

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1097

HETEROGENA OKOLINA ZA INTERNET STVARI

Marko Veizović

Zagreb, lipanj 2017.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 3. ožujka 2017.

DIPLOMSKI ZADATAK br. 1097

Pristupnik: **Marko Veizović (0036465810)**
Studij: Informacijska i komunikacijska tehnologija
Profil: Telekomunikacije i informatika

Zadatak: **Heterogena okolina za Internet stvari**

Opis zadatka:


Sustavi Interneta stvari mogu obuhvaćati velik broj povezanih uređaja, a svaki od njih može prikupljati različite podatke iz svoje okoline i utjecati na okolinu (npr. mjeriti temperaturu, tlak ili paliti/gasiti žarulju i sl.). S obzirom da je broj uređaja u takvoj mreži nužno konačan, iako se mogu dodavati i novi uređaji, cilj je maksimalno iskoristiti postojeću infrastrukturu za pružanje različitih složenih usluga. U jednom trenutku pojedini uređaj može sudjelovati u samo jednoj složenoj usluzi, stoga se mora izvršiti odabir koji će uređaji pružati svoje podatke za koju uslugu, uz nastojanje da se maksimizira njihova korisnost. Vaš zadatak je realizirati heterogenu okolinu Interneta stvari koja obuhvaća fizičke, emulirane i virtualne krajnje uređaje odnosno izvore podataka. Potrebno je definirati model složene usluge na temelju koje se pristupa dostupnim podatkovnim izvorima. Opis karakteristika uređaja i jednostavnih usluga koje su sposobni obavljati treba realizirati pomoću baze podataka s podrškom za semantički opis podataka.

Svu potrebnu literaturu i uvjete za rad osigurat će Vam Zavod za telekomunikacije.

Zadatak uručen pristupniku: 10. ožujka 2017.
Rok za predaju rada: 29. lipnja 2017.

Mentor:

Prof. dr. sc. Gordan Ježić

Djelovoda:

Izv. prof. dr. sc. Lea Skorin-Kapov

Predsjednik odbora za
diplomski rad profila:


Izv. prof. dr. sc. Miljenko Mikuc

Sadržaj

1.	Uvod.....	1
2.	Internet stvari	2
2.1.	Definicija.....	2
2.2.	Arhitektura	3
2.3.	Primjena i izazovi.....	5
3.	Realizacija heterogene okoline Interneta stvari	7
3.1.	Grafička baza podataka	7
3.1.1.	Neo4j	8
3.1.2.	Cypher upitni jezik	12
3.2.	Opis izvora podataka.....	13
3.3.	Povezivanje izvora podataka	20
3.3.1.	Primjeri upita nad Neo4j bazom podataka	22
4.	Usluge u Internetu stvari	26
4.1.	Uslužno orijentirana arhitektura.....	26
4.2.	REST	28
4.2.1.	Primjeri REST zahtjeva nad Neo4j bazom podataka	31
4.3.	Model složene usluge u Neo4j bazi podataka	35
5.	Zaključak.....	38
6.	Literatura.....	39
7.	Sažetak	40

1. Uvod

Suvremeni svijet gotovo je nezamisliv bez korištenja tehnoloških dostignuća, pa se u tom smislu, daljnjim napretkom trenutnih i razvojem nekih novih mogućnosti, nastoje poboljšati svi aspekti ljudskog života. Upravo zbog toga, kao i zbog činjenice da je Internet najpopularnija i najraširenija mreža na svijetu, skovan je pojam Interneta stvari (eng. *Internet of Things*, **IoT**), kao principa u kojem se brzo i efikasno mogu izvršiti najrazličitiji tipovi usluga, a gdje je jedini važan uvjet da su krajnji korisnici povezani u mrežu te se mogu identificirati. Premda je takav koncept itekako uspješan zbog gore navedenih razloga, činjenica je da se bilo koji uređaj može definirati kao „stvar“ koja šalje, prima ili obrađuje/pohranjuje podatke, što označava najveći izazov IoT-a, a to je heterogenost sustava, tj. kako dogovoriti parametre komunikacije između izvora podataka koji se zasnivaju na različitim tehnološkim izvedbama i komunikacijskim protokolima. Najčešći načini za rješenje tog problema su ili izravna komunikacija između uređaja putem općeprihvaćenih komunikacijskih protokola, ili centralni čvorovi za pohranu (najčešće baze podataka) kojima onda svaki uređaj pristupa tražeći samo ono što mu u danom trenutku treba. Ovaj potonji način bit će razjašnjen u ovom diplomskom radu. Prvo će se ukratko opisati koncept Interneta stvari, objasniti njegova arhitektura, kao i područja primjene te najveći izazovi u njegovom uspješnom funkcioniranju. Potom će se opisati na koji je način realizirana baza podataka koja služi za povezivanje heterogenih izvora podataka, a koja ima podršku za semantički opis podataka. Tu se radi o grafičkoj bazi podataka koja veliki značaj pridaje vezama između podataka zbog čega se na brz i jednostavan način može pristupiti određenom podatkovnom skupu. U radu će se naglasiti razlike između svojstava grafičkih baza podataka u odnosu na relacijske i nerelacijske baze. Nadalje, objasnit će se načini definiranja i kreiranja jednostavnih, ali i složenijih usluga Interneta stvari koje imaju za cilj što je moguće više iskoristiti postojeću infrastrukturu, te će se prikazati na koji je način realiziran pristup gore spomenutim podatkovnim izvorima.

2. Internet stvari

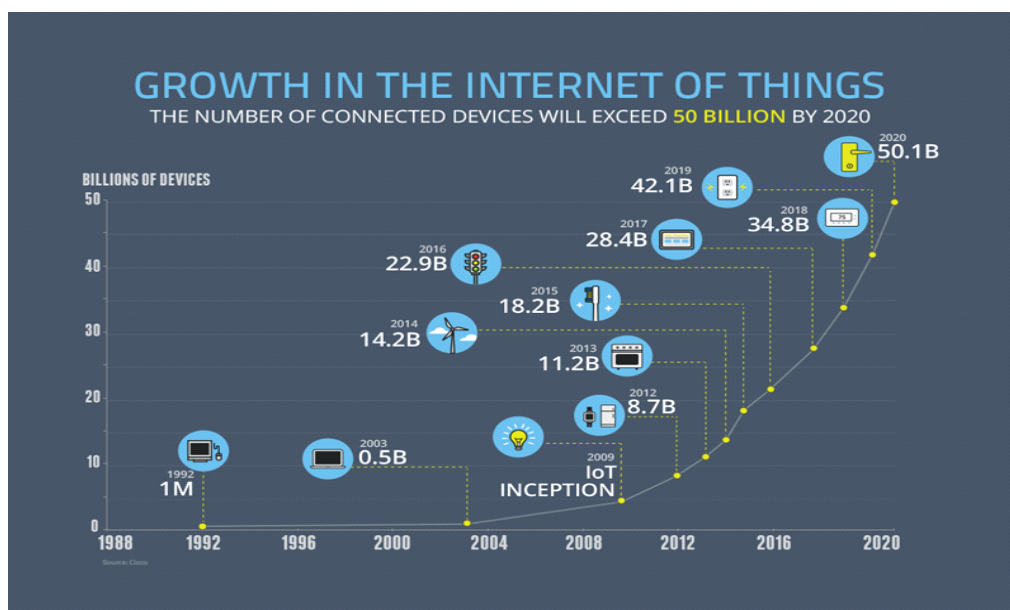
Internet stvari je koncept kojeg izgrađuju stvari ili objekti. Objekt može biti nešto opipljivo, iz stvarnog svijeta kao npr. senzori ili aktuatori, a može se koristiti i virtualni tip objekta [1]. U nastavku su dana definicije pojma, opis pripadajuće arhitekture te je objašnjeno gdje se sve taj koncept može koristiti.

2.1. Definicija

Premda ne postoji jedinstvena definicija Interneta stvari, jedna od šire prihvaćenih je ona iz preporuke ITU-T-a nastale 2012. godine prema kojoj je Internet stvari globalna infrastruktura informacijskog društva koja omogućuje međusobno povezivanje virtualnih i fizičkih stvari i koja se bazira na postojećim i razvijajućim interoperabilnim informacijsko-komunikacijskim tehnologijama [2]. Nadalje, kroz iskorištavanje identifikacije, prikupljanja i obrade podataka te komunikacijskih sposobnosti, Internet stvari u potpunosti koristi svojstva umreženih objekata kako bi ponudio usluge za razne vrste aplikacija, dok se pritom vodi računa o očuvanju sigurnosnih zahtjeva poput tajnosti podataka te identifikacije i autentifikacije korisnika. U široj perspektivi, Internet stvari je vizija tehnološko-društvene isprepletenosti. RFID grupa definira IoT kao svjetsku mrežu međusobno povezanih objekata, od kojih svaki posjeduje jedinstvenu adresu i komunicira putem standardnih komunikacijskih protokola. Europski klaster istraživačkih projekata vezanih uz IoT (*IoT European Research Cluster*, IERC) navodi kako je IoT integralni dio budućeg Interneta, a definira ga kao dinamičku globalnu mrežnu infrastrukturu s mogućnošću samo-konfiguracije, tj. mrežu koja se bazira na standardnim i interoperabilnim komunikacijskim protokolima, te mrežu u kojoj fizičke i virtualne stvari imaju identitet, fizička obilježja i virtualne osobnosti, a također koriste inteligentna sučelja, dok su istovremeno neprimjetno integrirane u informacijskoj mreži [2].

Kada pojedini objekt dobije neki jedinstveni identifikator preko kojeg mu se može pristupiti, tada je on umrežen i povezan na Internet (eng. *Internet connected object*, ICO). Glavna značajka ICO-a je da može primati i odašiljati podatke, tj. komunicirati s drugim umreženim objektima, kao i primati podatke o stanju u mreži i naredbe za konfiguraciju. Osim toga, može

izvršiti i pojedine aktivnosti, npr. upaliti/ugasiti svjetlo i sl. Na Slici 1 vidljiv je eksponencijalni rast broja umreženih uređaja s predviđanjem da će do 2020. godine taj broj doseći pedeset milijardi [1].



Slika 1 - Prikaz porasta broja umreženih uređaja

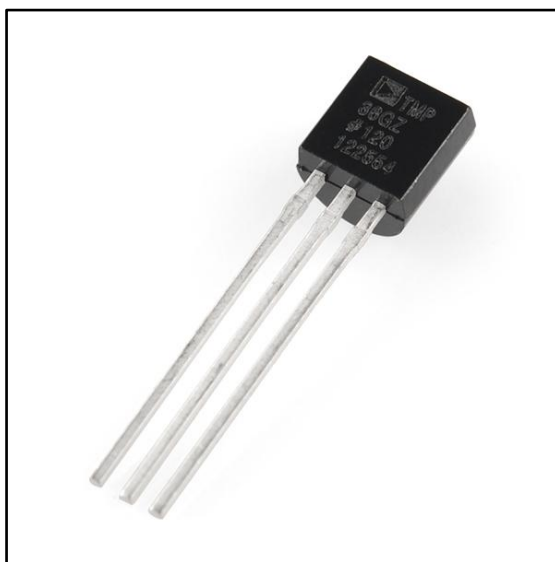
(IZVOR:<https://www.ncta.com/platform/broadband-internet/behind-the-numbers-growth-in-the-internet-of-things-2/>)

2.2. Arhitektura

Pod arhitekturu Interneta stvari ubrajamo sve uređaje i elemente koji sudjeluju u ostvarivanju usluga, počevši od manjih senzora koji prikupljaju i odašilju podatke, pa sve do npr. pametnih telefona na kojima krajnji korisnici ostvaruju pristup usluzi. Između tih krajeva postoje određeni slojevi od kojih svaki posjeduje određeni skup funkcionalnosti i može komunicirati sa susjednim slojevima. Osnovni cilj je napraviti arhitekturu uz što nižu cijenu operativnih troškova izgradnje i održavanja, a da se pritom može realizirati usluga zadovoljavajuće kvalitete [4]. S obzirom na to da ne postoji jedinstvena platforma Interneta stvari, već svaki pojedini ponuđač usluge nudi svoje izolirano rješenje, teško je definirati referentnu arhitekturu sustava. Nadalje, ne postoji jedinstven davatelj resursa koji se proteže duž cijelog vertikalnog rješenja za uslugu, već se cjelokupna arhitektura ostvaruje suradnjom npr. kompanije koja proizvodi senzore i programera koji izrađuju aplikacije vezane uz te

senzorske podatke. Unatoč tome, sva rješenja sadrže neke zajedničke osobine, pa se prema tome definira neka osnovna arhitektura Interneta stvari.

