LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

INTERAKTĪVAIS DATU APSKATES RĪKS, KAS BĀZĒTS UZ AD-HOC KOORDINĒTU GRAFA APSKATI INTERNETA PĀRLŪKĀ

KVALIFIKĀCIJAS DARBS

Autors: **Arnis Bērziņš**
Studenta apliecības Nr: ab12126
Darba vadītājs : M.Dat. Renārs Liepiņš

RĪGA 2014

ANOTĀCIJA

Kvalifikācijas darba mērķis ir radīt rīku, kas padara kveriju veidošanu ātrāku un interaktīvāku, kas ir noderīgs lietotājiem, kam kveriju veidošana (SPARQL sintaksē) ir mazāk pazīstama, bet ir priekšzināšanas IT jomā un pamatzināšanas par datu uzglabāšanas metodēm datubāzēs.

Kvalifikācijas darba izstrādes laikā ir izveidots rīks, kas darbojas Interneta pārlūkā, interaktīvi parādot RDF grafa datus programmatūras lietotājam.

Lietotnes idejas apraksts: Lietotājs var izvēlēties, kuru RDF grafa datubāzi parādīt pārlūkā. Lietotājs ar datorpeles un kursora palīdzību Interneta pārlūkā veido grafiskus elementus.

Izstrādātā programmatūra ir kā zinātnisks prototips, kuram vēlama tālāka uzlabošana.

Programmatūras izstrādes gaitā tika lietots spirālveida dzīves cikla modelis.

ABSTRACT

The main goal of this qualification work is to design a tool that will make query designing faster and more interactive. That will be helpful to users that don't really know this query syntax (SPARQL) so perfectly but who are competent in the IT field and knowledge about databases.

As a result of this project a tool is produced that works in an Internet browser, interactively showing a RDF graphs data to the user.

Interface description: User can select which RDF graphs database to visualize with this tool in the browser. With the help of the mouse user makes graphical elements in the browsers container.

Designed tool is still a prototype so place for improvement might be necessary.

In the process of making this project spiral life cycle model was used.

TERMINU SKAIDROJUMS

Ad-hoc - izstrādāde vienam konkrētam mērķim, bez rūpīga iepriekšēja plāna izstrādes.

Atkarība - saite starp divām virsotnēm, kas tiek izmantota, lai sasaistītu ārējos ierobežojumus ar attiecīgajām virsotnēm.

Aktīvā atlase - datu kopa virsotnē, ko izmanto kverijs, lai izveidotu ārējos ierobežojumus saistītajām virsotnēm.

Ārējie ierobežojumi - tie ir saistīto virsotņu iekšējie ierobežojumi, kas samazina virsotnes rezultatīvā skata datu kopu.

Aktīvais vizualizācijas tips - virsotnes izvēlētais tips, kas darbojas kā filtrs rezultatīvā skata datiem. Ja nav izvēlēts neviens tips, tad tiek parādīti visi dati, pēc idejas, ka visi tipi ir aktīvajā vizualizācijas tipā.

Entītija – konkrēta datubāzes tabula.

ER modelis – (Entītiju relāciju) datubāzes modelis, kas attēlo datubāzes shēmu, attēlojot attiecības starp entītijām.

Hiperatkarība - saite starp virsotni un atkarību, kas tiek izmantota, lai sasaistītu ārējos ierobežojumus ar attiecīgajām virsotnēm.

Iekšējie ierobežojumi - datu filtri, kas samazina rezultatīvajā skatā datu kopu, iekšējie ierobežojumi var būt Tipa izvēle vai Atlase .

Kverija Grafa skats - skats, ko redz lietotājs lietojot produktu. Kveriju Grafa skats sastāv no strukturālā un rezultatīvā skata.

Kverija kompilators - datu struktūra, kas paņem kverija topoloģiju un izveido no tā kveriju.

Kverija topoloģija - elementu apkopojums noteiktā struktūrā.

Kverija modelis - modelis kurā tiek turēti kverija topoloģijas elementi.

PPS - programmatūras prasību specifikācija.

PPA - programmatūra projektējuma apraksts.

RDF - (*Resource Description Framework*) resursu apraksta ietvars – Specifiska datubāzes struktūra, kas balstās uz semantiskā tīkla principa.

Rezultatīvais skats - datu attēlojums virsotņu iekšienē, tie ir dati, kas ir iegūti no kverija izsaukuma uz serveri.

Renderētājas - metožu kopa, kas uzzīmē Kverija modeļa elementus vizuāli un apstrādā to īpašības starp kverija grafa skatu un kverija modeli.

