev korisnika u bilo kom trenutku. Sa korisničke strane takođe je bitno da su podaci o aktivnostima sigurni, tako da prilikom čuvanja, ali i prenosa, podaci moraju biti zaštićeni.

U ovom radu, realizovana je platforma za prikupljanje podataka o korišćenju modula za glasovnu korisničku spregu, implementiranog u okviru postojećeg sistema kućne automatizacije „Oblo“. U uvodnom delu je opisan motiv realizacije ovakve platforme za prikupljanje podataka u IoT sistemu za kućnu automatizaciju, a u daljem radu je detaljno

opisan mehanizam realizacije platforme i njene implementacije u okviru postojećeg sistema za kućnu automatizaciju „Oblo“.

Rešenje obuhvata implementaciju platforme za prikupljanje podataka u okviru „Oblo“ sistema, uz komunikaciju sa poslužiocima treće strane, kao što je Kaa IoT platforma, zadužena za povezivanje aplikacija i „pametnih“ uređaja sa Cassandra bazom u kojoj se čuvaju podaci. Prevaziđeni su problemi koji su se javili usled neuskladenosti tehnologija koje koriste postojeće komponente sistema i novouvedene komponente. Rešenje je realizovano kao proširenje postojećeg servisa za podršku glasovnih komandi, pod nazivom Aleksa u okviru „Oblo“ sistema. Krajnje rešenje je ostvareno u programskom jeziku Javascript i razvojnom okruženju Node.js, a funkcionalnost pojedinih komponenti je implementirana u programskom jeziku C++.

2. Teorijske osnove

S obzirom da je rešenje proširenje i dopuna postojećeg IoT sistema, u ovom poglavlju je ukratko objašnjena trenutna arhitektura sistema, kao i tehnologije koje su u upotrebi i protokoli koji se koriste za komunikaciju komponenti sistema. Takođe su opisane tehnologije korištene za realizaciju same platforme za prikupljanje podataka, i njeno prilagođavanje postojećem sistemu.

2.1 Sistem kućne automatizacije „Oblo“

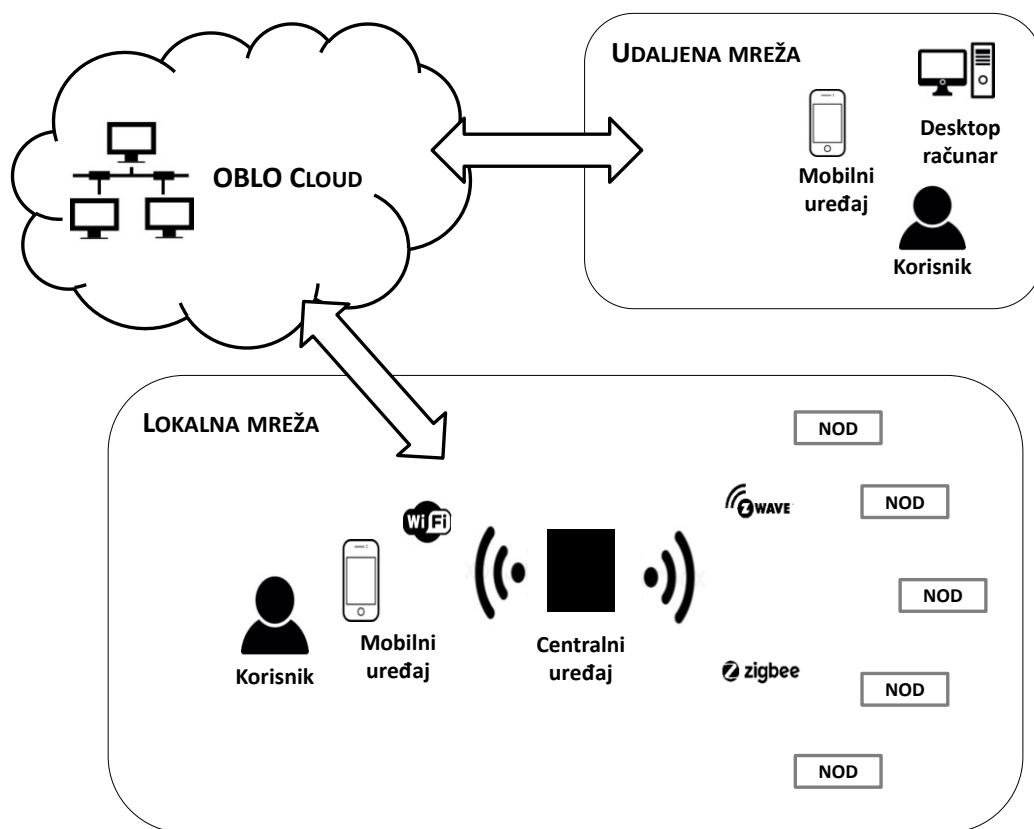
Trenutna arhitektura „Oblo“ sistema je prikazana na Slici 2.1, na kojoj su predstavljeni osnovni blokovi sistema. Kao što se može videti, sistem je organizovan tako da je komunikacija sa uređajima ostvarena najpre povezivanjem perifernih uređaja (nodova) na centralni uređaj, koji je u komunikaciji sa mrežnim ruterom. Mrežni ruter komunicira sa klijentom (mobilni klijent, klijent za glasovno izdavanje komandi), kao i sa servisom u oblaku. Servis u oblaku takođe ima mogućnost direktnе komunikacije sa klijentom.

U zavisnosti od toga da li se klijent nalazi u lokalnoj mreži u kojoj je mrežni ruter ili na udaljenoj lokaciji, njegova komunikacija sa centralnim uređajem se obavlja direktno ili preko servisa u oblaku.

2.1.1 Centralni uređaj

S obzirom da su periferni uređaji u sistemu od različitih proizvođača, koriste se drugačijim načinima komunikacije i neophodno ih je unifikovati [1]. Uređaji se prema funkcionalnosti mogu podeliti na senzorske i aktuatoriske. Funkcionalnost senzorskih uređaja je ograničena na detekciju određenih parametara u okolini, kao na primer temperature, pokreta, vlage kao i senzori kontakta vrata/prozora. Aktuatoriski uređaji se mogu kontrolisati, te kao rezultat mogu menjati stanje, tj. izvršavati neku akciju (npr. svetla, prekidači sa

potenciometrom, „pametne“ utičnice, itd). Različite funkcionalnosti ovih uređaja je neophodno apstrahovati i generalizovati pristup. Upravo kao rešenje ovog problema javlja se centralni uređaj koji omogućava da se funkcionalnostima različitih uređaja pristupa sa višeg, aplikativnog nivoa. Centralni uređaj podržava različite protokole kojima periferni uređaji komuniciraju. Takođe, uvodi i napredniju logiku u sistem, poput opcije da se uređaji grupišu po zonama po kući, različitim spratovima i sobama, kao i mogućnosti kreiranja scena i akcija u kojima uređaji učestvuju. Svaki centralni uređaj poseduje jedinstveni identifikacioni broj i za njega se vezuje korisnički nalog sa, takođe jedinstvenim, korisničkim brojem. Jedan korisnički nalog može biti povezan sa više centralnih uređaja sa različitim identifikacionim brojevima.



Slika 2.1 – Arhitektura „Oblo“ sistema

2.1.2 Servis u oblaku

Kao što je prethodno napomenuto, postoje dva moguća toka komunikacije klijentskih aplikacija sa uređajima: direktni, kada se klijent nalazi u istoj lokalnoj mreži kao i centralni uređaj, ili posredstvom servisa u oblaku, kada je klijent van domaća lokalne mreže. Servis u oblaku (eng. *cloud*) obezbeđuje konstantno dostupnu programsku podršku na Internet-u, koja je povezana sa centralnim uređajem sa jedne strane i klijentima sa druge. Servis u oblaku je

dostupan i kroz klijentsku web aplikaciju, odakle je takođe moguć pregled i kontrola svih centralnih uređaja na konkretnom nalogu, kao i uređaja povezanih sa njima. Ovaj vid korisničke sprege pogodan je i za održavanje celokupnog sistema od strane administratora. Svaki centralni uređaj sadrži identifikacioni broj, koji se povezuje sa korisničkim nalogom. Komunikacioni protokol koji je u upotrebi za povezivanje centralnog uređaja sa servisima u oblaku je MQTT protokol.

2.1.3 Scene i akcije

Pored kontrole osnovnih funkcionalnosti uređaja povezanih na centralni uređaj, takođe je moguće konfigurisati scene i akcije, koje omogućavaju kontrolu više uređaja istovremeno, i čija konfiguracija se čuva na centralnom uređaju. Scena obuhvata akciju i okidač, koji može biti stanje nekog od senzora ili aktuatora. Akcija je definisana skupom uređaja koji se kontrolišu, kao i njihovim željenim stanjima. Akcija se aktivira u okviru scene, ukoliko se dogodi definisani okidač. Scene u okviru modula za glasovnu korisničku spregu mogu biti pokretane i zadavanjem imena scene umesto okidača.