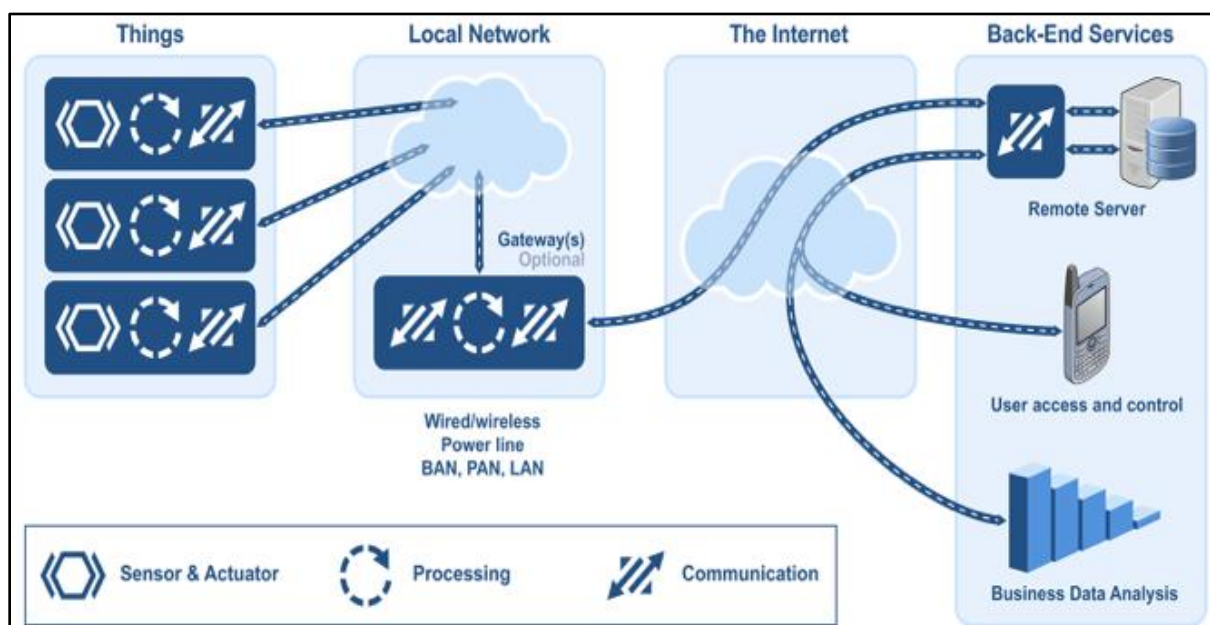
Prema pojednostavljenom prikazu infrastrukture Interneta stvari, na dnu arhitekture nalazi se mreža senzorskih čvorova. Njihova se funkcionalnost zbog ograničenih resursa (mala procesorska snaga i dostupnost energije) svodi na jednostavno očitavanje zadanih parametara iz okoline i prosljeđivanje tih podataka sljedećem sloju. Za senzore je važno da budu neosjetljivi na bilo koju drugu pojavu osim one koju mjere jer svako krivo očitavanje dovodi do neispravnosti u radu usluge, pa je stoga veoma važno stručno rukovanje i redovito kontroliranje i održavanje tih uređaja [3]. Na Slici 2 nalazi se senzor koji služi za očitavanje vrijednosti temperature zraka.



Slika 2 – Senzor za mjerenje temperature

Čvorovi koji prihvaćaju senzorske podatke te posjeduju funkcionalnost obrade tih podataka, kao i kontrole senzora, nazivaju se pristupnim čvorovima (eng. *gateway*). Oni služe kao posrednici između senzora i viših slojeva. Komunikacija prema sensorima, kao i višim slojevima, ostvaruje se putem prikladnih komunikacijskih protokola. U višim slojevima arhitekture ostvarena je pohrana i složena obrada podataka te obrada zahtjeva poslanih od strane korisničke aplikacije. Tamo su računala organizirana u strukturu računalnog oblaka čije karakteristike odgovaraju konceptu Interneta stvari. Neke od tih karakteristika su transparentnost, tj. skrivanje nepotrebnih informacija krajnjem korisniku i dobra skalabilnost, odnosno brza prilagodba povećanom broju krajnjih korisnika uz očuvanje dobrih performansi i očekivane kvalitete usluge [1]. Na vrhu se nalaze krajnji korisnici koji koriste kreirane

usluge pristupajući im putem aplikacija instaliranih na vlastitim uređajima (računala, pametni telefoni, tableti i sl.). Osim krajnjih korisnika, prikupljeni podaci ponekad se koriste i u svrhu poslovne inteligencije po kojoj se onda prikuplja i analizira velika količina podataka (eng. *Big Data*) kako bi daljnji razvoj i napredak neke usluge bio što efikasniji i prilagođeniji korisnicima. Na Slici 3 nalazi se pojednostavljena skica koja odgovara opisanoj arhitekturi Interneta stvari.



Slika 3 – IoT arhitektura (IZVOR: architectcorner.yolasite.com)

2.3. Primjena i izazovi

Kako je u infrastrukturu Interneta stvari moguće umrežiti razne tipove uređaja, tako su područja primjene za koje se može ostvariti neka usluga veoma raznolika. S ciljem pojednostavljenja zadataka i brzog rješavanja brojnih problema, odnosno generalnog povećanja kvalitete ljudskog života neke od značajnijih primjena uključuju princip pametne kuće (eng. *Smart Home*), princip pametnog grada (eng. *Smart City*) te razne primjene u industriji, očuvanju okoliša, trgovini, birokraciji i mnoge druge.

Princip pametne kuće (Slika 4) predstavlja jednostavno rukovanje kućanskim aparatima, primjerice pomoću pametnih telefona ili tableta. Na taj se način može upravljati osvjetljenjem, grijanjem/hlađenjem, sigurnosnim sustavom, rukovati elektroničkim uređajima i sl.

Vrlo popularna primjena senzora je i u konceptu pametnog grada koja omogućuje povećanje kvalitete življenja u urbanoj sredini gdje se mjerenjem raznih pojava u prometnom, zdravstvenom, vodoopskrbnom i energetske sektoru povećava protočnost ljudi i stvari uz značajnije smanjenje troškova upotrebe resursa te očuvanje okoliša. Jedan od primjera može biti praćenje slobodnih mjesta na parkiralištu kako bi se izbjegle nepotrebne gužve i komplikacije. Također, pametna gradska rasvjeta, ako je kvalitetno kontrolirana, može pridonijeti uštedama u odnosu na standardno osvjetljenje. Vrlo važna primjena je i sigurnost građana i gradske infrastrukture gdje se sustavnim praćenjem događanja može brže reagirati ako dođe do nepogode ili nesreće.

U industriji je moguće pomoću senzorskog sustava nadgledati svaki korak u procesu proizvodnje, uz kontrolu svake proizvedene jedinice i dojavu kod otkrivanja pogreške. U ekološkom smislu, svaku pojavu u okolišu, poput koncentracije plinova u zraku, temperature, vlažnosti itd. moguće je sustavno pratiti.

S obzirom na sve gore navedene osobine Interneta stvari, vidljiv je veliki potencijal u takvom konceptu i jasno je kako će se nastaviti s ulaganjem i razvijanjem s ciljem da se broj korisnika i dalje povećava. Kako bi se taj trend uspješno nastavio potrebno je suočiti se s nekim izazovima koje Internet stvari donosi. Najvažniji od njih je gore spomenuta heterogenost uređaja i nedostatak jedinstvene platforme. [1] Osim toga, povećanjem broja korisnika, odnosno umreženih objekata, potrebno je uložiti i više vremena u održavanje sustava, kao i u dodatne resurse koji bi omogućili kvalitetan rad s ogromnom količinom podataka. Uvijek važna stavka je i pitanje sigurnosti, gdje svaki neispravan rad neke usluge potencijalno može ugroziti ljudski život te pitanje privatnosti budući da se korisnički podaci nalaze na nekoj korisniku nepoznatoj lokaciji koja može biti pogodna za napade.



Slika 4 - Princip pametne kuće (IZVOR: <http://www.makeuseof.com/tag/much-smart-home-really-cost/>)

3. Realizacija heterogene okoline Interneta stvari

U ovom poglavlju opisana su svojstva i načini korištenja grafičke baze podataka Neo4j te je objašnjen način na koji se baza popunila podacima iz relevantnih izvora. Nadalje, nabrojani su i opisani svi ti izvori podataka te je prikazan način na koji su se pripremili prije unosa u bazu. Sve gore navedeno prikazuje postupak kojim se ostvarila heterogena okolina, tj. kontekst koji objedinjuje različite podatke iz različitih izvora te ih obrađuje tako da postanu smisljena cjelina koja se može dalje iskoristiti, primjerice kod kreiranja usluga za koje ti podaci imaju određeno značenje.

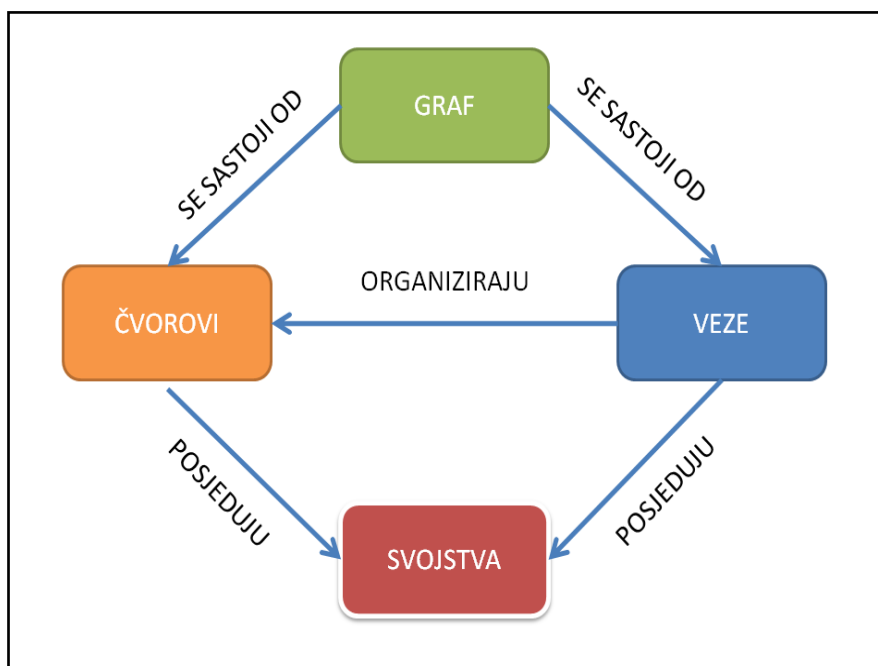
3.1. Grafička baza podataka

Grafička baza podataka, kao što njezin naziv govori, jest baza podataka u kojoj se podaci spremaju u obliku grafova nad kojima se onda po potrebi mogu izvoditi različiti upiti. Korisnost ovakvog tipa pohrane leži u činjenici da su podaci u grafu dobro povezani, što omogućuje brza i efikasna dohvaćanja većih skupova podataka malim brojem operacija/upita.

Osnovna razlika između grafičkih i relacijskih tipova baza podataka je ta da su kod relacijskih baza poveznice među podacima nalaze u zapisu zbog čega se prilikom svakog upita kojim se traži neka međuovisnost mora točno znati koje se sve relacije spajaju, te kojim su atributima povezane. Upravo zbog toga, grafička baza podataka koristi se kod onih tipova podataka, gdje je poprilično komplicirano modelirati ih relacijskim putem, primjerice kod hijerarhijske organizacije i sl. Nadalje, prednost grafa je njegova jednostavna čitljivost gdje je primjerice, ako se radi o velikom skupu zapisa, lakše vizualizirati njihove odnose kao čvorove povezane bridovima, nego kao zapise u relacijskim tablicama. Mehanizam pohrane podataka kod grafičkih baza podataka razlikuje se ovisno o korištenoj tehnologiji. Neke se baziraju na relacijskom modelu te zapisuju podatke u tablice, dok se druge zasnivaju na NoSQL strukturama kao što su ključ-vrijednost (eng. *key-value*) ili zapisima u obliku dokumenata. Dok se kod relacijskih baza podataka SQL smatra univerzalnim upitnim jezikom, kod grafičkih to nije slučaj te su kod njih najrašireniji jezici Gremlin, SPARQL i Cypher.

Grafičke baze podataka zasnivaju se na sljedećim elementima: čvorovi (eng. *nodes*), veze ili bridovi (eng. *edges, relationships*) i svojstva (eng. *properties*). Čvorovi predstavljaju entitete, odnosno sve ono o čemu ima smisla voditi zapise. U usporedbi s relacijskom bazom, čvor je analogija relaciji, odnosno retku u tablici. Bridovi ili veze su linije koje označavaju

povezanost različitih entiteta i upravo su one najznačajniji dio grafičkih baza podataka, budući da predstavljaju apstrakciju koja nije implementirana niti u jednom drugom sustavu baza podataka [6]. Svojstva označavaju osnovne informacije koje pobliže opisuju entitete i veze, po čemu su određena ekvivalencija atributima kod relacijskih baza podataka. Odnosi tih elemenata prikazani su na Slici 5.



Slika 5 – Odnos elemenata grafičke baze podataka

Prema gornjem opisu strukture grafičke baze te prema osnovnim postavkama teorije grafova, mogu se izvući njezina svojstva poput dobre skalabilnosti kod povećanja broja zapisa, kao i brza prilagodljivost na promjene u organizaciji podataka, što nije slučaj kod relacijskog tipa gdje se relacijske sheme rijetko mijenjaju, a veliki broj zapisa donosi značajnije kašnjenje. Nadalje, kako su zapisi pohranjeni kao grafovi, korisnost takve strukture dolazi do izražaja ako se nad njima provode upiti kao što je računanje najkraćeg puta i sl [6].

3.1.1. Neo4j

Neo4j je sustav za upravljanje grafičkom bazom podataka razvijen od strane kompanije Neo Technology. Razlog zbog kojeg je odabran za potrebe ovog diplomskog rada leži u činjenici da je to najraširenija tehnologija za grafičke baze podataka, kao i u činjenici da zbog jednostavnog sučelja i dobre podrške pruža kvalitetnu okolinu za realizaciju heterogene okoline. Neo4j daje veliki naglasak na gore spomenute veze među podacima, budući da se

danas većina aplikacija i usluga zasniva velikim skupovima podataka (eng. *Big data*), ali i njihovim međuodnosima.