Strukturālais skats - kopējais skats, ko lietotājs redz savā pārlūkā. Virsotņu un atkarību izvietojums.

Virsotne - Kverija Grafa skata elements, kas satur rezultatīvo skatu, informācija par virsotni tiek turēta Kverija modelī.

Darbā tika izmantoti termini, kuru tulkojums tika apskatīts Terminu vārdnīcā [9].

Contents

IEVADS

Labs veids, kā saprast datus, ir tos vizualizēt, tāpēc šī projekta pamatā ir uzdevums izveidot saskarni, kas citādāk ļaus lietotājam nonākt pie datiem, izmantojot grafisku saskarni. Viens attēls spēj aizstāt tūkstoš vārdus.

Darba gaitā tiks izveidots rīks, kas ļauj lietotājam apskatīties uz datiem citādākā veidā, nekā mūsdienās ir pierasts. Rīka saskarne ar lietotāju notiks ar interneta pārlūka palīdzību. Rīka pamatā ir ideja padarīt kverija ievades procesu lietotājam draudzīgāku, mainot līdzšinējo kverija tekstuālo formu uz lietotāja izveidotu grafa struktūru, ko programma pēc tam attiecīgi apstrādās un izveidos tekstuālu kveriju, ko izsauks uz serveri. Saņemtie dati no servera tiks attēloti nevis atsevišķā logā, bet gan tajos pašos grafa struktūras elementos, virsotņu iekšienēs.

Darba gaitā tiks veikta:

- Struktūras elementu izveide
- Kverija izveidošana no topoloģijas
- Servera iegūto datu attēlošana

Piedāvātais grafiskais kveriju uzdošanas veids ir pretstatā klasiskajai pieejai, kur kverijs tiek uzdots tekstuālā formā, un iegūtie rezultāti tiek attēloti vienā tabulā. Saskarne klasiskajai pieejai sastāv no diviem pamatelementiem: tekstuāls ievades lauks un izvade tabulas formā. Ievades laukā tiek padots kverijs tekstuālā formā, kverijs tiek izsaukts uz datubāzi un datubāze savu atbildi attēlo izvades laukā, kur dati tiek parādīti tabulā. Strādāšana ar šiem elementiem ir pietiekami intuitīva, ja lietotājs zina kveriju sintaksi, datubāzes struktūru un informāciju, ko lietotājs vēlas iegūt no datiem, bet ja tā nav, tad ir pārliecinoši grūtāk.

Klasiskā tekstuālo kverija mīnusi saistās arī ar to, ka kverijs spēj izpildīties tikai tad, kad tas ir pabeigts, tādēļ tekstuālā kverija rakstīšanas laikā lietotājs neredz starprezultātus. Pretstatā šajā darba izstrādātā rīkā lietotājs gandrīz vienmēr redzēs, ar kādiem datiem viņš darbojas.

Tā kā klasiskajā modelī nākas lēkāt no tekstuāla kverija uz grafisku datu struktūru serverī, tas var radīt problēmas izpratnē, ko ir vērts uzlabot. Ja jau ER modeli, kas ir datubāzes relāciju attēlojums grafa formā, izmanto, lai iegūtu vispārēju izpratni par datubāzi, tad kādēļ arī kverijus nevarētu padot grafiskā formā?

Autors izveidos programmatūru, kas ļaus lietotājam uzdot kveriju grafiskā formā, līdzīgi kā ER modeļi ļauj lietotājam grafiskā formā uzdot datubāzes struktūru. Kveriju grafiskā forma sastāvēs no diviem pamat elementiem – virsotnēm un atkarībām, gluži kā grafā. Izveidojot virsotņu iekšienēs ierobežojumus un sasaistot virsotnes ar atkarībām, lietotājs varēs interaktīvi apskatīt datus datubāzē, nemainot topoloģijas struktūru. Šī pieeja ir piezemētāka lietotājiem, kas nav pārāk pazīstami ar SPARQL kveriju valodu un kas vēl pilnībā neizprot izvēlētās datubāzes struktūru.

Jānorāda, ka rīka prototips tiks izveidots LUMII projekta ietvaros, izmantojot tajā izveidotās iestrādes tekstuāla kverija ģenerēšanā no grafiskas topoloģijas.