2.1.4 MQTT protokol

MQTT protokol je M2M protokol [2] koji je veoma zastupljen u IoT sistemima zbog preplata-i-objava mehanizma komunikacije koji podržava. Prema specifikaciji, razlikuju se tri logičke komponente – primalac, pošiljalac i posrednici. Posrednik je neophodan u razmeni poruka između različitih MQTT klijenata i naziva se broker. Pošiljalac šalje poruke ka posredniku, pri čemu definiše temu poruke. Primaoci su preplaćeni na određene teme. Poruke primaocu isporučuje posrednik, na osnovu tema na koje je primalac preplaćen. Ovaj protokol se zasniva na neprekidnoj TCP vezi, tako da je neopodno u pozadini u svakom trenutku zadržavati aktivan proces. Prvi korak pri inicijalizaciji komunikacije jeste povezivanje primaoca sa brokerom preko kog se kasnije vrši dalja razmena informacija.

2.2 Modul za glasovnu korisničku spregu

U okviru „Oblo“ sistema jedan od klijentskih modula predstavlja modul koji omogućava krajnjim korisnicima da sistemu izdaje komande glasom. Razumevanje koncepta ovog modula sistema je ključno s obzirom da platforma za prikupljanje podataka, koja je realizovana i opisana u ovom radu, radi upravo u okvirima ovog modula. Ovaj modul komunicira sa servisom u oblaku paralelno i nezavisno od ostalih klijenata (npr. mobilna aplikacija). Koncipiran je tako da koristi Amazon-ov servis za prepoznavanje glasa. Jednom prepoznat komandni obrazac biva mapiran na stvarne uređaje u okviru klijentske aplikacije, koji su prethodno pribavljeni sa korisnikovog naloga i željenog centralnog uređaja (u slučaju

da korisnik ima više centralnih uređaja). Izdavanje komandi glasom fizički je omogućeno uvođenjem novog uređaja u sistem. Ovaj uređaj u sebi ima ugrađen Amazon-ov servis za prepoznavanje glasa, a povezan je i sa Amazon korisničkim nalogom.

2.2.1 Amazon Aleksa

Amazon je američka kompanija koja je vlasnik servisa Aleksa (eng. *Alexa*), koji omogućava prepoznavanje glasa [3]. Trenutno dostupni jezici su engleski (američki i britanski izgovor) i nemački. Ovaj servis je još uvek u razvoju i konstantno se unapređuje, a korisnicima je dostupan putem Amazon-ovih uređaja kao što su Echo (eng. *Echo*) i Dot, koji u sebi imaju ugrađene mikrofone i zvučnike. Do skora, svi uređaji koje Amazon nudi imali su istu funkcionalnost, a razlikovali su se pre svega po veličini i broju mikrofona. Noviji uređaji koje je ova kompanija izbacila u poslednjih godinu dana sadrže i podršku za vizuelnu spregu sa korisnikom, te imaju ekran i kamere. Aleksa može reagovati na pitanja o trenutnoj temperaturi, dodavati stavke na listu za kupovinu, naručiti hranu ili taksi ili prosto pročitati nešto sa Internet-a. Najzanimljiviji aspekt Alekse je svakako mogućnost sprege sa takozvanim Aleksa skupom veština (eng. *Alexa Skills Kit*). Aleksa skup veština omogućava dizajn glasovne korisničke sprege prilagođene proizvoljnog sistemu ili proizvoljnoj svrsi. U okviru procesa konfiguracije Aleksa veštine, predefinišu se govorni obrasci koje korisnik može korisiti i način na koji se ti obrasci mapiraju na komande u sistemu.

Aleksa servis za raspoznavanje glasa (AVS) je skalabilni servis u oblaku koji omogućava uređajima u kojima je sadržan da prepoznaju glasovne komande. Ono što je neophodno da uređaj sadrži su mikrofon i zvučnik. Pored postojećih Amazon-ovih uređaja, inženjeri ovaj servis mogu inkorporirati i u okviru sopstvenog namenskog uređaja, koristeći raspoloživi API. Takođe je ostvarena sprega sa mobilnom i web aplikacijom, koja korisnicima dozvoljava da kontrolišu Amazonove uređaje povezane na svoj nalog, konfigurišu ih, kao i da prate raspoznavanje govora od strane Aleksa servisa.

2.2.2 Programska podrška za Aleksa veštine (eng. *Skill Service*)

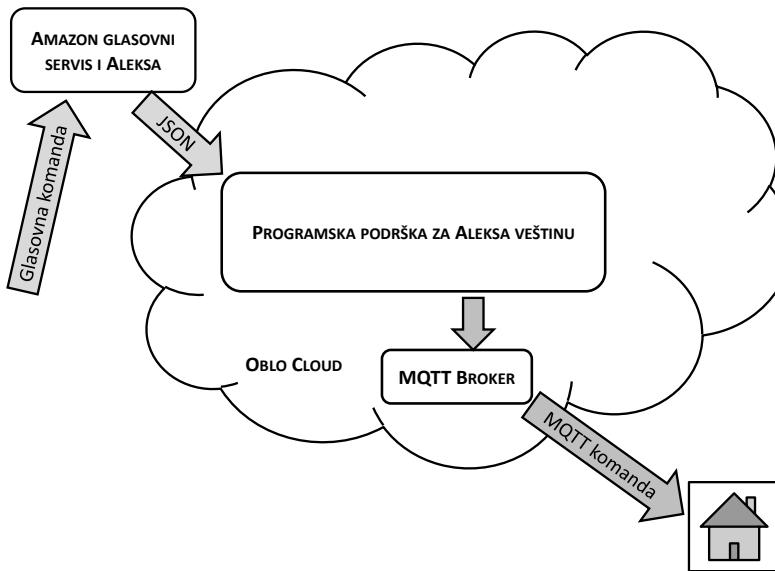
Da bi se komande glasom zaista mogle izvršiti u okviru postojećeg sistema za kućnu automatizaciju, neophodno je definisati logiku funkcije u okviru servisa u oblaku. Postoji mogućnost kreiranja sopstvenog servisa, ili upotrebe takozvane Lambda funkcije [4].

Lambda funkcija je zahvalna po pitanju održavanja, s obzirom na to da se Amazon brine o serverskoj podršci. Kod se pokreće i izvršava samo u trenutku kada korisnik izda komandu, tako da nema potrebe sa kontinualnim serverskim procesom.

U slučaju kada se programska podrška implementira u okviru sopstvenog rešenja u oblaku, potrebno je definisati u okviru Aleksa veštine serversku instancu na čiju će adresu

Aleksa servis za prepoznavanje glasa upućivati zahteve. Ovakav način funkcionisanja je sa korisničke strane istovetan sa Lambda funkcijom, ali u ovom slučaju implementirani servis može ostati stalno aktivan i pamtitи stanja uređaja, što je bolje sa stanovišta performansi.

Kao što je napomenuto, programska podrška za Aleksa veština sadrži logiku sistema za upravljanje glasom, dakle tok obrade od trenutka kada Aleksa servis za raspoznavanje glasa prepozna izrečenu komandu i mapira je na predefinisane obrascce i polja u okviru obrazaca. Ova polja prestavljanu ulazne parametre za implementiranu programsku podršku za Aleksa veština, i predstavljaju željene akcije, imena uređaja koji se kontrolisu, kao i vrednosti parametara koji se postavljaju određenim komandama. Komunikacija sa Aleksa servisom za prepoznavanje glasa je zatvorena za korisnika i može se presresti samo rezultat u krajnjem formatu, a komunikacija sa „Oblo“ servisom u oblaku se odvija preko MQTT protokola. Na slici 2.2 se može jasnije sagledati arhitektura modula za prepoznavanje glasa i sprega sa servisima u oblaku.



Slika 2.2 – Arhitektura modula za prepoznavanje glasa

2.2.3 Aleksa korisnička sprege (eng. *Alexa Skill Interface*)

Prilikom podešavanja Aleksa korisničke sprege prevashodno definišu načini na koje se korisnik obraća sistemu za prepoznavanje glasa, kao i podržani komandni obrasci. Definišu se očekivane semantičke celine komandi (polja), kao i očekivane vrednosti za ta polja. Aleksa koristi ove informacije da poboljša prepoznavanje komandi.

Takođe, pri podešavanju Aleksa korisničke sprege, definiše se i da li se funkcionalnost izvršava u okviru Lambda funkcije ili implementiranog servisa u oblaku. Priroda funkcije ne utiče na dizajn same korisničke sprege.

2.3 Kaa platforma

U ovoj realizaciji korišćena je Kaa IoT [5] platforma koja omogućava povezivanje sistema za kućnu automatizaciju sa bazom podataka, u cilju direktnog skladištenje informacija sa uređaja koji se nalaze u sistemu. Ova platforma je besplatna, potkrepljena opširnom dokumentacijom i primerima upotrebe, kao i nezavisna od osobina fizičkih komponenti (eng. *hardware*) na kom je potrebno da funkcioniše.

Ova platforma pokriva više različitih aspekata upotrebe, ali je pre svega namenjena za izgradnju kompletног sa-kraja-na-kraj rešenja. Povezuje aplikacije i „pametne“ uređaje i sisteme. Ova platforma je izabrana zbog svoje brzine, kao i mogućnosti sprezanja sa različitim alatima za obradu podataka i bazama podataka, poput Kasandre (eng. *Cassandra*), Mongo baze, Cepelin alata za analizu podataka (eng. *Zeppelin*) i mnogih drugih. Kao što je već spomenuto, Kaa platforma je nezavisna od arhitekture fizičkih komponenti IoT sistema.