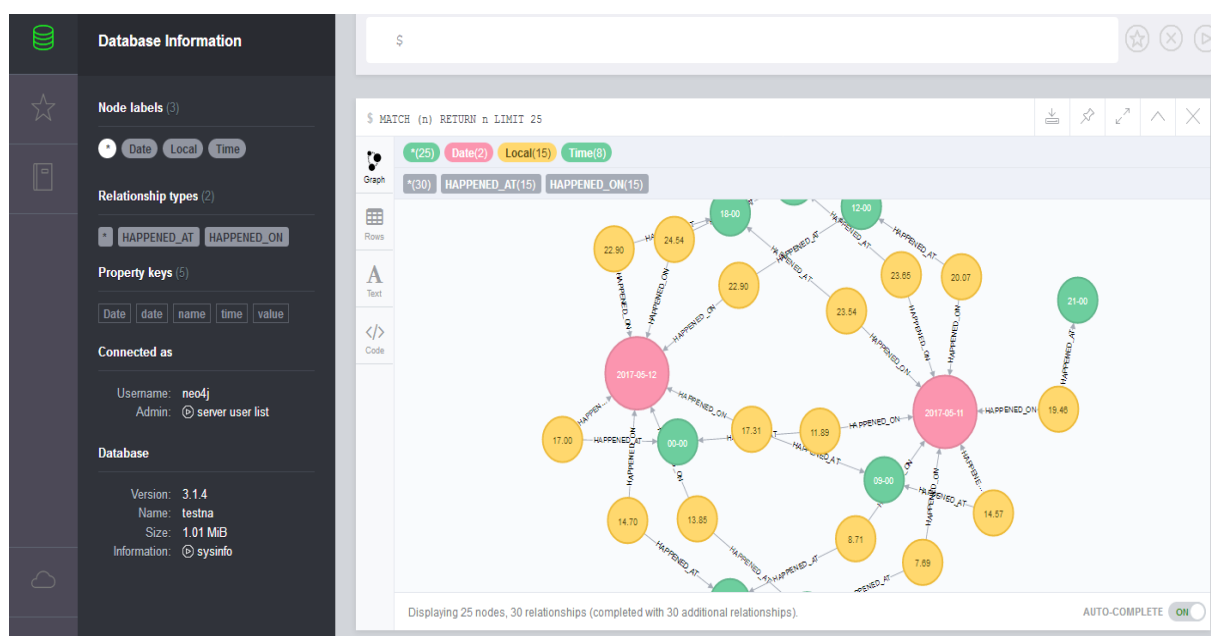
Osnovna svojstva Neo4j baze podataka jesu fleksibilnost sheme, tj. mogućnost brzih i efikasnih promjena podataka, njihovih svojstava i veza među njima, očuvanost ACID (eng. *Atomicity, Consistency, Isolation, Durability*) svojstava prilikom izvođenja transakcija te elastična skalabilnost [6]. Provođenje upita nad Neo4j bazom podataka vrši se putem Cypher upitnog jezika. Osim toga, postoji podrška za brojne programske jezike i radne okvire poput Jave, Pythona, C#, PHP-a itd. U Tablici 1 nalazi se usporedba Neo4j grafičke baze podataka u odnosu na relacijske i NoSQL baze podataka.

Tablica 1 – Usporedba Neo4j, relacijskih i NoSQL baza podataka [6]

	Neo4j	Relacijske baze podataka	Ostale NoSQL baze podataka
Pohrana podataka	<i>grafička struktura omogućuje brže transakcije i obradu povezanih podataka</i>	<i>fiksne tablice s unaprijed određenim atributima; povezani podaci često u različitim tablicama</i>	<i>nema podrške za povezane podatke na nivou baze podataka; opadanje performansi s porastom složenosti veza</i>
Modeliranje podataka	<i>fleksibilno, intuitivno, s velikim stupnjem kontrole</i>	<i>mora biti unaprijed definiran model podataka; nije sklon kasnijim promjenama</i>	<i>model podataka ne dopušta veliku kontrolu</i>
Performanse upita	<i>obrada grafičke strukture omogućuje veliku brzinu, tj. izvođenje u stvarnom vremenu</i>	<i>performanse značajnije opadaju povezivanjem većeg broja tablica</i>	<i>nema podrške za grafičku obradu podataka; sve veze rješavaju se u aplikacijskoj domeni</i>
Upitni jezik	<i>Cypher upitni jezik prilagođen izvođenju upita nad povezanim podacima</i>	<i>upitni jezici temeljeni na SQL-u koji povezuje tablice operacijom JOIN</i>	<i>različiti upitni jezici, bez podrške za izraze za veze među podacima</i>

Podrška transakcijama	<i>transakcije s konzistentnim podacima zbog očuvanosti ACID svojstva</i>	<i>potrebno ostvariti ACID svojstva radi očuvanja integriteta podataka</i>	<i>moгуća nekonzistentnost podataka</i>
Skalabilnost	<i>očuvanje integriteta podataka zbog replikacije</i>	<i>ostvariva putem replikacije, ali potencijalno skupa</i>	<i>dobra za unos podataka, slabija za čitanje; moguća nepouzdanost podataka</i>
Efikasnost podatkovnog centra	<i>zajedničko spremanje podataka i veza među njima omogućuje očuvanje performansi prilikom rasta složenosti</i>	<i>moгуćnost vertikalne ili horizontalne skalabilnosti, ali uz popriličan trošak</i>	<i>potreba za novim resursima što unosi potencijalne rizike</i>

Instalacija je poprilično jednostavna, nakon čega se može pokrenuti Neo4j poslužitelj na lokalnom poslužitelju putem web preglednika. Izgled sučelja nalazi se na Slici 6.

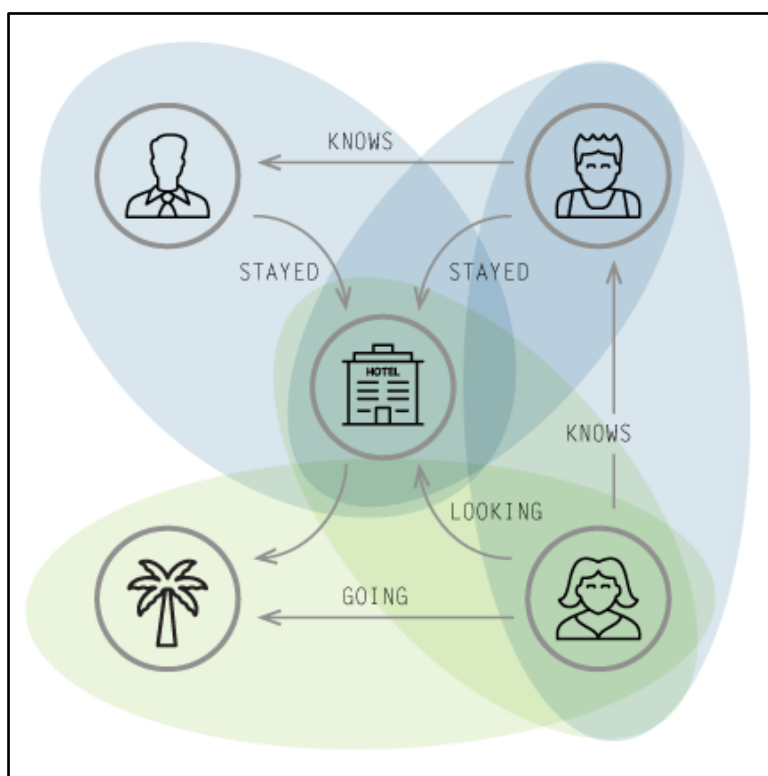


Slika 6 – Neo4j sučelje

Postoje brojna područja gdje Neo4j i grafičke baze općenito nalaze široku primjenu, a ovdje su nabrojana samo neka od njih [6]:

- Detekcija financijskih prevara;
- Pretraga temeljena na svojstvima grafa;
- Mrežne operacije;
- Mehanizmi za preporuke u stvarnom vremenu;
- Društvene mreže;
- Povezivanje podataka u poslovnim modelima;
- Upravljanje pristupom

Svaki od gore navedenih obrazaca upotrebe pridaje veliku važnost povezanosti entiteta i podataka, što je, kao što je gore već objašnjeno, najjače svojstvo grafičke baze podataka. Na Slici 7 nalazi se primjer vezan uz preporuke u stvarnom vremenu gdje su entiteti povezani tako da se u obzir uzimaju poznanstva, kao i povijest ponude i potražnje.



Slika 7 – Povezanost entiteta u mehanizmu preporuke u stvarnom vremenu (IZVOR: <https://neo4j.com/blog/enterprise-real-time-recommendation-engines/>)

3.1.2. Cypher upitni jezik

Cypher je jezik za provođenje upita nad grafičkom bazom podataka. Prvotno ga je razvila kompanija Neo Technology za Neo4j grafičke baze podataka, no s vremenom se otvorio i prema drugim kompanijama putem openCypher projekta [6]. Baziran je na modelu grafa sa svojstvima koji osnovnim elementima grafa (čvorovi, bridovi/veze) pridaje određena svojstva i oznake (eng. *labels*). Svaki čvor ili veza može imati 0 ili više svojstava i oznaka. Sintaksa Cypher-a zasniva se na ključnim riječima koje su nabrojane u nastavku:

- MATCH - dohvaća čvorove i/ili veze;
- WHERE - dodaje ograničenja na upite;
- RETURN - vraća rezultate upita;
- CREATE - stvara čvorove ili veze;
- DELETE - briše čvorove ili veze;
- SET - dodaje svojstva ili oznake;
- REMOVE - briše svojstva ili oznake;
- MERGE – kombinacija MATCH i CREATE

Na Slici 8 se nalazi primjer Cypher upita koji dohvaća one filmove iz baze podataka u kojima je glumio Tom Hanks. U ovom primjeru postoji veza između čvorova m i p koji predstavljaju film, odnosno glumaca koji glumi u nekom filmu.

```
MATCH (p:Person {name:"Tom Hanks"})  
      -[:ACTS_IN]->(m)  
RETURN m.title  
  
MATCH (p:Person {name: {name}})  
      -[:ACTS_IN]->(m)  
RETURN m.title
```

Slika 8 – Cypher upit

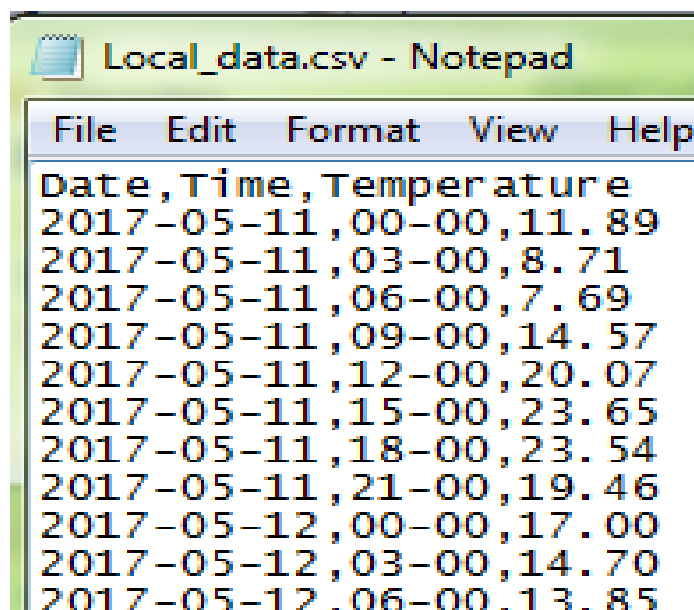
3.2. Opis izvora podataka

Za realizaciju heterogene okoline Interneta stvari putem Neo4j grafičke baze podataka korišteni su sljedeći podatkovni izvori:

- lokalni izvor s povijesnim podacima;
- web-izvor;
- emulirani senzorski čvorovi.

Lokalni izvor s povijesnim podacima vrijednosti temperature jest CSV (eng. *Comma-separated values*) datoteka koja je preuzeta sa sljedećeg web-izvora: <https://www.meteoblue.com/en/weather/archive/export/>

Za potrebe realizacije zadatka odabran je prikaz petnaestodnevnih očitavanja temperature za grad Zagreb s očitanjima vrijednosti svaka 3 sata. Kako bi se preuzete vrijednosti moglo uspješno unijeti u bazu podataka Neo4j, potrebno je prilagoditi datoteku. Izgled CSV datoteke nalazi se na Slici 9.