1. PROGRAMMATŪRAS PRASĪBU SPECIFIKĀCIJA

1.1. Ievads

1.1.1. Nolūks

Programmatūras prasību specifikācijas veidošanas nolūks ir aprakstīt prasības programmatūrai "Interaktīvais datu apskates rīks, kas bāzēts uz ad-hoc koordinētu grafa apskati interneta pārlūkā" un noskaidrot visas detaļas, kas ir nepieciešamas, lai labi aprakstītu un noprojektētu programmatūru tālākai lietošanai. Dokuments ir paredzēts kā papildinājums programmatūras izstrādātājiem un tās lietotājiem [5].

1.1.2. Darba sfēra

Programmatūra ir paredzēta kompleksu datu saprašanai un interaktīvai datu apstaigāšanai, kas neprasa pārāk dziļas zināšanas kveriju veidošanā un datu bāžu struktūras izpratnē.

Programmatūrai jāspēj attēlot dažādas RDF grafu datus. Rīks ir spējīgs pieslēgties *Stardog* datubāzei, ko lietotājs ir izveidojis vai arī paņēmis gatavu. Programmai jāļauj lietotājam izvēlēties, kuru datubāzi parādīt. Lai rīks sāktu darbu, tam ir jānorāda, kuru datubāzi lietotājs vēlas attēlot grafiski, kas tiks attēlots pēc lietotāja ievadītajiem parametriem.

1.1.3. Saistības ar citiem dokumentiem

Dokuments ir izveidots neformālas sazināšanās rezultātā, kas norisinājās tiekoties klātienē. Dokuments tiks izmantots kā paraugs izstrādājot programmatūras projektējuma aprakstu.

Sastādot šo dokumentu, tika izmantots Latvijas Standarts LVS 68:1996 „Programmatūras prasību specifikācijas ceļvedis" [1].

1.1.4. Pārskats

PPS sastāv no trīs nodaļām:

- Pirmajā nodaļā tiek aprakstīts īss ievads par PPS.

- Otrajā nodaļā tiek runāts par programmatūras vispārējo aprakstu.
- Trešajā nodaļā tiek detalizēti aprakstītas visas programmatūras funkcijas, kā arī nefunkcionālās prasības, kas nepieciešamas, lai tiktu izveidots detalizēts projektējums.

1.2. Vispārējais apraksts

1.2.1. Produkta perspektīva

Programmatūra ir RDF grafa vizualizētājs, kas veic interaktīvu datu apskati. Produkts ir paredzēts kā zinātnisks prototips, ar kuru tiek veikti izmēģinājumi ar mērķi noskaidrot, kā lietotājam grafiski labāk parādīt no datu bāzes iegūtos datus.

1.2.2. Produkta funkcijas

Programmatūra nodrošina dažādas funkcijas, kuras lietotājam jāspēj izpildīt tiešā vai netiešā veidā.

1. Konekcijas izveide ar datubāzi - lietotājs spēj padot savas datubāzes parametrus, kas atļaus programmatūrai veiksmīgi pieslēgties datubāzei.

2. Virsotnes izveide - tā saturēs datus no RDF grafa. Šis ir galvenais programmatūras elements, caur kuru mēs varam apskatīt datus no daudzām dažādām pusēm.

3. Virsotņu dzēšana - lietotājs var izdzēst virsotni, ja lietotājs kāda iemesla dēļ to vēlas.

4. Virsotnes pārvietošana - lietotājs spēj pārvietot virsotnes un izkārtot tās savā internetā pārlūkā pēc izvēles.

5. Virsotnes izmēra izmaiņa - lietotājs var izmainīt virsotnes lielumu jeb izmērus pēc savas nepieciešamības

6. Virsotnes rezultatīvā skata maiņa - izmaina servera datu attēlošanas veidu virsotnes ietvaros.

7. Tipa izvēle - parādīt rezultatīvajā skatā tikai tos datus, kam ir noteikts tips.

8. Atkarību izveidošana - ļauj lietotājam izveidot atkarību starp virsotnēm, kas uzliek dažādus ierobežojumus datiem rezultatīvajā skatā. To var izmantot, lai samazinātu redzamo datu apjomu un spētu sameklēt konkrēti vēlamo datu kopu rezultatīvajā skatā.

9. Atkarību dzēšana - lietotājs var izdzēst atkarību, ja tā sāk traucēt lietotāja darba gaitā.

10. Vērtību atlasīšana - lietotājs spēj atlasīt vērtības no rezultatīvā skata datiem, kas darbosies kā ārējie ierobežojumi citu virsotņu rezultatīvajiem skatiem.