Kaa omogućava praćenje i regulisanje toka podataka između krajinjih uređaja i infrastrukture za prikupljanje podataka (baze). Ova platforma obezbeđuje serversku instancu koja komunicira sa bazom podataka i krajinjim uređajima, kao i SDK koji se može upotrebiti za izradu programske podrške koja obezbeđuje logovanje podataka u krajinjim uređajima, odnosno čvorovima sistema (eng. *nodes*). Zbog prirode arhitekture „Oblo“ sistema, i činjenice da čvorovi u sistemu mogu biti proizvedeni od strane različitih proizvođača (te u opštem slučaju nije moguće proširiti njihovu funkcionalnost dodavanjem programske podške za logovanje), prikupljanje podataka je moguće vršiti ili na centralnom uređaju, ili na „Oblo“ servisu u oblaku.

Kaa SDK obezbeđuje skup API poziva, koji se mogu koristiti u klijentskoj aplikaciji kao sprega sa Kaa serverskom instancom, a samim tim i bazom podataka namenjenom za kasnije skladištenje informacija. Alati za razvoj se generišu automatski na osnovu različitih konfiguracionih parametara i mogu biti u više različitih programskih jezika: C, C++, Java, Android ili Objektni C. Zahvaljujući ovakvim osobinama skupova alata za razvoj, moguća je njihova integracija u gotovo svaki tip krajinjih uređaja ili sistema.

Kaa platforma obezbeđuje svu neophodnu pozadinsku podrušku. Razrešava komunikaciju između komponenti sistema i moguće greške u konekciji, obezbeđuje sigurnost i konzistentnost podataka, i prevazilazi međusobnu nekompatibilnost uređaja u sistemu.

2.4 Kasandra (eng. *Cassandra*)

Kasandra baza podataka, razvijena od strane organizacije Apače (eng. *Apache*) je izabrana u ovom rešenju realizacije platforme za skupljanje podataka, pre svega zbog svoje skalabilnosti i performansi [6]. Kompatibilna je za rad sa servisima u oblaku ili u okviru lokalnog sistema. Otporna je na greške i podržava repliciranje preko više paralelnih centara podataka. Ono što je čini otpornom na greške jeste osobina decentralizovanosti. Naime, svi čvorovi sistema su nezavisni jedni od drugih, a čvor u kom nastane eventualna greška biva odmah zamenjen, bez čekanja na odziv vidljiv korisniku ili aplikaciji koja koristi bazu. Bitne osobine ove baze podataka su i elastičnost i pouzdanost. Čak i u slučajevima da čitav centar podataka zakaže, podaci ostaju očuvani. Kasandru koriste svetski renomirana udruženja, organizacije i firme, poput CERN-a, Instagram-a, GitHub-a, Netflix-a, Ebay-a i mnoge druge [7].

2.4.1 Kasandra jezik za upite (CQL)

Jezik za upite koji Kasandra koristi nudi sličan je popularnom SQL-u. Podaci se čuvaju u tabele, koje se sastoje od redova i kolona.

Ovaj jezik koristi identifikatore za identifikovanje tabela, kolona ili drugih objekata. Takođe postoje rezervisane ključne reči komandi, poput ključne reči za odabiranje tabele, reda ili kolone („SELECT“). Identifikatori i ključne reči su neosetljivi na velika ili mala slova, ali se po ustaljenoj konvenciji za identifikatore koriste mala slova, a za ključne reči velika. Bilo koje reči stavljene pod navodnike, uključujući tu i ključne reči, mogu se koristiti za imenovanje tabele, kolone ili reda, što može dovesti do dvosmislenosti i grešaka, te bi ovu praksu trebalo izbegavati. CQL je tipiziran jezik i podržava opsežan skup tipova podataka, a korisnici takođe mogu sami da kreiraju sopstvene tipove. Manipulacija podacima se svodi na ubacivanje novih, brisanje ili izabiranje podataka iz tabele, brisanje ili izabiranje podataka iz čitave tabele, označavanje, uslovno pristupanje, kao i osvežavanje podataka.

3. Koncept rešenja

U prethodnom poglavlju pojašnjen je princip funkcionisanja „Oblo sistema“, modula za glasovne komande koji je u okviru njega implementiran, kao i Kaa platforme. Cilj ovog rada je ispitivanje mogućnosti da se Kaa platforma iskoristi za prikupljanje podataka u postojećem „Oblo“ sistemu, pre svega na strani servisa u oblaku. U svrhu potvrde ovakve mogućnosti, implementirana je funkcionalnost beleženja komandi izdatih glasom. Pristup korišćen u ovom slučaju može se dalje iskoristiti za potpunu integraciju „Oblo“ servisa u oblaku sa Kaa platformom. U nastavku rada, fokus je na detaljnoj analizi postupka i koncepta realizacije platforme za prikupljanje korisničkih komandi i stanja sistema, ostvarenih preko modula za glasovne komande i njihovo slanje preko Kaa IoT platforme do Kasandra baze podataka.

3.1 Koncept dizajna sistema za prikupljanja podataka

3.1.1 Kaa aplikacija i okruženje za razvoj

Prvi korak u kreiranju Kaa aplikacije jeste pokretanje instance Kaa servera. Najbrži način da se Kaa server pokrene je upotreba virtuelnog okruženja koje u sebi ima integrisane sve neophodne Kaa servise. Ovo virtuelno okruženje naziva se Kaa Sandbox. Osim što objedinjuje sve potrebne servise i alate, Kaa Sandbox takođe sadrži korisničku spregu pogodnu za kreiranje Kaa aplikacije, konfigurisanje alata za razvoj programske podrške, a sadrži i mnoštvo primera korišćenja Kaa platforme.

Ovaj alat može biti pokrenut na dva načina:

- Alat je dostupan za skidanje i pokretanje u okviru virtuelne mašine (Oracle VirtualBox VM). Prethodno je potrebno instalirati virtuelnu mašinu, i potom

uvesti programsku sliku Kaa Sandbox aplikacije, dodeliti potrebne resurse (minimum 2 procesora i 4096 MB RAM memorije) i pokrenuti je.

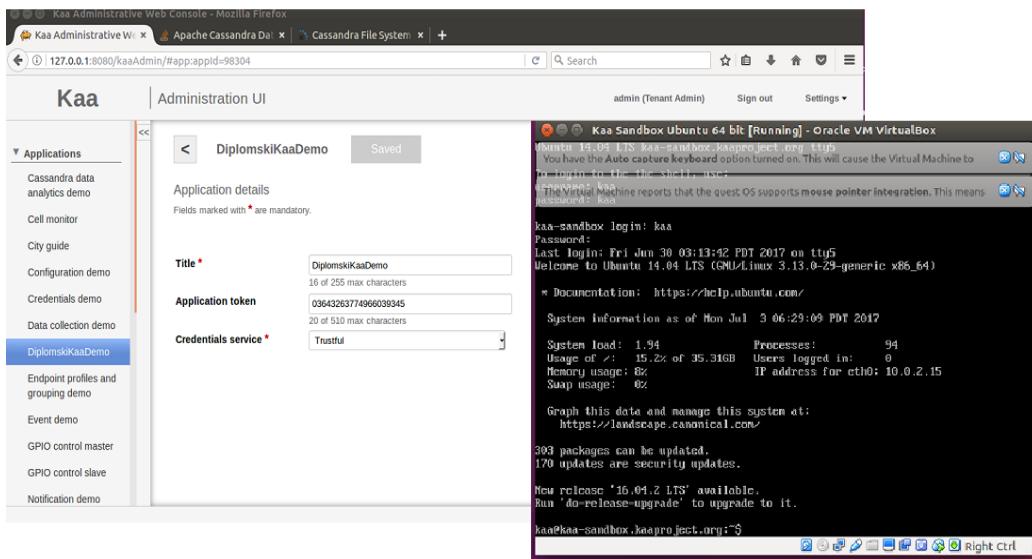
- Instanca Kaa Sandbox-a može biti pokrenuta na Amazonom Web servisu.

U ovom rešenju korišćena je prva opcija, tj. Kaa Sandbox instanca pokretana je u okviru virtuelne mašine, s obzirom da se pokretanje u okviru Amazon Web servisa naplaćuje.

Ovako privatna instanca Kaa Sandboxa dostupna je na Internetu preko adrese u obliku: **http://<Sandbox IP adresa>:9080/sandbox**.

Sledeći korak u implementaciji sistema za prikupljanje podataka je kreiranje aplikacije na Kaa platformi. Pod aplikacijom na Kaa platformi podrazumeva se skup definicija za šeme logova, korišćene baze podataka, razvojne alate i spregu sa klijentskom aplikacijom.

Aplikacija se kreira prijavljivanjem na sistem sa kredencijalima administratora instance. Na Slici 3.1 može se videti snimak ekrana prilikom pokretanja Kaa Sandbox instance sa administratorskim kredencijalima.