Date	Time	Temperature
2017-05-11	,00-00	,11.89
2017-05-11	,03-00	,8.71
2017-05-11	,06-00	,7.69
2017-05-11	,09-00	,14.57
2017-05-11	,12-00	,20.07
2017-05-11	,15-00	,23.65
2017-05-11	,18-00	,23.54
2017-05-11	,21-00	,19.46
2017-05-12	,00-00	,17.00
2017-05-12	,03-00	,14.70
2017-05-12	,06-00	,13.85

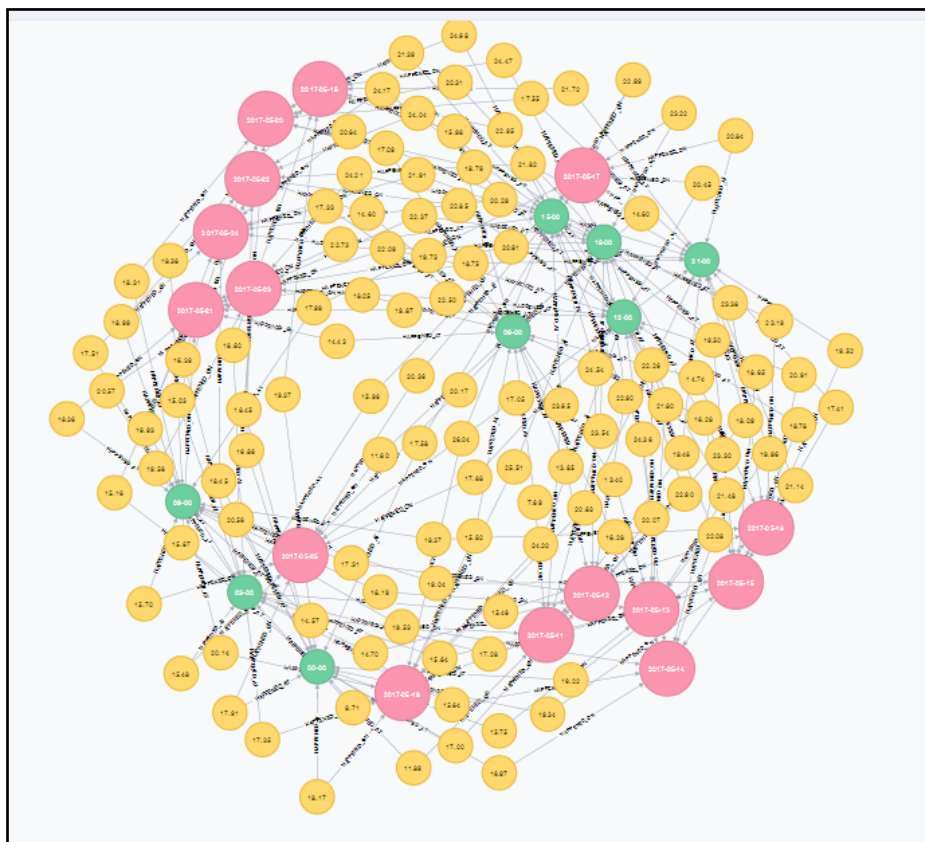
Slika 9 – Lokalna CSV datoteka s temperaturnim vrijednostima

Potom je tako oblikovanu datoteku moguće učitati u Neo4j te vrijednosti zapisati grafički. Način organizacije podataka napravljen je tako da vrijednosti temperature budu povezane s datumom kada su mjerene te vremenom očitavanja. Time je ostvaren minimalan broj čvorova s

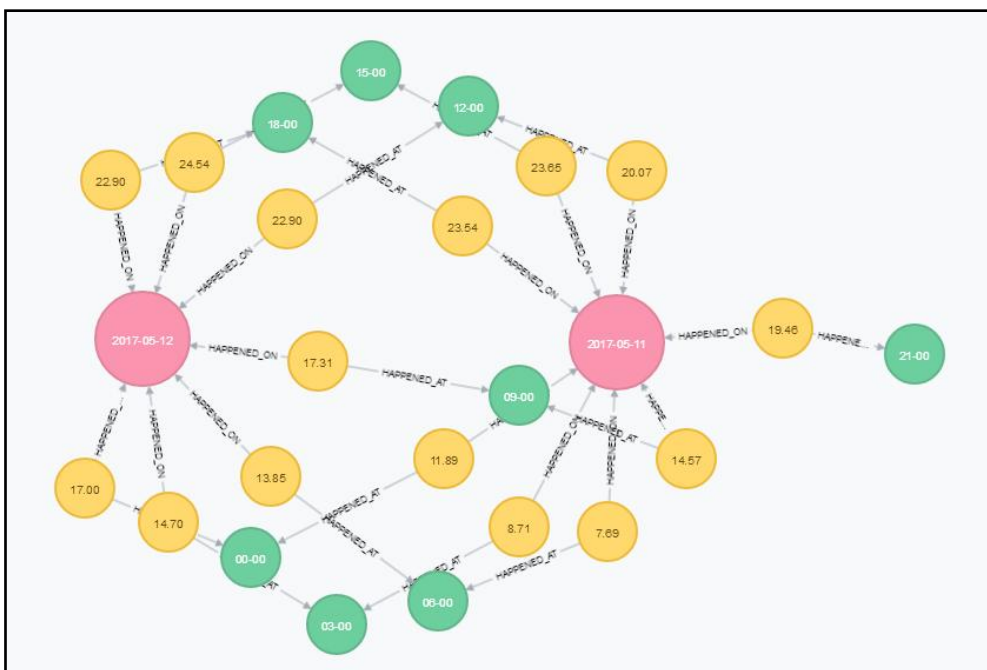
organizacijom koja ima smisla i na koju se mogu povezati očitavanja s drugih izvora, a koja su se vršila u to određeno vrijeme. Cypher upit kojim je to ostvareno nalazi se u nastavku:

```
LOAD CSV WITH HEADERS FROM "file:///C:/Local_data.csv" AS csvLine  
  
MERGE (date:Date { date: csvLine.Date })  
  
MERGE (time:Time { time: csvLine.Time })  
  
CREATE (local:Local {value: csvLine.Temperature})  
  
CREATE (local)-[:HAPPENED_ON]->(date)  
  
CREATE (local)-[:HAPPENED_AT]->(time)
```

Unosom tog upita stvara se zapis s očitanjima koji su u međudnosu s datumom i vremenom mjerenja. S obzirom na to da je u prvotnoj CSV datoteci bilo 120 zapisa, ukupno je stvoreno 143 čvorova (120 s vrijednosti, 15 s datumima i 8 s vremenima), te 240 veza, po 2 za svaki čvor s vrijednošću temperature. Grafički prikaz nalazi se na Slici 10. Budući da je prikaz pomalo kaotičan i nepregledan, u Neo4j pregledniku prvotni prikaz ograničen je na 25 čvorova, što se može vidjeti na Slici 11.



Slika 10 – Grafički prikaz podataka iz lokalne datoteke



Slika 11 – Ograničeni prikaz podataka iz lokalne datoteke

Web-izvor podataka jest skup podataka u JSON formatu preuzet sa sljedećeg URL-a:

<https://developer.worldweatheronline.com>

Na toj web-stranici moguće je odabrati različite parametre vezane uz vrijeme na različitim lokacijama. Zbog relevantnosti podataka odabran je isti skup podataka kao za lokalni izvor podataka, tj. za isto razdoblje (15 dana) uzimaju se očitavanja temperature za grad Zagreb svaka 3 sata. Izgled stranice s JSON podacima nalazi se na Slici 12.

```

data:
  request:
    0:
      type: "City"
      query: "Zagreb, Croatia"
  weather:
    0:
      date: "2017-05-11"
      astronomy:
        0:
          sunrise: "05:30 AM"
          sunset: "08:16 PM"
          moonrise: "08:43 PM"
          moonset: "06:10 AM"
      maxtempC: "23"
      maxtempF: "73"
      mintempC: "14"
      mintempF: "57"
      totalSnow_cm: "0.0"
      sunHour: "69.5"
      uvIndex: "0"
      hourly:
        0:
          time: "0"
          tempC: "7"

```

Slika 12 – JSON podaci iz web-izvora

S obzirom na to da gore navedeni skup podataka u JSON formatu nije prilagođen za direktan unos podataka u Neo4j grafičku bazu, bilo je potrebno iz odabranog skupa prvo odabrati željene parametre (datumi i vremena očitavanja te izmjerene vrijednosti temperature), a potom ih zapisati i povezati u bazi podataka. To je ostvareno putem skripte koja je napisana u Python programskom jeziku. Kako bi se u Pythonu mogle koristiti sve mogućnosti koje nudi Neo4j potrebno je instalirati knjižnicu Py2neo. Skripta se sastoji od 2 dijela. U prvom dijelu se povezuje s jedne strane na URL grafičke baze podataka, a s druge na URL gdje se nalaze podaci, dok se u drugom dijelu provodi Cypher upit koji prvo odabire one vrijednosti koje mu trebaju, a potom kreira čvorove i povezuje ih na prikladan način. Kod skripte nalazi se u nastavku.

```
import os
import requests
from py2neo import Graph
from py2neo import authenticate

# set up authentication parameters
authenticate("localhost:7474", "neo4j", "388878")

# Connect to graph and add constraints.
neo4jUrl = os.environ.get('NEO4J_URL', "http://localhost:7474/db/data/")
graph = Graph(neo4jUrl, secure=False)

# Build URL.
apiUrl = "http://api.worldweatheronline.com/premium/v1/past-
weather.ashx?key=e26e98dc469f4c6490d104509172505&q=Zagreb&format=json&date=2
017-05-11&enddate=2017-05-25&tp=3"

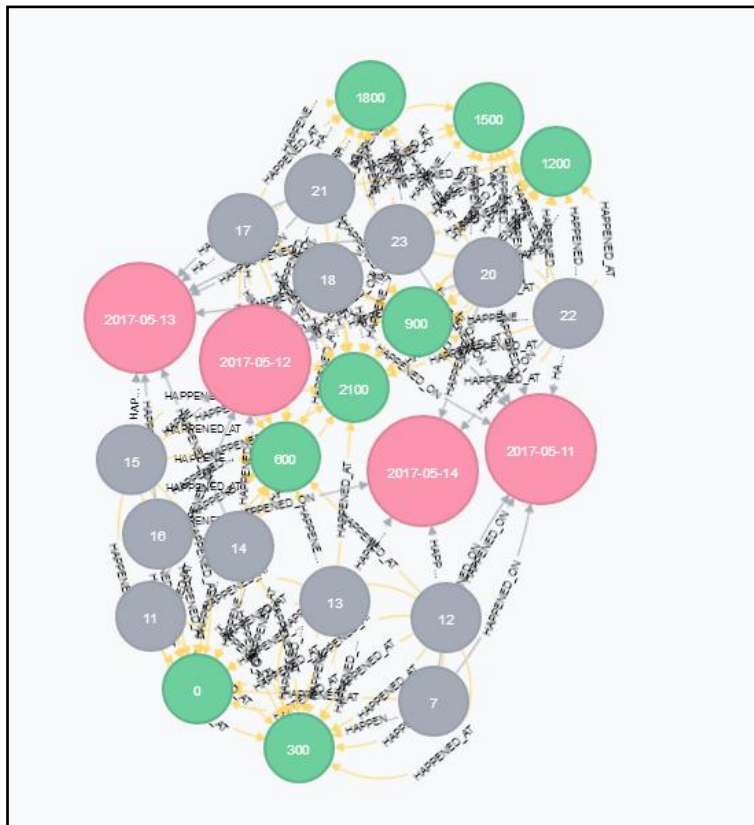
# Send GET request.
json = requests.get(apiUrl, headers = {"accept": "application/json"}).json()

# Build query.
query = """
WITH {json} as data
UNWIND data.data.weather as q
FOREACH (a IN q | MERGE (date:Date { date: q.date })
FOREACH (b IN q.hourly | MERGE (time:Time {time:b.time}))
MERGE (web:Web {value: b.tempC})
CREATE (web)-[:HAPPENED_ON]->(date)
```

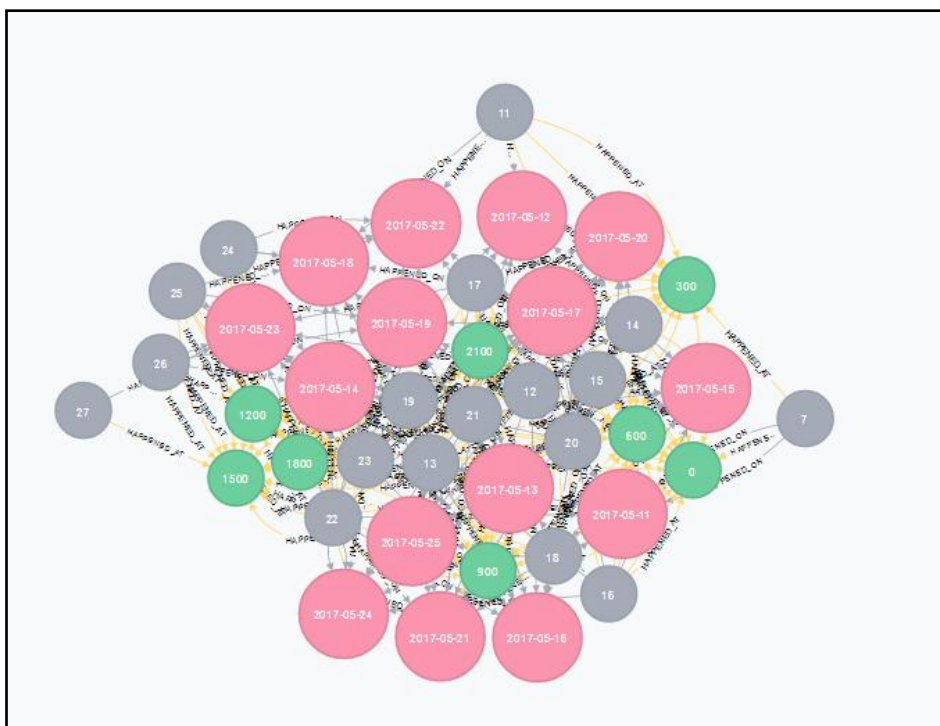
```
CREATE (web)-[:HAPPENED_AT]->(time)
))
.....
```

```
results = graph.run(query,json=json)
```

Prikaz tako spremljenih podataka nalazi se na Slikama 13 i 14. Kao i kod lokalnog izvora podataka, temperaturene vrijednosti povezane su s datumom i vremenom očitavanja. Na Slici 13 nalazi se prikaz ograničenog broja čvorova, a na Slici 14 svi kreirani čvorovi i veze. Čvorovi koji sadrže temperaturene vrijednosti obojeni su sivom bojom, vremenski čvorovi su zelene boje, a čvorovi koji sadrže u sebi datumski zapis su roze boje.