11. Kolonnu izveide - lietotājs spēj izveidot kolonas, kas tiek pieliktas virsotnes rezultatīvajam skatam

12. Kolonnu izdzēšana - lietotājs var izdzēst nevajadzīgās kolonnas

13. Kolonnu rediģēšana - lietotājs var izmainīt kolonnu īpašības

14. Strukturālā skata nodrošināšana

15. Rezultatīvā skata nodrošināšana

1.2.3. Lietotāja raksturiezīmes

Lietotājam jābūt pamatiemaņām darbam ar datoriem, lai spētu pilnvērtīgi izmantot izstrādāto produktu.

Lietotājam ar jābūt pazīstamam vismaz nedaudz ar RDF grafiem un vienkāršām ontoloģijas zināšanām.

1.2.4. Vispārējie ierobežojumi

Programmatūra nesatur jau gatavu RDF grafu, tādēļ to jānodrošina lietotājam. Kā arī programmatūra nav atbildīga par kļūdām padotajā RDF grafā.

1.3. Konkrētās prasības

Nodaļā tiek specificētas pievienojumprogrammas konkrētās prasības.

1.3.1. Funkcionālās prasības

Šeit tiek aprakstītas visas programmatūras funkcionālās prasības.

1.3.1.1. Konekcijas datu ievade

Identifikators: Connection_data_input
Mērķis
Izveidot sakarus ar lietotāja izvēlētu datubāzi
Apraksts
Lietotājam var piederēt vairākas datubāzes. Lai katru reizi, kad lietotājs vēlas apskatīt citu datubāzi, viņam nebūtu jāmaina koda saturs, lietotājs var ievadīt sava RDF grafa parametrus un programmatūra izmantos tos, lai parādītu šīs datubāzes datus.
Ievade
Datubāzes vārds, adrese, lietotāja vārds, parole
Apstrāde
1. Lietotājs ievada parametrus attiecīgajos laukos 2. Programma apstrādā tos 3. Tiek izveidota konekcija ar datubāzi 4. Programmatūra ir gatava attēlot datubāzes datus
Komentāri
Ja lietotājs ievada nepareizus datus, tad netiek nodrošināts paziņojums, ka konekcija nav izdevusies.

1.3.1.2. Virsotnes izveidošana

Identifikators: make_node
Mērķis
Izveidot virsotni, ko redzēs lietotājs un katra virsotne saturēs datus no servera
Apraksts
Funkcija izveido virsotni un to parāda strukturālajā skatā, virsotne satur rezultatīvo skatu, kur tiek parādīti servera dati, ko iegūst no kverija izsaukuma uz serveri. Virsotne ir viens no galvenajiem projekta elementiem, jo satur datus, kas mums interesē.
Ievade
Vieta, kur lietotājs vēlas izveidot virsotni
Apstrāde

1. Saņem vietu, kur jābūt jaunajai virsotnei 2. Izveido virsotni datos 3. Parāda virsotni strukturālajā skatā
Komentāri
Virsotnes nevajadzētu veidot pārāk tuvu vienu otrai, jo nav specificēts nepārklāšanās princips.

1.3.1.3. Virsotnes izdzēšana

Identifikators: delete_node
Mērķis
Izdzēš virsotni, kas vairs nav vajadzīga
Apraksts
Funkcija izdzēš virsotni, lai virsotne vairs netiek parādīta strukturālajā skatā.
Ievade
Virsotnes dzēšanas notikuma izsaukums (iespējams uzspiežamas pogas formātā)
Apstrāde
1. Lietotājs izsauc dzēšanas notikumu 2. Izdzēš visus datus par virsotni no saistītiem elementiem 3. Izdzēš pašu virsotni 4. Atjauno strukturālo un rezultatīvo skatu, kur vairs nav redzama izdzēstā virsotne

1.1.3.4. Virsotņu pārvietošana

Identifikators: drag_nodes
Mērķis
Novietot virsotni lietotājam pieņemamā vietā
Apraksts
Lietotājs var pārbīdīt virsotni, ja virsotņu izvietojums viņam liekas nepiemērots, tas atļauj lietotājam novietot vairākas virsotnes blakām un vienkārši salīdzināt datus katrā.
Ievade
Saņem virsotnes vietu, kur virsotne atradīsies pēc pārbīdes
Apstrāde
1. Saņem virsotnes 2. Saņem jauno atrašanās vietu 3. Pārbīda virsotni 4. Atjauno strukturālo skatu
Komentāri
Lietotāja ziņā paliek virsotņu izkārtojums, lai tās nepārklājas .