Slika 3.1 – Pokretanje Sandbox-a u okviru virtuelne mašine i kreiranje Kaa aplikacije u web okruženju

Postoje dva smera komunikacije između Kaa platforme i uređaja u sistemu:

- **Pribavljanje podataka** – omogućava slanje podataka sa krajnje tačke korisničkog sistema (u ovom slučaju iz servisa za glasovne komande) na Kaa server.
- **Konfiguracija** – omogućava slanje konfiguracionih parametara sa Kaa servera do krajnjih tačaka sistema. Ove informacije se distribuiraju klijentima (tj. odgovarajućoj klijentskoj aplikaciji). Jedan od najčešće upotrebljavanih

konfiguracionih parametara je vremenski period u kome se podaci šalju sa krajnje tačke korisničkog sistema na Kaa server.

Sledeći korak u konfigurisanju Kaa aplikacije je definisanje šema za format podataka koji će se slati od strane korisničke aplikacije, kao i organizacija istih. Takođe, potrebno je konfigurisati period odabiranja podataka koji se šalju od čvorova (tj. odgovarajuće korisničke aplikacije) do Kaa serverske instance.

3.1.2 Konfigurisanje i generisanje skupa alata za razvoj programske podrške

Sledeći korak se sprovodi takođe u okviru Sandbox virtuelnog okruženja, ali ovog puta prijava je sa kredencijalima projektanta sistema. Šeme logova prethodno kreirane u JSON formatu dobijaju novi oblik, i biva im dodeljen broj verzije koji se kasnije koristi pri generisanju alata za razvoj programske podrške (SDK). Na slikama 3.2 i 3.3 može se ispratiti proces kreiranja ovih šema.

Da bi se Kaa platforma mogla iskoristiti kao sprega prema bazi za prikupljanje podataka, neophodno je da konfigurisati izgled tabele u bazi podataka, koja je proširiva i ima mogućnost prilagođavanja potrebama korisnika (eng. *Log Appender*). Pri konfiguraciji ove sprege, bira se tip tabele koja se inicijalizuje (Slika 3.4). U ovom primeru kao izbor uzima se sprega sa Kasandrom (Slika 3.5).

U toku ovog procesa, konfigurišu se kolone i redovi tabele, tipovi podataka koji se očekuju u poljima tabele, nazivi zaglavlja, i očekivane veličine podataka.

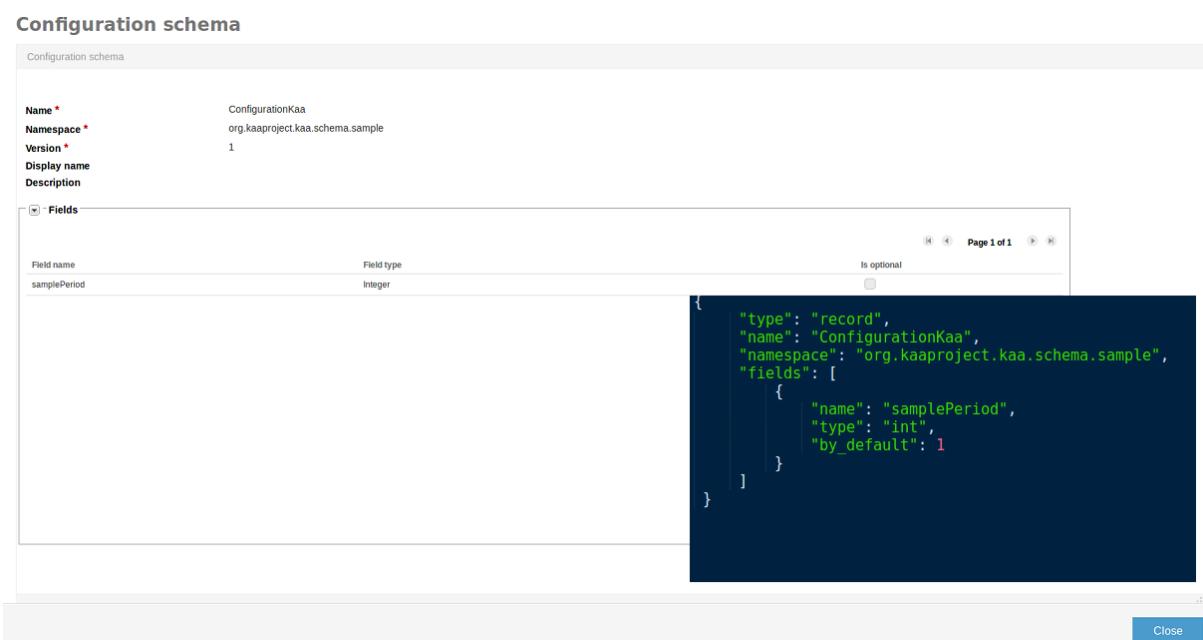
Tabela za potrebe implementiranog rešenja je kreirana tako da sadrži polja za identifikaciju klijenta, kao i vreme prihvatanja komande (vreme izdavanja komande), a izrečena komanda od strane korisnika parsirana je tako da se odvoji deo koji sadrži:

- Zahtev korisnika (*request* kolona)
- Uređaj nad kojim žeilo da akcija bude izvršena (u slučaju aktuatorских uređaja) ili sa kog želimo očitavanje stanja (u slučaju senzorskih uređaja)
- Tip kontrole koji je kao takav okarakterisan u okviru Aleksa skupa veština gde su definisani obrasci komunikacije i prepoznavanja korisnika (*Intent*). Primer sadržaja ovog polja je kontrola svetala (eng. *Light Intent*), kontrola senzora (eng. *Sensor Intent*), kontrola nivoa osvetljenja (eng. *Dim Intent*) i drugi.
- Osobinu uređaja koju u zadatoj komandi ciljamo da kontrolišemo, npr. temperatura, vlaga, osvetljenost, boja svetla i drugi (eng. *Property*)
- Vrednost na koju postavljamo traženu osobinu, tj. uređaj. Prepoznaje se na šta se odnosi na osnovu tipa kontrole i osobine uređaja (eng. *Value*). Primer

vrednosti polja bio bi visina temperature očitane sa senzora, količina osvetljenja u procentima ili boja svetla lampe.

Kao što je već napomenuto, tabela je proširiva i nezavisna od prethodno opisane konfiguracije šema.

Nakon podešenih svih ovih komponenti, vreme je da se definiše tip profila razvojnog alata (eng. *SDK profile*) i da se izgeneriše SDK. On se generiše na osnovu šema i tabele za prikupljanje podataka, u izabranom programskom jeziku. Među ponuđenim tehnologijama su Java, C, C++, Android i Objektivni C. Izabran je programski jezik C++. Na slici 3.6 se može pratiti tok generisanja razvojnog alata. Ovim se zaključuje koncept kreiranja sistema za prikupljanje podataka. Ovako izgenerisan razvojni alat koristi klijentska komponeta opisana u narednom poglavljtu.



Slika 3.2 – Konfiguraciona šema i prikaz u okviru Sandbox okruženja

Schema

The screenshot shows the schema configuration interface. At the top, there are fields for Name, Namespace, Version, Display name, and Description. Below this, a 'Fields' section lists six fields: property, request, device, intent, UserId, and value. Each field has a 'Field type' (String) and an 'Is optional' checkbox. To the right of the fields, a JSON representation of the schema is displayed:

```

{
  "type": "record",
  "name": "DataCollectionKaa",
  "namespace": "org.kaaproject.kaa.schema.sample",
  "fields": [
    {
      "name": "temperature",
      "type": "int"
    },
    {
      "name": "humidity",
      "type": "int"
    },
    {
      "name": "request",
      "type": "string",
      "is_optional": true
    },
    {
      "name": "device",
      "type": "string"
    },
    {
      "name": "percentage",
      "type": "int"
    }
  ]
}
  
```

Slika 3.3 – Šema podataka i prikaz u okviru Sandbox okruženja

CassandraConfig

The screenshot shows the CassandraConfig interface. It starts with a 'CassandraConfig' header and a 'Create' button. Below that, 'Authentication credentials' and 'Keyspace name' (set to 'kaa') are specified. A 'Table name pattern' is set to 'diplomski_logovi'. The main area is titled 'Column Mapping' and contains a table with the following data:

Type	Value	Name	Column Type	Is part of partition key?	Is part of clustering key?	Delete
EVENT_FIELD	property	property	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	X
EVENT_FIELD	request	request	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	X
EVENT_FIELD	UserId	UserId	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	X
EVENT_FIELD	device	device	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	X
EVENT_FIELD	intent	intent	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	X
EVENT_FIELD	value	value	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	X
TS	yyyy-MM-dd'T'HH:mm:ss.SS	date	TEXT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	X

Slika 3.4 – Dizajn izgleda Kasandra tabele

Kaa Administration UI

devuser (Tenant Developer)

Log appender

Saved

Log appender details
Fields marked with * are mandatory.