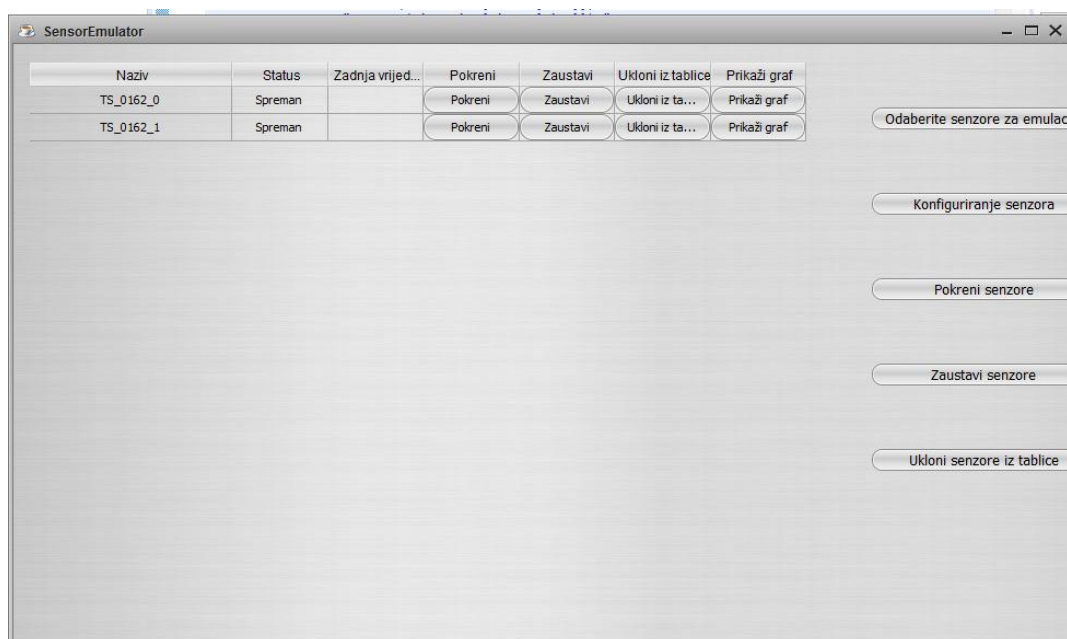


Slika 13 – Ograničeni prikaz podataka iz web-izvora



Slika 14 – Podaci iz web-izvora

Za izvor podataka iz emuliranih senzorskih čvorova iskorištena je računalna aplikacija izrađena u okviru diplomskog rada [10], u programskom jeziku Java. Ta aplikacija služi kako bi se mogli odabrati određeni tipovi senzora koji se potom kreiraju, a zatim generiraju određene vrijednosti. Izgled aplikacije nalazi se na Slici 15.



Slika 15 – Računalna aplikacija za emuliranje senzorskih čvorova

Kako su i kod lokalnog i kod web-izvora korištene temperaturne vrijednosti, tako je i za emulirani izvor podataka funkcija kreirana za upis vrijednosti generiranih od strane senzora za temperaturu. S obzirom na to da je spremanje tih podataka zamišljeno za bazu podataka koja se nalazi u oblaku, potrebno je promijeniti, odnosno dodati funkciju u postojeći kod koja prilikom generiranja podataka otvara konekciju prema Neo4j grafičkoj bazi, a potom putem Cypher upitnog jezika upisuje vrijednosti u obliku čvorova. Osim čvorova s vrijednostima, u funkciji se generiraju i čvorovi za datum, odnosno vrijeme generiranja koji se potom povezuju s vrijednostima kojima pripadaju. Kod funkcije nalazi se u nastavku.

```
public static boolean insertTempValueNeo(String value) {

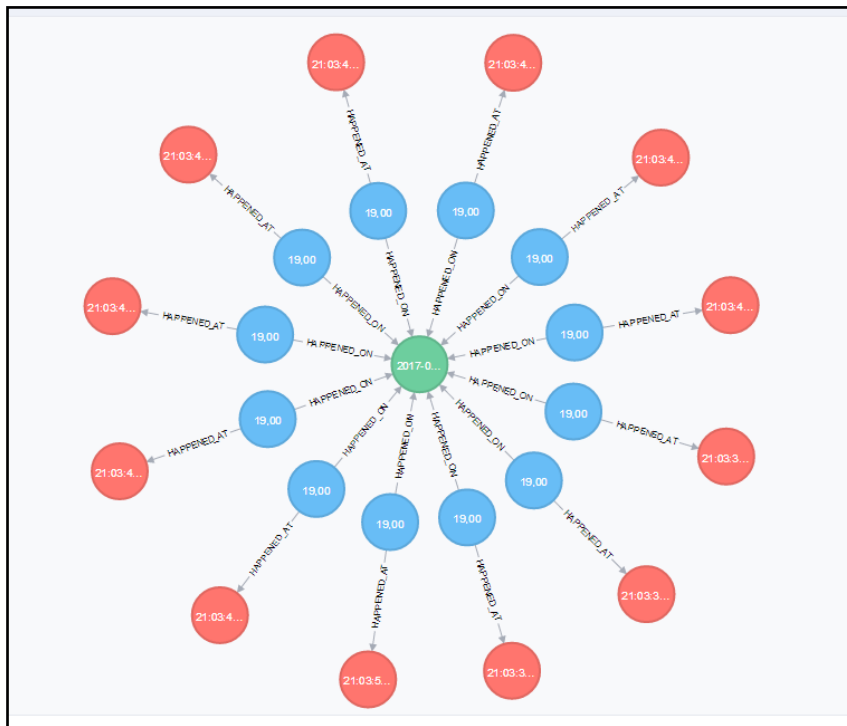
    Driver driver = GraphDatabase.driver( "bolt://localhost:7687",
AuthTokens.basic( "neo4j", "388878" ) );

    Session session = driver.session();
    LocalDateTime timePoint = LocalDateTime.now();
    String timestamp = timePoint.toString();
    String[] parts = timestamp.split("T");
    String date = parts[0];
    String time = parts[1];

    session.run( "CREATE (emulated:Emulated {value: {value}}) "
        + "MERGE (date:Date {date: {date}}) "
        + "CREATE (time:Time{time: {time}}) "
        + "CREATE (emulated)-[:HAPPENED_ON]->(date) "
        + "CREATE (emulated)-[:HAPPENED_AT]->(time)",
        parameters( "value", value, "date", date, "time", time ) );

    session.close();
    driver.close();
    return true;
}
```

Prilikom generiranja vrijednosti temperature dobiveni podaci zapisuju se u Neo4j grafičku bazu podataka. Izgled tih podataka nalazi se na Slici 16. Zelenom bojom obojen je čvor s datumom, crvene boje su vremenske oznake, a plavi čvorovi predstavljaju temperaturne vrijednosti. Vidljivo je da je svaka temperatura povezana sa svojim vremenom generiranja, a da su sve vrijednosti generirane istog datuma.



Slika 16 – Ograničeni prikaz podataka emuliranih senzora

3.3. Povezivanje izvora podataka

Gore navedeni izvori (lokalna datoteka, web-izvor) povezani su u smislenu cjelinu. Vrijednosti koje generiraju emulirani senzorski čvorovi ovdje nisu uzimane u obzir, budući da nemaju nikakvu korelaciju s ostalim izvorima podataka. Kako se radi o podacima vrijednosti temperature na području grada Zagreba, a koje su očitavane svaka tri sata u istom razdoblju (15 dana) kod oba izvora, struktura grafičke baze podataka zamišljena je tako da se na čvorove datuma i vremena očitavanja povezuju čvorovi koji sadrže vrijednosti temperatura. Čvorovi nose različite oznake: Local koji označava vrijednosti iz lokalne CSV datoteke te Web koji se odnosi na vrijednosti iz web-izvora u JSON formatu. Skriptu navedenu u prethodnom poglavlju bilo je potrebno djelomično modificirati kako bi se izvori mogli

povezati, te dodati Cypher upit objašnjen također u prethodnom poglavlju koji uzima vrijednosti iz lokalne datoteke. Python skripta nalazi se u nastavku.

```
import os
import requests
from py2neo import Graph
from py2neo import authenticate

# set up authentication parameters
authenticate("localhost:7474", "neo4j", "388878")

# Connect to graph and add constraints.
neo4jUrl = os.environ.get('NEO4J_URL', "http://localhost:7474/db/data/")
graph = Graph(neo4jUrl, secure=False)

# Build URL.
apiUrl = "http://api.worldweatheronline.com/premium/v1/past-
weather.ashx?key=e26e98dc469f4c6490d104509172505&q=London&format=json&date=2
017-05-11&enddate=2017-05-12&tp=3"

# Send GET request.
json = requests.get(apiUrl, headers = {"accept": "application/json"}).json()

# Build queries.
query1 = """
LOAD CSV WITH HEADERS FROM "file:///C:/Local_data.csv" AS csvLine
MERGE (date:Date { date: csvLine.Date })
MERGE (time:Time { time: csvLine.Time })
MERGE (local:Local {value: csvLine.Temperature})
MERGE (local)-[:HAPPENED_ON]->(date)
MERGE (local)-[:HAPPENED_AT]->(time)
"""

results1 = graph.run(query1)

query2 = """
WITH {json} as data
UNWIND data.data.weather as q
FOREACH (a IN q | MERGE (date:Date { date: q.date })
FOREACH (b IN q.hourly | MERGE (time:Time {time:b.time}))
MERGE (web:Web {value: b.tempC})
```

```

MERGE (web)-[:HAPPENED_ON]->(date)
MERGE (web)-[:HAPPENED_AT]->(time)
))

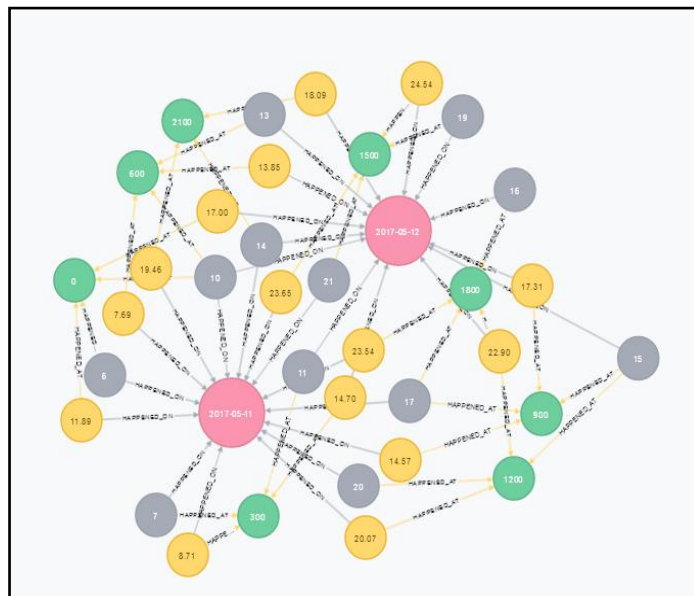
```

```

results2 = graph.run(query2,json=json)

```

Zbog jednostavnosti prikaza, kao podatkovni skup koji se spremio u bazu podataka uzete su vrijednosti za samo 2 dana. Prikaz te grafičke strukture nalazi se na Slici 17. Čvorovi s vrijednostima iz lokalne datoteke obojeni su žutom bojom, sivom bojom obojeni su čvorovi iz web-izvora, datumski čvorovi su roze boje, a vremenski zelene.



Slika 17 – Prikaz podataka heterogene okoline

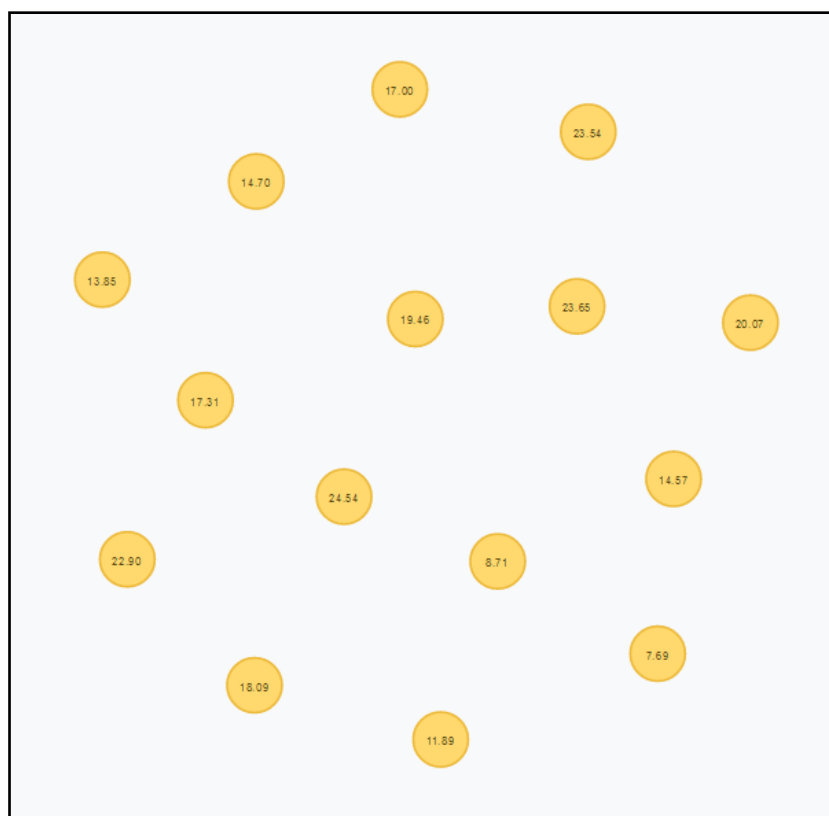
3.3.1. Primjeri upita nad Neo4j bazom podataka

Nad gore realiziranom bazom podataka moguće je na jednostavan način izvoditi upite putem Neo4j grafičkog sučelja. U nastavku su dani primjeri nekih upita.