Name * Diplomski Log Appender
22 of 255 max characters

Min schema version * 1

Max version * Infinite

Confirm delivery

Log metadata Timestamp Application token

Description

Slika 3.5 – Konfigurisanje log appendera

SDK profiles

+ Add SDK profile

Name	Created by	Date created	Configuration	Profile	Notification	Log	SDK token	Event class families	Generate SDK	Delete
DiplomskiKaaSDK	devuser	06/29/2017	v2	v0	v1	v2	ckNoOgp1J7ZKAOGBnIq...	0		
DiplomskiDateSDK	devuser	06/30/2017	v2	v0	v1	v3	c5hfE46cW5kWErzn7Ku...	0		

Generate SDK

Target platform:

SDK profile details

SDK token: c5hfE46cW5kWErzn7KuVSvFugA
SDK name: DiplomskiDateSDK
Created by: devuser
Date created: 06/30/2017
Configuration schema: ConfKaaSchema (v.2)
Client-side EP profile schema: Generated (v.0)
Notification schema: Generated (v.1)
Log schema: KaaDataSchema (v.3)
Event family mappings

Slika 3.6 – Generisanje SDK-a

3.2 Koncept dizajna sistema za slanje podataka

Cilj ovog rada je ispitivanje mogućnosti da se Kaa platforma integriše sa „Oblo“ sistemom u oblaku, tj. da se klijentska aplikacija za prikupljanje podataka, koja je u sprezi sa Kaa serverom, realizuje upravo u okviru „Oblo“ oblak platforme. Kao prvi korak u potpunoj integraciji odabрано je da se klijentska aplikacija za prikupljanje podataka realizuje u okviru modula za upravljanje glasom. U okviru modula za upravljanje glasom, poznati su relevantni podaci o korisniku, i direktno se mogu pratiti aktivnosti korisnika. Pri tome, komande koje

pristižu od Aleksa servisa za prepoznavanje govora sadrže struktuirane podatke, koji su pogodni za logovanje.

3.2.1 Klijentska aplikacija za prikupljanje podataka

Klijentska aplikacija zadužena je za slanje podataka sa krajnjeg čvora u sistemu na Kaa server. U tu svrhu, klijentska aplikacija implementira API pozive obezbedene od strane izgenerisanog SDK-a. U okviru generisanog SDK-a, strukture podataka su prilagođene formatu podataka definisanog u šemi logova na Kaa serveru.

Klijentska aplikacija konstantno osluškuje informacije o konfiguracionim parametrima koje dolaze od Kaa servera, poput informacija o učestanosti slanja logova, kao i informacija o načinu slanja podataka. Podaci se od klijentske aplikacije ka Kaa serveru šalju ili kada se skupi određena količina podataka za slanje, ili kada prođe definisani vremenski interval. U slučaju implementiranog sistema, okidač je količina podataka. Podaci se šalju nakon svake zabeležene komande korisnika. Podaci se u tom trenutku isporučuju operativnom servisu u okviru Kaa serverske instance (eng. *Operations Service*) i nakon obrade i prilagođavanja formatu Kasandra baze podataka šalje se u bazu na čuvanje.

3.2.2 Kaa operativni servis

Operativni servis je jedan od tri servisa Kaa servera. Preostala dva su kontrolni servis i servis za konekciju. Primarna uloga operativnog servisa je komunikacija sa više različitih krajnjih korisnika (klijentskih aplikacija) konkurentno. U slučaju preopterećenja servisa, automatski se vrši preusmeravanje na sledeći operativni servis koji je slobodan. Ovo preusmeravanje Kaa server obavlja u trenutku izvršavanja neopaženo.

Takođe je moguće za svakog pojedinačnog klijenta konfigurisati po jednu instancu operativnog servisa koji će ga opsluživati. U tom slučaju, instance servisa rade paralelno. U implementaciji opisanoj u ovom radu koristi se jedan operativni servis, s obzirom da je implementiran samo jedan klijent, tj. klijentska aplikacija.

3.2.3 Kaa kontrolni servis

Kontrolni servis procesira API pozive i šalje obaveštenja operativnom servisu. On raspolaže listom svih trenutno dostupnih operativnih servisa tako što neprekidno prima informacije od komponente treće strane koja vodi računa o koordinaciji Kaa čvorova („Zookeeper“). Takođe, kontrolni servis je zalužan za obezbeđivanje web korisničke sprege za administratorske korisnike, preko kog mogu da rukuju korisničkim nalozima, aplikacijama i podacima. Kaa instanca za potrebe performanse visoke dostupnosti mora imati barem dva Kaa čvora sa aktivnim kontrolnim servisom. Jedan od njih je uvek aktivan, a drugi u pripravnom

režimu. U slučaju otkaza aktivnog kontrolnog servisa, pomenuta komponenta treće strane obaveštava pripravni kontrolni servis o tome i ovaj postaje aktiviran.

3.2.4 Kaa servis za komunikaciju

Kaa servis za komunikaciju nadgleda stanje konekcionih parametara operativnih servisa i šalje informacije o njima ka krajnjim klijentskim aplikacijama. U zavisnosti od konfiguracije, konekcioni parametri su različiti za potrebe svakog korisnika ponaosob. Ovi parametri mogu sadržati IP adresu, broj porta i korisničke kredencijale. Korisnička aplikacija šalje upite ovom servisu, koristeći generisani razvojni alat čijim funkcijama se služi, kako bi zadobila ove parametre vezane za operativne servise na koje kasnije šalje podatke. Ova komunikacija je takođe koordinisana od komponente treće strane („ZooKeeper“).

3.2.5 Baze podataka

Kaa podržava korišćenje i SQL i ne-SQL baza podataka. U ovom rešenju korišćena je ne-SQL baza podataka. Ovakve baze su koalocirane sa Kaa čvorovima u okviru iste fizičke ili virtuelne mašine i pokreću se sa opcijama za visoku dostupnost, kako bi to doprinelo visokoj dostupnosti čitavog sistema. Ovo podrazumeva brži odziv i sigurniju tok podataka. Od ne-SQL baza podataka Kaa podržava Kasandra bazu podataka i Mongo bazu [8].

Kao rezultat generisanja jedinstvenog skupa alata za razvoj programske podrške sa serverske strane Kaa, dobijamo skup funkcija koje koristimo u okviru klijentske aplikacije napisane u istom programskom jeziku kao i generisani skup alata (C++). Korišćenje ovih funkcija zavisi od namere projektanta sistema, tj. korisnika o tome kako se podaci šalu na Kaa server i kasnije čuvaju u bazi.

Upotreba ovih funkcija je u jednu ruku ograničena tipovima podataka, jer se moraju koristiti izgenerisane strukture za tipove podataka, tako da realne postojeće tipove je potrebno adekvatno formatirati.

Problem koji se takođe javlja u konkretnoj implementaciji jeste nekompatibilnost programskih jezika klijentske aplikacije sa modulom u okviru kojeg biva implementirana. Način prevazilaženja ovog problema opisan je u narednom poglavlju, gde je dat detaljan koncept programskog rešenja korišćenog prilikom implementiranja platforme. Ovaj opis je propraćen mnoštvom adekvatnih snimaka ekrana pri pokretanju i nastajanju programske podrške, kao i delovima programskog koda.

4. Programsко rešenje

Implementirana klijentska aplikacija vrši prikupljanje podataka o korišćenju sistema iz modula za glasovnu korisničku spregu sa „Oblo“ sistemom. Polazni problem ove implementacije je činjenica da je klijent za glasovnu korisničku spregu zasnovan na Amazonovom Aleksa servisu, pisan u programskom jeziku JavaScript. Takođe, celokupno rešenje za kontrolu „Oblo“ sistema pametne kuće u oblaku je realizovano na Node.js platformi. Sa druge strane a Kaa klijentska aplikacija je realizovana u C++ programskom jeziku. Stoga je bilo neophodno uvezati C++ i Node.js kod.

4.1 Programsko rešenje modula za glasovnu korisničku spregu

Kao što je objašnjeno u poglavljiju 2.2, modul za glasovnu korisničku spregu realizovan u okviru „Oblo“ sistema komunicira sa jedne strane sa Amazon servisom za prepoznavanje glasa i servisom koji sadrži logiku mapiranja prepoznate komande na dizajnirane obrasce, a sa druge strane sa „Oblo“ servisom u oblaku. Komunikacija sa „Oblo“ servisom u oblaku omogućava pribavljanje konfiguracije sistema sa korisničkog naloga, kao i slanje komandi na izvršenje.

Realizacija programske podrške za glasovnu kontrolu je ograničena dizajnom govornih obrazaca, tj. poljima koja su definisana kao očekivana u okviru jezičkih obrazaca. Pod ograničenjem se misli na zavisnost programskog rešenja od ulaznih parametara i njihovog formata.

Najpre se parsira ulazni JSON objekat koji u sebi sadrži pomenuta polja. Vrednosti ovih polja se preuzimaju i mapiraju na stvarne vrednosti odgovarajućih polja pribavljenih sa korisnikovog centralnog uređaja, posredstvom servisa u oblaku. Komunikacija sa konkretnim klijentom, tj. njegovim nalogom omogućeno je povezivanjem Amazon i „Oblo“ korisničkog

naloga (eng. *Account Linking*). Na osnovu autorizacionog koda, koji je jedinstveni identifikator svakog korisnika, moguće je pribaviti identitet korisnika koji izdaje komandu. Slanjem ovog identifikatora preko HTTP POST zahteva [9] dobijaju se teme na koje se klijentska aplikacija pretplati (MQTT protokol) radi pribavljanja podataka o korisniku, kasnije za slanje komandi.

Ovaj modul je izabran kao pogodan za dodavanje klijentske aplikacije za prikupljanje podatka zbog činjenice da na njegov ulaz komande već pristižu u formatu pogodnom za prosleđivanje na Kaa server i kasnije u bazu. Naime nakon parsiranja ulaznog objekta dobijena polja odgovaraju poljima u Kasandra bazi tako da nije potrebno praviti novu logiku pribavljanja vrednosti polja. Takođe ostvareno je i povezivanje sa korisničkim nalogom i pribavlja se jedinstvena identifikacija korisnika u okviru ovog modula, koja je potrebna i u krajnjoj tabeli, pa nema rizika prenosa poverljive identifikacije.

4.2 Programsко rešenje klijentske aplikacije za prikupljanje podataka

Prikupljanje podataka vrši se iz klijentske aplikacije koja je implementirana na Node.js platformi. S obzirom na to da Kaa platforma ne obezbeđuje Node.js SDK, bilo je neophodno generisati isti na osnovu raspoloživog C++ SDK-a. Dodatna pogodnost implementacije funkcionalnosti za logovanje u programskom jeziku C++ je što se ona može upotrebiti i za izradu klijentske aplikacije koja se pokreće sa centralnog uređaja.

Podešavanje konfiguracionih parametara je neophodno svaki put izvršiti, jer se konfiguracija može menjati sa serverske strane, a neusklađenost između klijentske aplikacije i serverske strane može dovesti do gubljenja podataka i grešaka. Za potrebe ovog rada, konfiguracioni parametri nisu se menjali tokom testiranja – komande su slate sekvencijalno, svaki put kad su pribeležene.

Pri inicijalizaciji klijentske aplikacije, najpre se proziva funkcija zadužena za prijem konfiguracionih parametara. Konfiguracione parametre čine period i učestanost odabiranja.

Slanje podataka ka Kaa serveru vrši se uvek kada se sakupi određena količina podataka za slanje. Prag za slanje predstavlja jedan od konfiguracionih parametara, a za potrebe testiranja podešen je na vrednost 1. Stoga se funkcija za slanje podataka proziva svaki put kada se generišu podaci za slanje, tj. podaci se odmah po pribavljanju šalju na Kaa server. Ovakav način beleženja komandi je realizovan da bi se omogućilo praćenje vremenske oznake komandi, koja kasnije može biti značajna za analizu ponašanja. Takođe, za potrebe testiranja pri izradi ovog rada bilo je bitno pratiti rad sistema korak po korak, pa je postavljena ovakva konfiguracija kako bi se mogla proveravati ispravnost rada nakon svake komande.

Struktura podataka koji se šalju zavisi od podešavanja postavljenih prilikom definisanja šeme podataka. Što je jednostavnija šema, struktura će biti jednostavnija. U konkretnom slučaju, sva polja tabele objedinjena su u okviru jednog objekta. Polja su string tipa. Vrednosti se dodeljuju pojedinačnim poljima, a zatim se nad celim objektom poziva funkcija za slanje podataka.

Kao što je već spomenuto, problem nastaje usled činjenice da je klijentska aplikacija za glasovnu korisničku spregu pisana u JavaScript programskom jeziku i nemogućnosti da se SDK izgeneriše u tom obliku. Sledeća najbolja opcija je C++ kao jezik Kaa klijentske aplikacije i SDK-a zbog mogućnosti da se ova funkcionalnost pisana u C++ zadrži i „upakuje“ u obliku JavaScript programske podrške sa funkcijom koja se poziva iz modula za glasovnu korisničku spregu. Tehnički detalji implementacije su detaljno opisani u nastavku teksta i kroz snimke ekrana.

4.2.1 Node.js dodaci (eng. *Addons*)

Node.js dodaci su dinamički povezani objekti napisani u programskom jeziku C++. Dodaci se preuzimaju i uvrštavaju u postojeće programsко rešenje pozivanjem funkcije *require()*, a potom se koriste na isti način kao i standardni Node.js moduli. Na ovaj način, ostvaruje se sprega između Javascript i C++ programskega koda. Mogućnost kreiranja Node.js dodatka omogućava da već postojeću funkcionalnost implementiranu u C++ programskom jeziku koristimo u okviru JavaScript programskega jezika, pokrenutog na Node.js platformi.

Okruženje u kojem se rukuje funkcijama ove biblioteke i u kojem se realizuje sprega dve funkcionalnosti naziva se V8. V8 je C++ biblioteka koja na Node.js platformi pruža mogućnost korišćenja JavaScript-a [10]. Konkretno, V8 obezbeđuje mehanizme za kreiranje JavaScript objekata, pozivanje funkcija i slično [11]. Na slici 4.1 može se videti deo funkcionalnosti implementiran u programskom jeziku C++, koji se prevodi u JavaScript.

```

Isolate* isolate = args.GetIsolate();

v8::String::Utf8Value param5(args[0]->ToString());
string request = string(*param5);
v8::String::Utf8Value param1,args[1]->ToString();
string device = string(*param1);
v8::String::Utf8Value param3,args[2]->ToString();
string intent = string(*param3);
v8::String::Utf8Value param2,args[3]->ToString();
string userId = string(*param2);
v8::String::Utf8Value param4,args[4]->ToString();
string value = string(*param4);

auto kaaClient = Kaa::newClient();
kaaClient->setLogUploadStrategy(std::make_shared<Kaa::RecordCountLogUploadStrategy>(1, kaaClient->getKaaClientContext()));

// Start an endpoint
kaaClient->start();

// Create a log entity (according to the org.kaaproject.sample.logData sample schema above)
KaauUserLogRecord logRecord;
logRecord.request = request;
logRecord.device = device;
logRecord.intent = intent;
logRecord.UserId = UserId;
logRecord.value = value;

// Push the record to the collector
auto recordDeliveryCallback = kaaClient->addLogRecord(logRecord);

try {
    auto recordInfo = recordDeliveryCallback.get();
    auto bucketInfo = recordInfo.getBucketInfo();
    std::cout << "Received log record delivery info. Bucket Id [" << bucketInfo.getBucketId() << "] . "
    << "Record delivery time [" << recordInfo.getRecordDeliveryTimeMs() << " ms]." << std::endl;
} catch (std::exception& e) {
    std::cout << "Exception was caught while waiting for callback result: " << e.what() << std::endl;
}

```

Slika 4.1 – Odsečak programskog koda koji sadrži logiku slanja podataka i prevezivanja C++ funkcija u JavaScript

Kao krajnji rezultat prevodenja ovako napisane aplikacije, ostaje JavaScript funkcija za slanje parametara na Kaa server. Njoj se prosleđuju tačno definisani parametri koji odgovaraju kolonama tabele u bazi podataka u koju se šalju. Ovi parametri su u Node.js dodatku konvertovani i prilagođeni su im tipovi.

U okviru definicije glavne metode Node.js dodatka koja na ulazu prima niz argumenata, ostvaruje se parsiranje ulaznih podataka i njihovo smeštanje u promenjive tipa V8::String. Pri preuzimanju ulaznog niza neophodno je naznačiti u kom formatu se parametar parsira. Pošto želimo da ga preuzmemosmo kao niz karaktera, poziva se metoda *toString()*.

Na osnovu funkcionalnosti implementirane u C++ funkciji, generiše se odgovarajuća JavaScript funkcija. Parametri ove funkcije su standardni JavaScript String tipovi. Generisanje JavaScript funkcije se obavlja pozivom NODE_SET_METHOD metode. Ovu metodu takođe obezbeđuje V8 biblioteka. Primer upotrebe ove metode može se videti u delu koda prikazanom ispod gde je obuhvaćen deo celokupnog programskog koda Node.js dodatka.

```

void Init(Local<Object> exports)
{
  NODE_SET_METHOD(exports, "send", SendData)
}

```

Drugi parametar ove metode predstavlja naziv koji će se koristi u okviru JavaScript modula, a treći parametar je postojeći naziv metode u node.js modulu.

Za ovako napisan program, potrebno je napisati i konfiguracionu datoteku na osnovu koje će se kasnije vršiti prevođenje. Ova datoteka se čuva pod nazivom „binding.gyp“ i u njoj se definiše ciljni objekat nad kojim će biti moguće pozivati napravljenu JavaScript funkciju.

Na slici 4.2 prikazan je sadržaj ove biblioteke. Kao jedan od parametara zadaje se i .cc fajl u kojem je ostvarena funkcionalnost modula (main.cc u ovom rešenju).

```

{
  "targets": [
    {
      "target_name": "addon",
      "sources": ["main.cc"],
      "include_dirs": ["kaa", "/usr/local/include/botan-1.11"],
      "libraries": [
        "/home/rtrk/kaa-cpp-ep-sdk-c5hftE46cW5kWERzn7KuVSvFugA/kaa/libkaacpp.a",
        "/usr/local/lib/libavrocpp.so",
        "/usr/local/lib/libbotan-1.11.so",
        "/usr/local/lib/libboost_system.so",
        "/usr/local/lib/libboost_thread.so",
        "/usr/local/lib/libboost_log.so"],
      "cflags!": ["-fno-exceptions"],
      "cflags_cc!": ["-fno-rtti", "-fno-exceptions"]
    }
  ]
}

```

Slika 4.2 – Izgled Binding.gyp datoteke

Takođe je potrebno uključiti biblioteke sa njihovom apsolutnom putanjom u okviru sistema datoteka mašine. Neophodne biblioteke su:

- Generisana statička .a biblioteka od prevedonog C++ SDK-a (libkaacpp.a)
- Dinamička biblioteka Avro [12] koja obuhvata: alate za sklapanje programskih šema, kao i odgovarajuće alate za njihovo parsiranje. Takođe obuhvata enkodere i dekodere za Avro format, JSON, binarni i validatorni. Ova biblioteka obezbeđuje kod generator, koji generiše programske kodove C++ klasa i funkcija. Generiše se u obliku C++ *header* datoteke.
- Dinamička biblioteka Botan [13] koja obezbeđuje kriptografska svojstva. Pisana je u C++11 pod BSD licencom.
- Dinamička biblioteka Boost [14] koja obezbeđuje ubrzanje pri razvoju bez grešaka pri ostvarivanju ubrzanja.