PRIMJER 1: prikaz svih čvorova s određenom oznakom (oznaka *Local*)

UPIT: ***MATCH (n:Local) RETURN n***

ODGOVOR: Slika 18 (grafički prikaz) Slika 19 (tablični prikaz)



Slika 18 – Grafički prikaz odgovora na upit

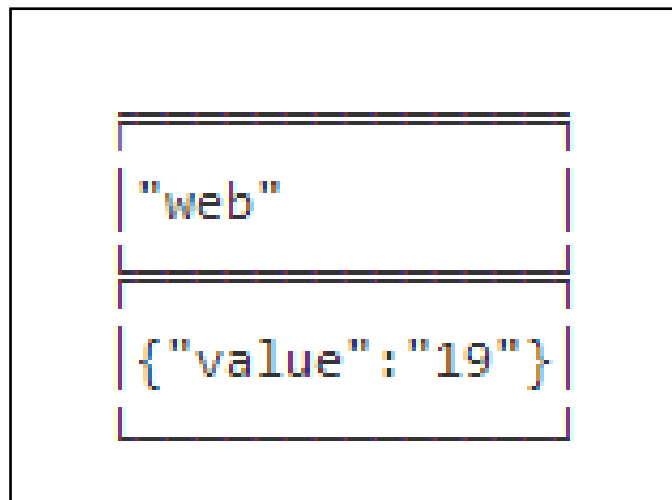
value	8.71
value	7.69
value	14.57
value	20.07
value	23.65
value	23.54

Slika 19 – Tablični prikaz odgovora na upit

PRIMJER 2: prikaz čvora s određenom vrijednošću

UPIT: *MATCH (web:Web {value:"19"}) RETURN web*

ODGOVOR: Slika 20

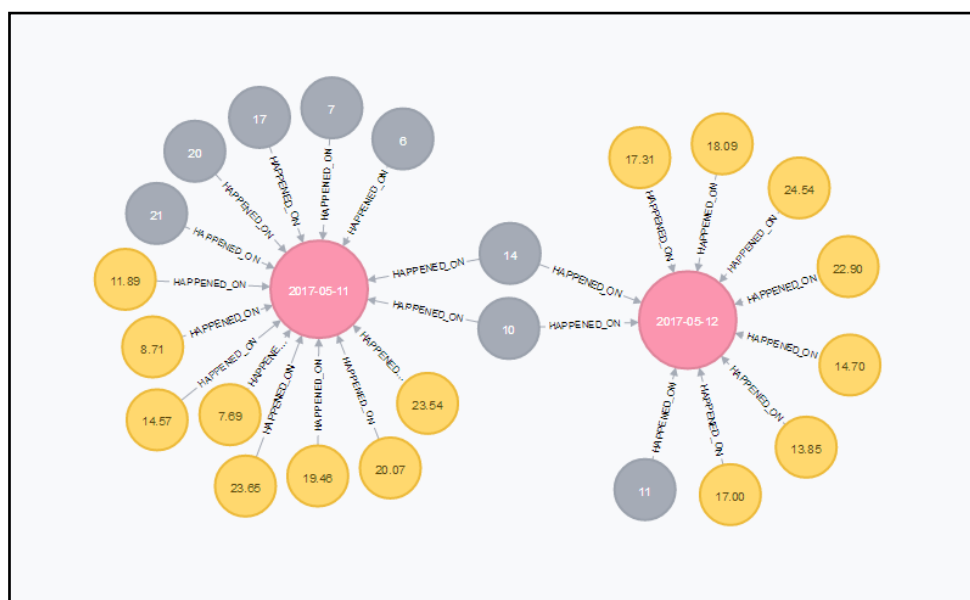


Slika 20 – Odgovor na upit iz Primjera 2

PRIMJER 3: prikaz početnih i krajnjih čvorova veza određene oznake

UPIT: *MATCH p=()-[r:HAPPENED_ON]->() RETURN p*

ODGOVOR: Slika 21



Slika 21 – Odgovor na upit iz Primjera 3

PRIMJER 4: prikaz broja čvorova za svaku od oznaka

UPIT: **MATCH** (n) **RETURN** *distinct labels(n), count(*)*

ODGOVOR: Slika 22

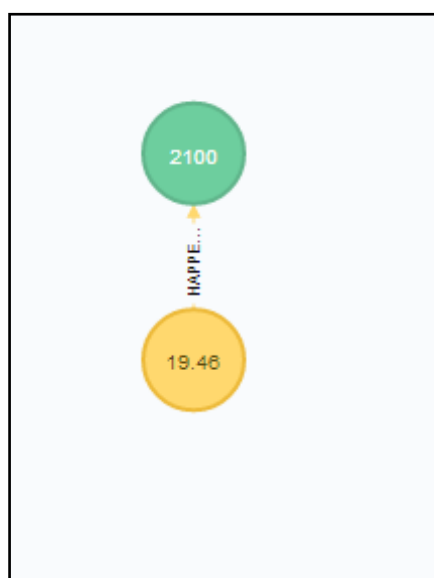
"labels(n)"	"count(*)"
["Date"]	"2"
["Local"]	"15"
["Web"]	"12"
["Time"]	"8"

Slika 22 – Odgovor na upit iz Primjera 4

PRIMJER 5: prikaz određene veze s početnim i završnim čvorom

UPIT: **MATCH** (a)-[r]-(b) **WHERE** ID(r)=76 **RETURN** r, a, b

ODGOVOR: Slika 23



Slika 23 – Odgovor na upit iz Primjera 5

4. Usluge u Internetu stvari

Cjelokupni smisao koncepta Interneta stvari je upravo u tome da krajnji korisnici mogu iskoristiti svoje uređaje i pristup mreži kako bi mogli promatrati okolinu i upravljati istom. Prethodno navedena područja primjene su samo dio onoga što se nudi na tržištu, a jasno je kako će se u budućnosti nadograđivati postojeće i smišljati potpuno nove usluge, ovisno o potrebi i trendovima. Zbog gore navedene heterogenosti IoT platforme, odnosno različitih uređaja na kojima se pojedine aplikacije koriste, osnovna ideja prilikom kreiranja svake usluge jest neovisnost o tehnologiji na kojoj se izvodi. Nadalje, usluga mora skrivati sve one informacije koje bi, ako bi se prikazivale krajnjem korisniku, činile tu uslugu neintuitivnom, neefikasnom i neprivlačnom za korištenje. S obzirom na to da je trend uređaja koji su povezani na Internet i ljudi koji te uređaje koriste u stalnom i velikom porastu, druga važna osobina novokreirane usluge je ta da se otpočetka razmišlja može li se njezina implementacija ponovno iskoristiti, i to višestruko, kao i može li se ta usluga uklopiti kao dio neke veće cjeline. Za tako nešto potrebno je ostvariti efikasnu logiku pohrane i pristupa podacima. U kreiranju i realizaciji pojedinih usluga Interneta stvari postoje neki ustaljeni arhitekturni principi od kojih su oni najčešće korišteni opisani u sljedećim potpoglavljima.

4.1. Uslužno orijentirana arhitektura

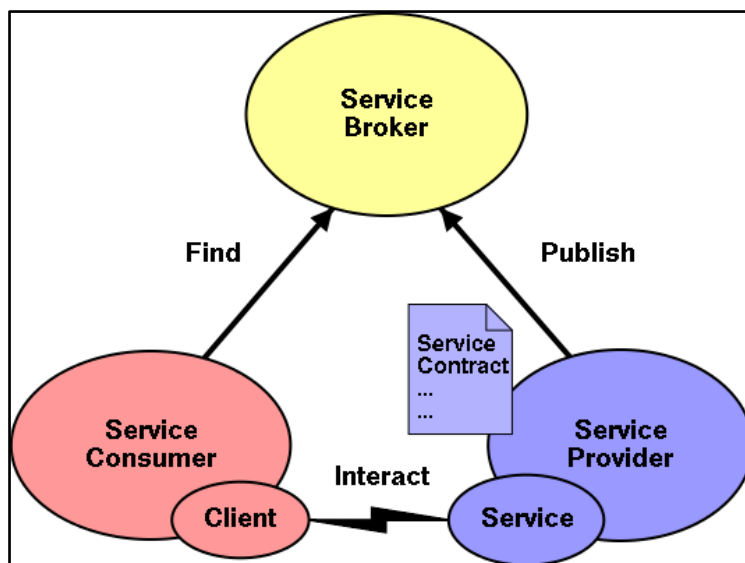
Uslužno ili servisno orijentirana arhitektura (eng. *Service-oriented architecture*, SOA) je način izrade usluga čije je osnovno svojstvo raspodijeljenost, tj. odvojenost pojedinih usluga koje se zbog toga mogu razvijati i nadograđivati neovisno jedna o drugoj [5]. Također, usluge kreirane po SOA-inom principu moraju biti neovisne o uređajima, proizvođačima i tehnologijama na kojima se koriste. Ako jedna usluga za ostvarenje svojih potreba mora koristiti drugu, tada ona mora poznavati njezinu lokaciju i opis te mora biti dogovoren izgled komunikacije među njima, odnosno format poruke i protokol po kojem se one šalju. Premda SOA nije nužno vezana uz web-usluge, najlakše ju je upravo izgraditi tim putem. Njezina osnovna svojstva su slaba povezanost usluga, uslužni ugovori kao prije spomenuti dogovori o komunikaciji, na što se dalje logično vežu autonomnost, odnosno da usluga ima kontrolu nad logikom koju enkapsulira, te apstrakcija kao svojstvo koje ukazuje na to da se izvan usluge vidi samo ono što je u uslužnom ugovoru [8]. Nadalje, sve su usluge ponovno iskoristive,

moгу se kombinirati u složenije usluge te su najčešće skalabilne što znači da se dobro nose s povećanjem broja zahtjeva/uređaja u smislu očuvanja zadovoljavajućih performansi [5].

Prema jednoj od definicija SOA-e, usluga ima 4 svojstva:

- dobro je definirana te je poznat njezin ishod,
- samoodrživa je,
- krajnji korisnici je vide kao cjelinu, tj. nepoznati su im detalji implementacije (eng. *black box*),
- može biti građena od manjih usluga.

Korištenje SOA-inih usluga ostvareno je komunikacijom triju entiteta: pružatelj usluge (eng. *Service provider*), korisnik usluge (eng. *Service consumer*) i ponuditelj usluge (eng. *Service broker*) [7]. Pružatelj usluge nudi određenu uslugu te daje informacije ponuditelju. Nadalje, on određuje pod koje područje spada usluga, kolika je njezina cijena te koji su prioriteti u njezinom funkcioniranju. Odnos između pružatelja i ponuditelja uređen je gore spomenutim uslužnim ugovorom koji se sastoji od poslovnog, funkcionalnog i tehničkog dijela. Ponuditelj usluge sadrži sve informacije o pojedinoj usluzi koje mu je pružatelj za nju dao, a po kojima ih onda korisnik može pronaći. Postoje javni i privatni ponuditelji, ovisno nudi li se pojedina usluga svima ili samo ograničenom broju korisnika. Korisnik usluge pronalazi njezinu specifikaciju kod ponuditelja, a potom se povezuje s pružateljem kako bi počeo s korištenjem. Odnos tih entiteta nalazi se na Slici 24.



Slika 24 – Odnos entiteta u SOA-i (IZVOR: <https://www.w3.org>)

4.2. REST

REST (eng. *Representational State Transfer*) je arhitekturni princip razvoja jezično neovisnih usluga koje su dostupne klijentima putem razmjene poruka. Čini ga skup ograničenja pomoću kojih se kreiraju komponente za skalabilne web servise i distribuirane sustave. Takvim načinom kreiranja komponenti ostvaruje se arhitektura koja je jednostavno održiva i postiže dobre performanse. REST je vrlo prihvaćen kao jednostavnija inačica u odnosu na SOAP (eng. *Service Oriented Architecture Protocol*) bazirane Web servise. Razlike između REST-a i SOAP-a prikazane su u Tablici 2. REST sustavi najčešće komuniciraju preko protokola HTTP (eng. *Hypertext Transfer Protocol*) koristeći iste metode kao što se koriste za komunikaciju na Internetu.

Neka od svojstava ovakvog tipa arhitekture su dobra skalabilnost kojom se omogućuje broj komponenti i interakcija među njima, jednostavnost sučelja putem kojih komponente komuniciraju, mogućnost brze izmjene komponenti, kao i njihova prenosivost te pouzdanost koja osigurava da se kvarom na nekoj komponenti, poveznici ili pogreškom u podacima koji se šalju ne sruši cijeli sustav.

Tablica 2 – Razlike između REST-a i SOAP-a

REST	SOAP
<i>arhitekturni princip</i>	<i>protokol</i>
<i>koristi samo protokol HTTP</i>	<i>protokol HTTP (ili FTP/SMTP) za prijenos podataka omotan u SOAP</i>
<i>podrška za brojne formate: JSON, XML, YAML...</i>	<i>koristi jedino XML format</i>
<i>dobre performanse i skalabilnost, mogućnost priručne memorije</i>	<i>slabije performanse i složena skalabilnost, nemogućnost priručne memorije</i>
<i>raširen, često se koristi</i>	<i>koristi se tamo gdje REST nije moguć</i>

Navedena svojstva se osiguravaju primjenom specifičnih ograničenja na komponentama, poveznicama i podacima. Među ta ograničenja spadaju odvojenost klijenta i poslužitelja koja se postiže sučeljem među njima. Tako se klijenti ne moraju brinuti o pohrani podataka koja se realizira na poslužitelju, pa se time povećava njihova prenosivost. S druge strane, poslužitelji

ne utječu na klijentsko sučelje i stanja u kojima klijent može biti što uvjetuje njihovu jednostavniju izvedbu i bolju skalabilnost. I klijent i poslužitelj mogu se razvijati neovisno jedan o drugome, sve dok se ne mijenja sučelje među njima. Nadalje, kod klijentsko-poslužiteljske komunikacije ne dolazi do spremanja klijentskoga konteksta na poslužitelju između pojedinih zahtjeva, već svaki zahtjev sadrži u sebi sve potrebne informacije da bi se mogao obraditi. Moguće je povećati performanse sustava korištenjem priručne memorije (eng. *cache*) za neke poruke tako što se eliminira određena količina interakcija između klijenata i poslužitelja. Drugi način je korištenje posredničkih poslužitelja. Svojstvo kojim se proširuje funkcionalnost klijenata naziva se kod na zahtjev (eng. *code on demand*) i funkcionira tako da poslužitelj pošalje klijentu dio izvršnog koda. To je ograničenje za razliku od ostalih opcionalno te se implementira ovisno o zahtjevima sustava. Osnova za oblikovanje svakog REST sustava je uniformno sučelje sa svojim ograničenjima. Ako je u sustavu prekršeno bilo koje od gore navedenih ograničenja, on se ne može smatrati baziranim na REST arhitekturi, premda može imati većinu njenih obilježja.