Ovako napisana datoteka se prevodi korišćenjem *node-gyp* alata koji se distribuira putem NPM menadžera paketa za JavaScript. Ovaj alat služi upravo prevođenju C++ koda u Node.js dodatke (eng. *addons*).

Prvi korak u prevodenju je izdavanje komande *node-gyp configure* za generisanje odgovarajućih datoteka za prevodenje i pokretanje. U zavisnosti od platforme na kojoj se vrši

prevodenje, generise se *Makefile* (Linux platforma) ili vcxproj datoteka (Windows platforma) u okviru novonastalog direktorijuma za prevodenje (eng. *build*).

Za konačno generisanje prevedene datoteke addon.node koristi se komanda *node-gyp build*. Putanja ovde datoteteke je Build/Release/ direktorijum.

Iz ovako prevedene datoteke koristimo objekat addon nad kojim se poziva metoda *send* u okviru JavaScript dela programske podrške. Tako je premošćen problem nekompatibilnosti programskih jezika SDK-a i Kaa klijentske aplikacije sa funkcijom modula „Oblo“ sistema za glasovnu korisničku spregu. Sada je jasan i motiv odabiranja C++ kao tipa SDK-a pre ostalih tipova.

Send funkciji se prosleđuju šest parametara koji odgovaraju kolonama tabele u bazi (zahtev, tip zahteva, vrednost, kontrolisana osobina uređaja, uređaj i korisnički identifikacioni kod). Konačno, ova JavaScript funkcija se poziva u okviru glavnog modula klijentske aplikacije za glasovnu korisničku spregu. Mesto poziva je deo programske podrške gde se prihvata odgovor centralnog uređaja na zadatu komandu. Na osnovu ovog odgovora popunjavaju se polja koja se prosleđuju *send* funkciji i bivaju poslata na Kaa server i prosleđena u bazu podataka.

5. Rezultati

Ispravan rad kreirane klijentske aplikacije za prikupljanje podataka potvrđen je u okviru „Oblo“ sistema kućne automatizacije. Testiranje je izvršeno na stvarnim podacima testnog korisnika, za sve raspoložive tipove komandi i uređaja.

Sandbox instanca Kaa servera pokrenuta je na virtuelnoj mašini pod Linux operativnim sistemom. Takođe, pokrenuta je i testna instanca modula za glasovnu korisničku spregu, zasnovanog na Amazon Aleksi. U okviru ovog modula, implementirano je slanje podataka ka Kaa platformi.

Na slici 5.1 prikazan je izgled prazne baze dok još nije bila izdata ni jedna komanda od strane klijentske aplikacije za prikupljanje podataka. Pristup Kasandra bazi u okviru virtuelne mašine, tj. Sandbox instance, vrši se komandom *cqlsh*. Zatim se komandama CQL jezika pristupa željenoj tabeli.

```
[cqlsh 5.0.1 | Cassandra 3.5 | CQL spec 3.4.0 | Native protocol v4]
Use HELP for help.
[cqlsh:kaa]> use kaa
...
[cqlsh:kaa]> SELECT * from oblo_kaa_logs;
Empty set (0 rows)
[cqlsh:kaa]> SELECT * from ep_user;
Empty set (0 rows)
[cqlsh:kaa]>
```

Slika 5.1 – Izgled prazne baze pri inicijalizaciji

Da bi se eliminisali slučajevi u kojima do modula za glasovnu korisničku spregu dospeva loše formirana komanda, usled problema prouzrokovanih od strane Amazonovog servisa za prepoznavanje govora, za potrebe testiranja sprege sa Kaa sistemom koristi se testni Alexa web servis. Na slici 5.2 vidi se izgled ovog okruženja i izgenerisani JSON koji se šalje u modul za glasovnu korisničku spregu. Na osnovu iskucanog teksta koji odgovara komandi koja bi bila izgovorena, generiše se odgovarajući JSON objekat.

The screenshot shows the Alexa Web Simulator interface. At the top, there are two tabs: "Text" (which is selected) and "JSON". Below the tabs, there is a section titled "Enter Utterance" containing the text "get temperature from sensor". Underneath this is a button labeled "Ask official oblo skill" and a "Reset" button. The main area is divided into two sections: "Service Request" and "Service Response". The "Service Request" section contains a JSON object with 16 numbered lines. The "Service Response" section also contains a JSON object with 14 numbered lines. A "Listen" button is located at the bottom right of the response panel.

```

1 {
2   "session": {
3     "sessionId": "SessionId.3c2b6897-1b47-426d-
4     "application": {
5       "applicationId": "amzn1.ask.skill.e0311fb
6     },
7     "attributes": {},
8     "user": {
9       "userId": "amzn1.ask.account.AG76L4Q537PF
10      "accessToken": "e093b1a35b4e40a26f37fd59
11    },
12    "new": false
13  },
14  "request": {
15    "type": "IntentRequest",
16    "requestId": "EdwRequestId.0f4f6fec-c2d4-4b

```

```

1 {
2   "version": "1.0",
3   "response": {
4     "outputSpeech": {
5       "type": "PlainText",
6       "text": "28"
7     },
8     "card": {
9       "content": "SessionSpeechlet - 28",
10      "title": "SessionSpeechlet - IntentReq
11      "type": "Simple"
12    },
13    "reprompt": {
14      "outputSpeech": {

```

Slika 5.2 – Simulacija glasovne komande za pribavljanje vrednosti sa senzora korisnika u Alexa web testnom okruženju

Na slici 5.2 može se videti i odgovor sistema na komandu, takođe u JSON formatu. Ovaj odgovor se u okviru klijentske aplikacije za prikupljanje podataka parsira, odgovarajuće vrednosti se smeštaju u njima namenjena polja i šalju. Prikaz ovog procesa se nalazi na slici 5.3.

The screenshot shows a terminal window with several tabs. The active tab displays Java code for a module named 'godjs'. The code includes logic for handling sensor values and generating speech output based on device intents like 'SensorIntent' or 'Contact'. It also interacts with a database named 'diplomski_logovi' using CQLSH to select rows corresponding to specific device intents and their properties.

```

godjs
581ac26136b963b3a62f | "opened"
(1 rows)
cqlsh:kaa> SELECT * from diplomski_logovi ;
+-----+-----+-----+-----+
| date | device | intent | property | request |
| rid  |        | value   |          |          |
+-----+-----+-----+-----+
2017-06-30-Time:09:02:08:390 | sensor | SensorIntent | contact | get | 57d
a581ac26136b963b3a62f | "opened"
2017-06-30-Time:09:01:21:26 | sensor | SensorIntent | contact | get | 57d
a581ac26136b963b3a62f | "opened"
(2 rows)
cqlsh:kaa> SELECT * from diplomski_logovi ;
+-----+-----+-----+-----+
| date | device | intent | property | request |
| userid |        | value   |          |          |
+-----+-----+-----+-----+
2017-06-30-Time:09:05:07:825 | sensor | SensorIntent | temperature | get | 57da581ac26136b963b3a62f | 28
2017-06-30-Time:09:02:08:390 | sensor | SensorIntent | contact | get | 57da581ac26136b963b3a62f | "opened"
2017-06-30-Time:09:01:21:26 | sensor | SensorIntent | contact | get | 57da581ac26136b963b3a62f | "opened"
(3 rows)
cqlsh:kaa>

```

Slika 5.3 – Prikaz programskog koda modula za glasovnu korisničku spregu odakle se presreću tražene vrednosti

Na slici 5.4 prikazana je baza sa prikupljenim testnim podacima. Red tabele sadrži vreme izdavanja komande, zatim ime uređaja koji se kontroliše, te vrstu komande koja se izdaje, a koja odgovara komandama iz Aleksa modula (LightIntent, DimIntent i slično). Sledeći podatak koji se čuva je tip parametra koji se kontroliše (temperatura, nivo osvetljenja, boja svetla), konkretna komanda koja se zadaje (paljenje, gašenje, postavljanje boje, postavljanje nivoa osvetljenja) i vrednost na koju se kontrolisani parametar postavlja. Vrednost se tumači na osnovu tipa parametra (npr. za boju vrednost je crvena, a za procenat osvetljenosti vrednost je broj).

Mogući tipovi komandi su:

- Komande za paljenje/gašenja svetala - *LightIntent*
- Komande za podešavanje nivoa osvetljenosti – *DimIntent*
- Komande za pribavljanje vrednosti sa senzora – *SensorIntent*
- Komande za podešavanje boja osvetljenja – *LightIntent*

Prilikom testiranja uspešno je ostvareno beleženje svih izdatih komandi, za raspoložive uređaje u okviru „Oblo“ sistema pametne kuće. Zabeležene su sledeće komande:

- „*Turn off lamp*“. Preuzima se *lamp* kao ime uređaja i *turn off* kao korisnička komanda. Ova komanda spada pod tip komande za paljenje/gašenje svetla *LightIntent*, a polje za tip parametra je stanje uređaja. Vrednost koja odgovara tipu parametra je ugašeno (*off*).

- „Set dimmer to fifty“. Preuzima se *dimmer* kao ime uređaja i *set* kao korisnička komanda. Ova komanda spada pod tip komande za podešavanje osvetljenosti *DimIntent*, a polje za tip parametra je nivo osvetljenosti. Vrednost koja odgovara tipu parametra je 50.
- „Set lamp color to red“. Preuzima se *lamp* kao ime uređaja i *set* kao korisnička komanda. Ova komanda spada pod tip komande za podešavanje boje lampe *LightIntent*, a polje za tip parametra je boja svetla. Vrednost koja odgovara tipu parametra je crvena boja.
- „Get temperature from temperature sensor“. Preuzima se *sensor* kao ime uređaja, a *get* kao korisnička komanda. Ova komanda spada pod tip komande za preuzimanje vrednosti sa senzora *SensorIntent*, a polje za tip parametra je temperatura. Vrednost koja odgovara tipu parametra je 25 (temperatura vazduha).

„Get contact state from sensor“. Preuzima se *sensor* kao ime uređaja i *get* kao korisnička komanda. Ova komanda spada pod tip komande za preuzimanje vrednosti sa senzora *SensorIntent*, a polje za tip parametra je informacija o tome da li je otvoren/zatvoren uređaj. Vrednost koja odgovara tipu parametra je da je da je senzor otvoren (tj. otvoreni su vrata ili prozor na kojima se senzor nalazi).

Time	Device	Intent Type	Parameter Type	Value	Action
2017-06-30-Time:09:42:41:891	lamp	LightIntent	Light State	turn off	
57da581ac26136b963b3a62f	off				
2017-06-30-Time:09:43:03:895	dimmer	DimIntent	Percentage	50	
57da581ac26136b963b3a62f	50				
2017-06-30-Time:09:42:20:780	lamp	LightIntent	Light State	turn on	
57da581ac26136b963b3a62f	on				
2017-06-30-Time:09:45:28:849	lamp	LightIntent	Color	red	
57da581ac26136b963b3a62f	red				
2017-06-30-Time:09:05:07:825	sensor	SensorIntent	temperature	28	
57da581ac26136b963b3a62f	28				
2017-06-30-Time:09:43:41:329	dimmer	LightIntent	Light State	turn off	
57da581ac26136b963b3a62f	off				
2017-06-30-Time:09:41:48:100	sensor	SensorIntent	temperature	25	
57da581ac26136b963b3a62f	25				
2017-06-30-Time:09:02:08:390	sensor	SensorIntent	contact	opened	
57da581ac26136b963b3a62f	opened				
2017-06-30-Time:09:37:52:169	sensor	SensorIntent	temperature	22	
57da581ac26136b963b3a62f	22				
2017-06-30-Time:09:43:55:216	lamp	LightIntent	Light State	turn off	
57da581ac26136b963b3a62f	off				
2017-06-30-Time:09:43:21:860	dimmer	DimIntent	Percentage	70	
57da581ac26136b963b3a62f	70				
2017-06-30-Time:09:01:21:26	sensor	SensorIntent	contact	opened	
57da581ac26136b963b3a62f	opened				
2017-06-30-Time:09:45:05:205	lamp	LightIntent	Color	blue	
57da581ac26136b963b3a62f	blue				

```

Open ▾
var device = event.request.intent.slots.Thing;
var request;
var property;
var value;

//event.session.user.userId;

index.handler(event, function(message) {
  console.log('message from Alexa Module');
  res.writeHead(200, {
    "Content-Type": "application/json"
  });

  switch (intent) {
    case "LightIntent":
      request = event.request.intent.s;
      if(request === 'change') {
        property = "Color";
        value = event.request.intent.s;
      } else if(request === 'turn on') {
        property = "Light State";
        value = "on";
      } else if(request === 'turn off') {
        property = "Light State";
        value = "off";
      }
      break;
    case "DimIntent":
      request = event.request.intent.s;
      property = "Percentage";
      value = event.request.intent.s;
      break;
    case "SensorIntent":
      request = event.request.intent.s;
      property = event.request.intent.slots.SensorService.value;
      value = JSON.stringify(message.response.outputSpeech.text);
  }

  (14 rows)
  cqlsh:kaa>

```

Slika 5.4 – Prikaz Kasandra tabele za čuvanje korisičkih komandi i zahteva

6. Zaključak

U ovom radu prikazan je mehanizam beleženja korisničkih informacija iz postojećeg sistema za kućnu automatizaciju „Oblo“. Ova sistem je centralizovan sa centrom u vidu centralnog uređaja (eng. *gateway*) na koji se povezuju periferni uređaji (eng. *nodes*). On ima ulogu unificiranja pristupanja uređajima i uvodi novi nivo apstrakcije i dodatnu logiku oko uređaja.

Komunikacija sa uređajima se može vršiti iz lokalne mreže direktnom komunikacijom sa centralnim uređajem mrežnog rutera ili posredstvom servisa u oblaku (*Oblo cloud*) kada se korisnik nalazi van lokalne mreže. Korisnik kontroliše uređaje putem 3 tipa sprege:

- Web sprega – interfejs dostupan preko web pregledača (eng. *web browser*) na Internetu gde korisnik sistema može da kontroliše uređaje, ali i administrator
- Aplikativna sprega – zadavanje komandi i manipulacija sistemom putem mobilne aplikacije. Podržan je i Android i IoS klijent.
- Glasovna korisnička sprega – omogućava zadavanje komandi glasom. Obezbeđena je uvođenjem nove fizičke komponente u sistem (Amazon Echo ili namenski korisnički uređaj) i povezivanjem sa Amazon servisom u oblaku - Aleksa koji vrši funkciju prepoznavanja na osnovu zadatih šema i povezuje sa programskom podrškom koja definiše logiku sistema.

Kaa platforma je iskorišćena kao posrednik pri slanju podataka u Kasandra bazu podataka zbog brzine, pouzdanosti i sigurnosti pri transakciji podataka. Ova platforma, nezavisna od arhitekture i tipa fizičkih uređaja, obezbeđuje neophodne alate za razvoj programske podrške za prikupljanje podataka o sistemu. Ovim se služi klijentska aplikacija koja je kreirana. S obzirom da je odlučeno da se funkcionalnost inkorporira u okvitu klijentskog modula za glasovnu korisničku spregu, bilo je neophodno prilagoditi programsku podršku programskom

jeziku postojećeg modula. Izvršena je adaptacija programskog koda iz C++ u JavaScript programski jezik bez gubitaka funkcionalnosti.

Ostvareno je beleženje korisničkih komandi zadatih putem Amazon web interfejsa. Rezultati su pribeleženi u Kasandra bazi podataka koja je odabrane zbog jednostavnosti manipulisanja, kao i pouzdanosti i kapaciteta.

Dalji razvoj će se vršiti u smeru potpune integracije Kaa platforme za prikupljanje podataka sa „Oblo“ sistemom u oblaku. U budućnosti, cilj je omogućiti i analizu sakupljenih podataka, radi obezbeđivanja autonomnosti i inteligencije sistema kućne automatizacije.

7. Literatura

- [1] Milan Tucić: IoT proširenje za povezivanje sistema za automatizaciju kuće korišćenjem M2M protokola, Univerzitet u Novom Sadu, Fakultet Tehničkih Nauka, 2015.
- [2] V. Lampkin, W. Tat Leong, L. Olivera, S. Rawat, N.Subrahmanyam, and Authors: Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry, IBM Redbooks, 2012.
- [3] Aleksa servis, [Online] Dostupno na: <https://developer.amazon.com/alexa>, jul 2017.
- [4] Vodič za pisanje funkcionalnosti Aleksa modula, [Online] Amazon Web Services, Inc. and/or its affiliates: AWS Lambda Developer Guide, 2017
- [5] Kaa Platforma, [Online] Dostupno na: <https://www.kaaproject.org/overview>, jul 2017.
- [6] A.Chebotko, A. Kashlev and S. Lu, “A Big Data Modeling Methodology for Apache Cassandra”, *Proc. of IEEE 2015 International Congress on Big Data – New York*, 2015
- [7] Kasandra baza podataka, [Online] Dostupno na: <http://cassandra.apache.org>, jul 2017.
- [8] Mongo baza podataka, [Online] Dostupno na: <https://docs.mongodb.com>, jul 2017.
- [9] [RFC2616], R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, “Hypertext Transfer Protocol - HTTP/1.1”, jun 1999
- [10] Vodič kroz Node.js dodatke, [Online] Dostupno na:
<https://nodejs.org/api/addons.html>, jul 2017.
- [11] V8 okruženje, [Online] Dostupno na: <https://v8docs.nodesource.com>, jul 2017.

- [12] Avro Apache biblioteka, [Online] Dostupno na: <https://avro.apache.org/docs>, jul 2017.
- [13] Botan biblioteka, [Online] Dostupno na: <https://botan.randombit.net>, jul 2017.
- [14] Boost biblioteka, [Online] Dostupno na: <http://www.boost.org/>, jul 2017