REST koji se bazira na razmjeni HTTP poruka definiran je sljedećim parametrima:

- URI-jem (eng. *Uniform Resource Identifier*), tj. nazivom za potrebni resurs,
- formatom podataka koji se šalju,
- vrstom HTTP metode i
- hipertekst vezom prema potrebnim resursima.

HTTP metode koje se najčešće koriste u REST-u su: GET, POST, PUT i DELETE.

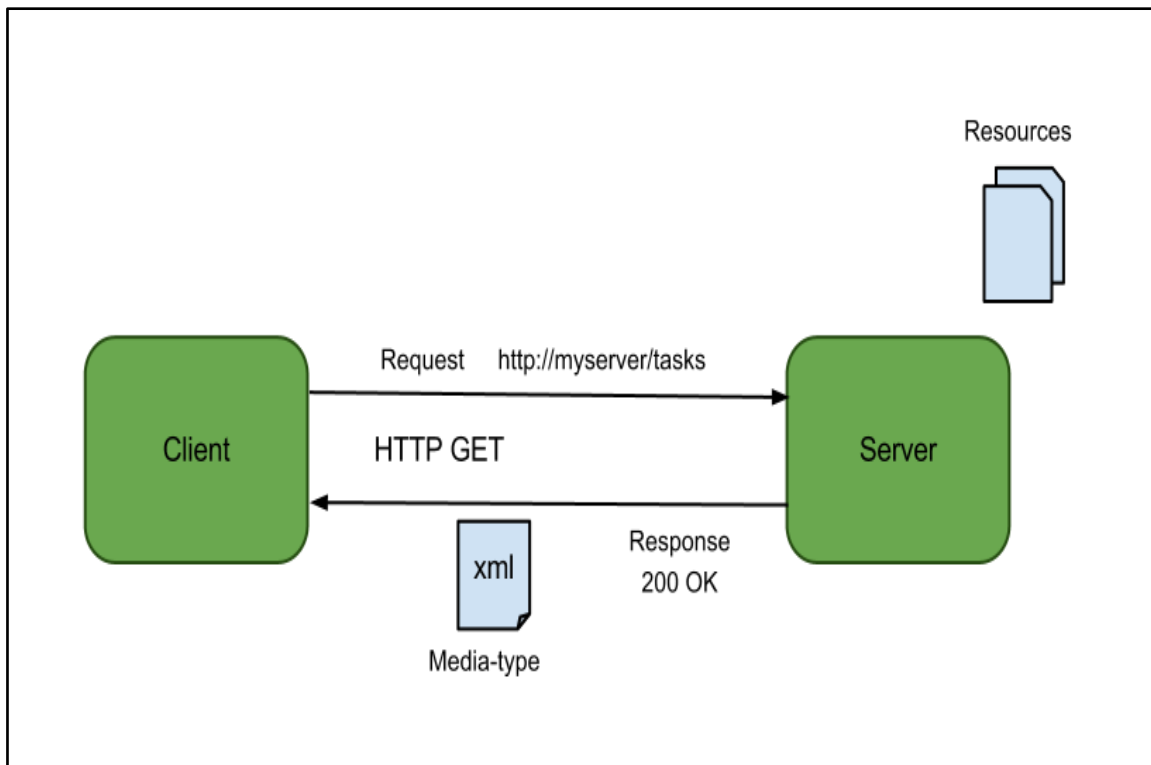
GET metoda dohvaća i ispisuje sve ili određenog člana unutar neke kolekcije.

POST metoda stavlja novi zapis u kolekciju.

PUT metoda mijenja jednog člana unutar kolekcije ili općenito cijelu kolekciju.

DELETE metoda briše člana kolekcije ili cijelu kolekciju.

Primjer razmjene poruka između klijenta i poslužitelja dan je na Slici 25.



Slika 25 - Razmjena poruka putem REST-a

4.2.1. Primjeri REST zahtjeva nad Neo4j bazom podataka

Nad gore realiziranom bazom podataka moguće je na jednostavan način dohvaćati podatke u JSON formatu putem REST zahtjeva. Kao REST klijent koji upućuje zahtjeve prema Neo4j bazi podataka korišten je REST Client dodatak za web-preglednik Mozilla Firefox. U nastavku su dani primjeri zahtjeva.

PRIMJER 1: dohvaćanje jednog čvora po ID-u

REST ZAHTJEV

Metoda: GET

URL: `http://localhost:7474/db/data/node/61`

Zaglavlje: `Accept: application/json; charset=UTF-8`

Tijelo poruke: /

ODGOVOR: Slika 26

```
{
  "extensions" : { },
  "metadata" : {
    "id" : 61,
    "labels" : [ "Local" ]
  },
  "paged_traverse" : "http://localhost:7474/db/data/node/61/paged/traverse/{returnType}?pageSize,leaseTime",
  "outgoing_relationships" : "http://localhost:7474/db/data/node/61/relationships/out",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/61/relationships/out/{-list|&|types}",
  "create_relationship" : "http://localhost:7474/db/data/node/61/relationships",
  "labels" : "http://localhost:7474/db/data/node/61/labels",
  "traverse" : "http://localhost:7474/db/data/node/61/traverse/{returnType}",
  "all_relationships" : "http://localhost:7474/db/data/node/61/relationships/all",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/61/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/61/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/61",
  "incoming_relationships" : "http://localhost:7474/db/data/node/61/relationships/in",
  "properties" : "http://localhost:7474/db/data/node/61/properties",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/61/relationships/in/{-list|&|types}",
  "data" : {
    "value" : "24.54"
  }
}
```

Slika 26 – Odgovor na REST zahtjev iz Primjera 1

PRIMJER 2: dohvaćanje svih oznaka u bazi

REST ZAHTJEV

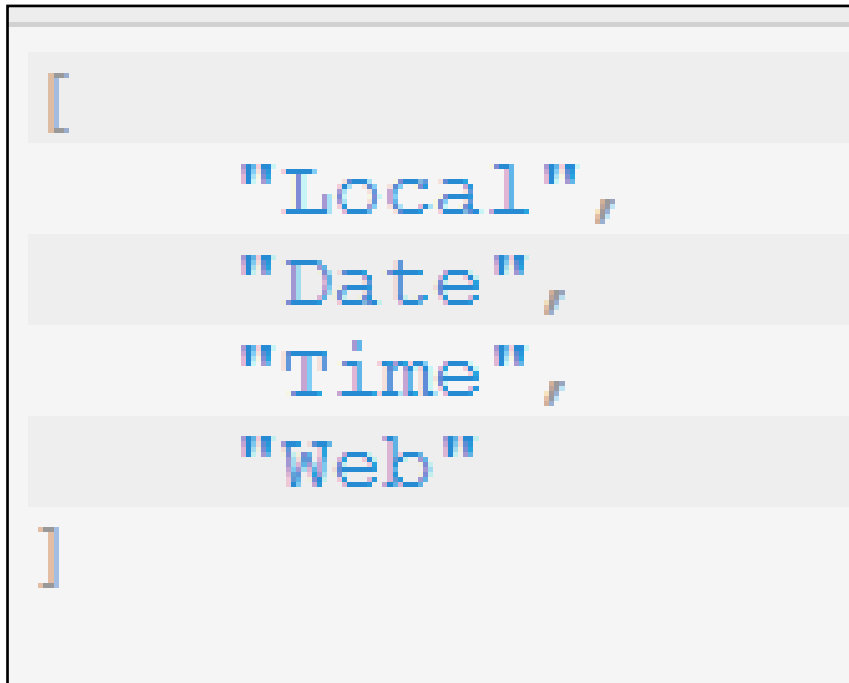
Metoda: GET

URL: `http://localhost:7474/db/data/labels`

Zaglavlje: Accept: application/json; charset=UTF-8

Tijelo poruke: /

ODGOVOR: Slika 27



```
[  
  "Local",  
  "Date",  
  "Time",  
  "Web"  
]
```

Slika 27 – Odgovor na REST zahtjev iz Primjera 2

PRIMJER 3: dohvaćanje svih veza za neki čvor

REST ZAHITJEV

Metoda: GET

URL: <http://localhost:7474/db/data/node/61/relationships/all>

Zaglavlje: Accept: application/json; charset=UTF-8

Tijelo poruke: /

ODGOVOR: Slika 28

```
[ {
  "extensions" : { },
  "metadata" : {
    "id" : 88,
    "type" : "HAPPENED_AT"
  },
  "data" : { },
  "start" : "http://localhost:7474/db/data/node/61",
  "property" : "http://localhost:7474/db/data/relationship/88/properties/{key}",
  "self" : "http://localhost:7474/db/data/relationship/88",
  "end" : "http://localhost:7474/db/data/node/49",
  "type" : "HAPPENED_AT",
  "properties" : "http://localhost:7474/db/data/relationship/88/properties"
}, {
  "extensions" : { },
  "metadata" : {
    "id" : 87,
    "type" : "HAPPENED_ON"
  },
  "data" : { },
  "start" : "http://localhost:7474/db/data/node/61",
  "property" : "http://localhost:7474/db/data/relationship/87/properties/{key}",
  "self" : "http://localhost:7474/db/data/relationship/87",
  "end" : "http://localhost:7474/db/data/node/55",
  "type" : "HAPPENED_ON",
  "properties" : "http://localhost:7474/db/data/relationship/87/properties"
}
```

Slika 28 – Odgovor na REST zahtjev iz Primjera 3

PRIMJER 4: dohvaćanje svih čvorova za neku oznaku

REST ZAHITJEV

Metoda: GET

URL: <http://localhost:7474/db/data/label/Date/nodes>

Zaglavlje: Accept: application/json; charset=UTF-8

Tijelo poruke: /

ODGOVOR: Slika 29

```
[ {
  "metadata" : {
    "id" : 38,
    "labels" : [ "Date" ]
  },
  "data" : {
    "date" : "2017-05-11"
  },
  "paged_traverse" : "http://localhost:7474/db/data/node/38/paged/traverse/{returnType}?pageSize,leaseTime",
  "outgoing_relationships" : "http://localhost:7474/db/data/node/38/relationships/out",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/38/relationships/out/{-list|&|types}",
  "create_relationship" : "http://localhost:7474/db/data/node/38/relationships",
  "labels" : "http://localhost:7474/db/data/node/38/labels",
  "traverse" : "http://localhost:7474/db/data/node/38/traverse/{returnType}",
  "extensions" : { },
  "all_relationships" : "http://localhost:7474/db/data/node/38/relationships/all",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/38/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/38/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/38",
  "incoming_relationships" : "http://localhost:7474/db/data/node/38/relationships/in",
  "properties" : "http://localhost:7474/db/data/node/38/properties",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/38/relationships/in/{-list|&|types}"
}, {
  "metadata" : {
    "id" : 55,
    "labels" : [ "Date" ]
  },
  "data" : {
    "date" : "2017-05-12"
  },
  "paged_traverse" : "http://localhost:7474/db/data/node/55/paged/traverse/{returnType}?pageSize,leaseTime",
  "outgoing_relationships" : "http://localhost:7474/db/data/node/55/relationships/out",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/55/relationships/out/{-list|&|types}",
  "create_relationship" : "http://localhost:7474/db/data/node/55/relationships",
  "labels" : "http://localhost:7474/db/data/node/55/labels",
  "traverse" : "http://localhost:7474/db/data/node/55/traverse/{returnType}",
  "extensions" : { },
  "all_relationships" : "http://localhost:7474/db/data/node/55/relationships/all",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/55/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/55/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/55",
  "incoming_relationships" : "http://localhost:7474/db/data/node/55/relationships/in",
  "properties" : "http://localhost:7474/db/data/node/55/properties",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/55/relationships/in/{-list|&|types}"
}
```

Slika 29 – Odgovor na REST zahtjev iz Primjera 4

4.3. Model složene usluge u Neo4j bazi podataka

Usluga zamišljena i realizirana u Neo4j bazi podataka sastoji se od 3 pristupna čvora koji prikupljaju podatke koje im njihovi senzori odašilju. Ti se pristupni čvorovi nalaze na 3 kata C-zgrade FER-a te posjeduju senzore koji mjere iste vrijednosti (temperatura, vlažnost, tlak, koncentracija CO₂/SO₂/NO₂ u zraku), ali u različitim kombinacijama što znači da neki pristupni čvor ima više senzora jednog tipa, a drugi nekog drugog, npr. pristupni čvor na šestom katu ima 2 senzora za temperaturu, a onaj na sedmom samo jedan i sl. Osim toga, svaki senzor ima vlastiti parametar preciznosti te izmjerenu vrijednost i vrijeme mjerenja, a u određenom trenutku može i ne mora biti raspoloživ.

Podaci o očitanjima tih senzora koji se unose u Neo4j bazu podataka nalaze se u CSV datoteci. Datoteka se nalazi na Slici 30.

```
Gateway,Type,Value,Location,Availability,Accuracy,Time
Gw1,temperature,27,C6,true,0.93,17:35
Gw1,temperature,27,C6,true,0.96,17:37
Gw1,humidity,64.5,C6,true,0.90,17:43
Gw1,pressure,1012.3,C6,true,0.95,17:52
Gw1,pressure,1011.1,C6,true,0.91,17:54
Gw1,CO2,0.2,C6,true,0.94,17:58
Gw1,CO2,0.2,C6,true,0.89,17:58
Gw1,SO2,1.6,C6,true,0.93,18:00
Gw1,NO2,37.7,C6,true,0.97,18:00
Gw2,temperature,28,C7,true,0.93,17:40
Gw2,humidity,68,C7,true,0.90,17:45
Gw2,humidity,71.5,C7,true,0.88,17:47
Gw2,pressure,1010,C7,true,0.91,17:56
Gw2,CO2,0.25,C7,true,0.89,17:59
```

Slika 30 – Podaci za realizaciju usluge

Podaci se učitavaju Cypher upitom koji se nalazi u nastavku.

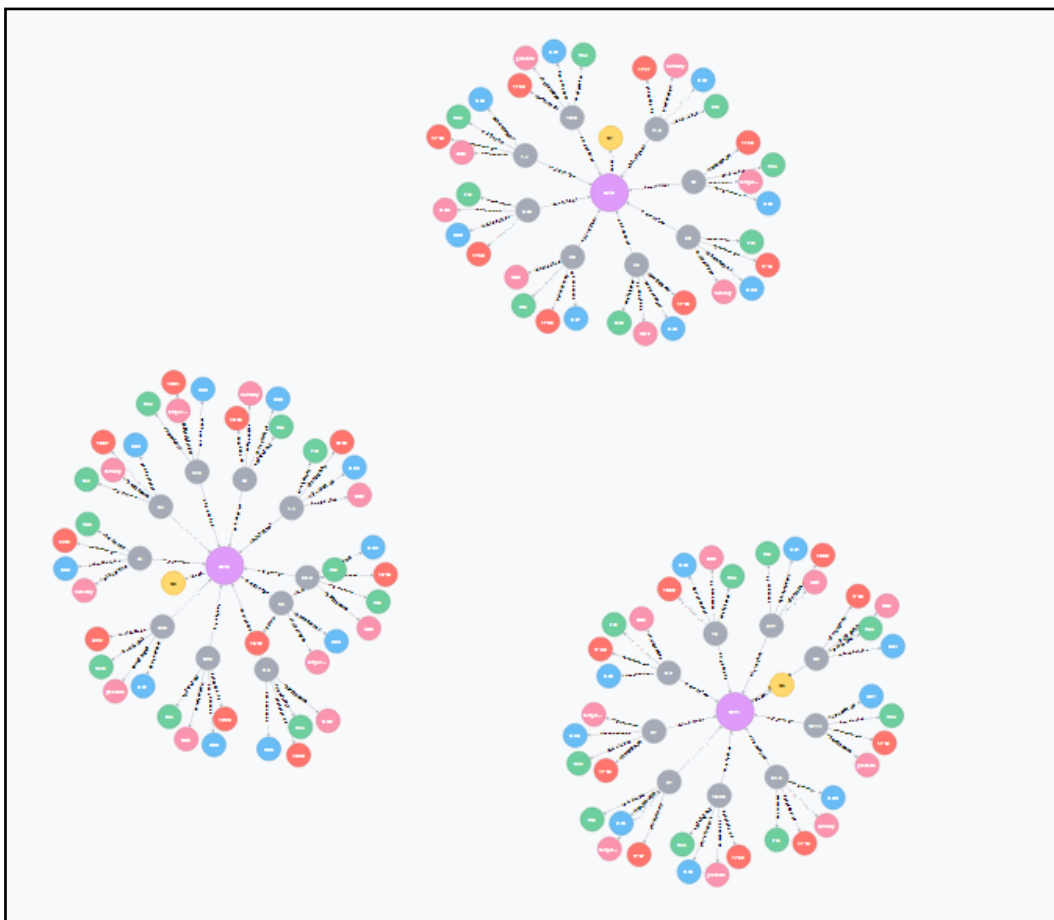
```
LOAD CSV WITH HEADERS FROM "file:///C:/Model_usluge.csv" AS csvLine
MERGE (gateway:Gateway { gateway: csvLine.Gateway })
CREATE (type:Type { type: csvLine.Type})
CREATE (value:Value { value: csvLine.Value})
MERGE (location:Location { location: csvLine.Location })
CREATE (availability:Availability { availability: csvLine.Availability })
CREATE (accuracy:Accuracy { accuracy: csvLine.Accuracy })
```

```

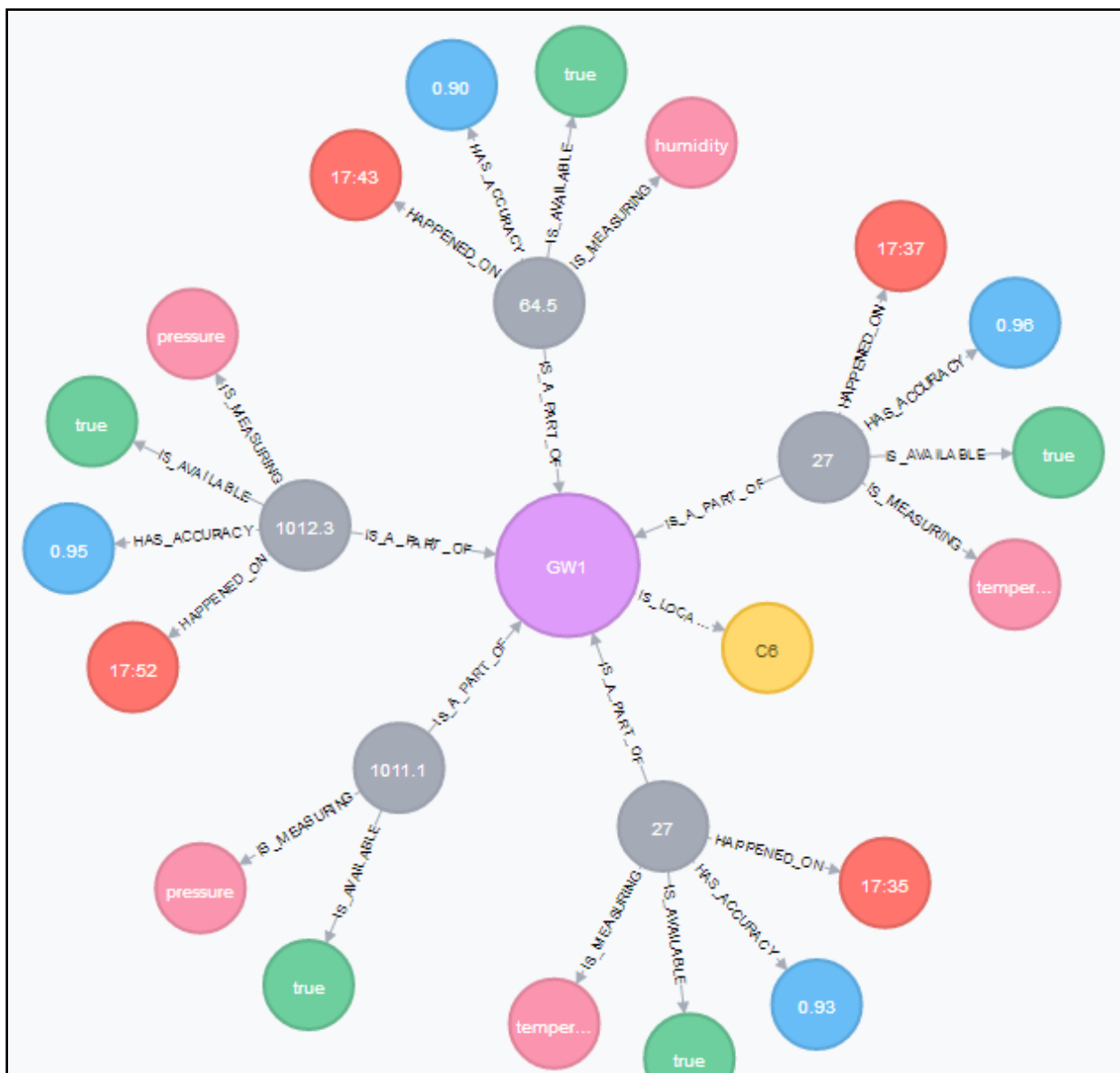
CREATE (time:Time { time: csvLine.Time })
CREATE (value)-[:IS_A_PART_OF]->(gateway)
CREATE (value)-[:IS_MEASURING]->(type)
MERGE (gateway)-[:IS_LOCATED_ON]->(location)
CREATE (value)-[:IS_AVAILABLE]->(availability)
CREATE (value)-[:HAS_ACCURACY]->(accuracy)
CREATE (value)-[:HAPPENED_ON]->(time)

```

Tako kreirana baza sadrži prikaz pristupnih čvorova koji su povezani sa svojom lokacijom te na njih povezanih senzora sa svojim vrijednostima i parametrima. Na Slici 31 nalaze se podaci svih triju čvorova, a na Slici 32 ograničeni prikaz podataka za jedan pristupni čvor.



Slika 31 – Prikaz podataka kreirane usluge



Slika 32 – Ograničeni prikaz podataka usluge

5. Zaključak

Kako bi se u potpunosti mogla iskoristiti trenutno dostupna tehnologija, potrebno je konstantno poboljšavati postojeće i smišljati nove obrasce njezine upotrebe. Upravo zato je veliki naglasak stavljen na koncept Interneta stvari gdje se skupljaju ogromne količine raznih vrsta podataka koje onda dobivaju pravi smisao kada se upotrijebe u obavljanju nekakve smislene funkcije ili posluže kao informacije krajnjem korisniku. Važno je te podatke efikasno pohraniti, uzimajući pritom u obzir njihovu povezanost, kako bi kontekst u kojem se oni koriste bio potpun. Tu je veliki značaj grafičkih baza podataka, budući da one povezuju podatke strukturama grafa kao najintuitivnijem načinu povezivanja. Ako podaci dolaze iz različitih izvora, potrebno ih je prilagoditi te stvoriti okolinu u kojoj im se može jednostavno pristupiti. Tek je tada moguće ostvariti cjelokupnu uslugu koja posjeduje određenu vrijednost i donosi korist onome koji se njome služi.

6. Literatura

- [1] Raspodijeljeni sustavi, 13. *Internet stvari*,
http://www.fer.unizg.hr/download/repository/RS-2015_13.pdf, 23.05. 2017.
- [2] Mario Weber, *Regulatorni izazovi Interneta stvari*,
https://www.fer.unizg.hr/download/repository/KDI_Mario_Weber.pdf, 25.05.2017.
- [3] McKinsey Quarterly, *The Internet of Things*,
<http://www.mckinsey.com/industries/high-tech/our-insights/the-internet-of-things>,
25.05.2017.
- [4] Ovidiu Vermesan, Peter Friess, *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*, http://www.internet-of-things-research.eu/pdf/Converging_Technologies_for_Smart_Environments_and_Integrated_Ecosystems_IERC_Book_Open_Access_2013.pdf, 26.05.2017.
- [5] Raspodijeljeni sustavi, 3. *Arhitekture web-aplikacija, tehnologije weba*,
http://www.fer.unizg.hr/download/repository/RS-2016_03.pdf, 01.06.2017.
- [6] *Neo4j*, <https://neo4j.com/>, 19.06.2017.
- [7] *Chapter 1: Service oriented architecture (SOA)*, <https://msdn.microsoft.com/en-us/library/bb833022.aspx>, 17.06.2017.
- [8] Više autora, *SOA Manifesto*, <http://www.soa-manifesto.org/>, 17.06.2017.
- [9] Roy Thomas Fielding, *CHAPTER 5 Representational State Transfer (REST)*,
http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, 20.06.2017.
- [10] Luka Cindrić, *Programska emulacija senzorskih čvorova u Internetu stvari*,
25.06.2017.

7. Sažetak

Naslov: Heterogena okolina za Internet stvari

Sažetak: Ovaj diplomski rad obrađuje ostvarenje heterogene okoline u Internetu stvari. Prvo je dana definicija koncepta Interneta stvari. Potom je objašnjena njegova arhitektura te su naznačena područja primjene i najveći izazovi u njegovom funkcioniranju. Zatim su objašnjena svojstva grafičke baze podataka, način povezivanja različitih izvora podataka u takvu bazu te kako im se može pristupiti. Na kraju su opisana svojstva usluga Interneta stvari te osnovni arhitekturni principi koji se za to koriste.

Ključne riječi: Internet stvari, grafička baza podataka, Neo4j, Cypher, heterogena okolina, SOA, REST

Title: Heterogeneous environment for Internet of Things

Summary: This master's thesis deals with the realisation of a heterogeneous environment in the Internet of Things. Firstly, the definition of the concept of the Internet of Things is given, along with the explanation of its architecture as well as where it's being used and also the challenges affecting its functioning. After that, the characteristics of a graph database are explained. It is shown how to connect data from different sources in a graph database and how to approach that data. In the end, the characteristics of a service from the Internet of Things are described, as well as some fundamental architectural principles used for that services.

Key words: Internet of Things, graph database, Neo4j, Cypher, heterogeneous environment, SOA, REST