1.1.3.5. Virsotnes izmēra izmaiņa

Identifikators: resize_node
Mērķis
Izmainīt virsotnes izmērus, ja ir tāda nepieciešamība
Apraksts
Lietotājs var mainīt virsotnes izmērus pēc ieceres; ja ir daudz datu, tad var palielināt virsotni, bet ja ir maz datu un virsotnes izskatās tukšas, tad tās var arī samazināt.
Ievade
Jaunie izmēri
Apstrāde
1. Iegūst virsotnes datus no Kveriju Grafa 2. Padod jaunos izmērus 3. Izmaina virsotnes izmērus Kverija grafā 4. Atjauno strukturālo skatu
Komentāri
Jābūt noteiktam virsotnes lielumam, lai virsotni nevarētu samazināt par daudz

1.1.3.6. Rezultatīvā skata attēlošanas veidu maiņa

Identifikators: toggle_node_view
Mērķis
Pamainīt rezultatīvā skata veidu
Apraksts
Rezultatīvo skatu varam mainīt uz citiem skatiem, oriģinālā ideja ir ar tabulām, kas attēlo datus kolonnās, bet ir iespējami citi vizualizācijas veidi, kā piemēram stabiņu diagrammas.
Ievade
Virsotnes dati
Apstrāde
1. Inicializē notikumu, kas maina skatu 2. Pārģenerē kverijus 3. Apskata, kāds skata veids ir iestatīs Kveriju Grafā attiecīgajai virsotnei 4. Pēc skata noteiktajām īpašībām skatā tiek attēloti dati
Kļūdu paziņojumi
Nav vēl atnākuši nepieciešamie dati (*buffering*...) Kļūda datos („*Error*…”)

1.1.3.7. Datu tipa izvēle

Identifikators: select_type

Mērķis
Samazināt datu kopu virsotnes ietvaros, lai tiktu parādīti tikai tie dati, kam ir saite uz vienu datu tipu RDF grafā
Apraksts
Virsotnēm var būt visādi ierobežojumi, tomēr pie lieliem RDF grafiem parādītā datu kopa vēl joprojām var būt liela, tāpēc ir iespēja izvēlēties, lai virsotnē tiktu parādīti tikai dati, kam ir kāds noteikts tips.
Ievade
Rezultatīvā skata kverijs
Apstrāde
1. Modificējam rezultatīvā skata kveriju, lai dabūtu visus izmantotos tipus 2. Šos tipus parādam kādā izvēlnē, lai būtu no kā izvēlēties 3. Ja kāds no tipiem tiek izvēlēts, tad kverijs tiek pārģenerēts, atlasot tikai tos datus, kam ir tāds tips
Kļūdu paziņojumi
„*Error*...” rezultatīvajā skatā.

1.3.1.8. Atkarības izveidošana

Identifikators: make_edge
Mērķis
Izveidot atkarību starp divām virsotnēm vai virsotni un atkarību, kas parādīs saites starp rezultatīvo skatu datiem
Apraksts
Atkarību izveido ar funkcijas izsaukšanu, kas izveido grafisku līniju starp divām virsotnēm vai virsotni un atkarību.
Ievade
Lietotājs izvēlas, starp kuriem diviem elementiem tiks izveidota atkarība
Apstrāde
1. Saņem abus elementus, ko lietotājs ir izvēlējies 2. Pārbauda vai starp šiem diviem elementiem var būt atkarība 3. Izveido atkarību 4. Atjauno strukturālo un rezultatīvo skatu

1.3.1.9. Izdzēst atkarību

Identifikators: delete_edge
Mērķis
Tiek vaļā no atkarības, kas vairs nav vajadzīga vai traucē
Apraksts

Atkarība tiek izdzēsta no Kveriju modeļa un no piesaistītajiem elementiem, kad tiek izdzēsta kāda no virsotnēm, kas ir saistīta ar atkarību. Šādi mēs tiekam vaļā no atkarībām, kas vairs nav nepieciešamas.
Ievade
Pogas vai cita notikuma inicializēšana, kas padod atkarību
Apstrāde
1. Saņem atkarību 2. Izdzēš atkarību no saistītajiem elementiem 3. Izdzēš atkarību 4. Atjauno strukturālo un rezultatīvo skatu

1.3.1.10. Vērtību atlasīšana

Identifikators: data_selection
Mērķis
Atlasīt specifiskus datus no rezultatīvā skata, lai dati, ko izmanto citu virsotņu datu iegūšanai, vairs nebūtu tik vispārēji
Apraksts
Lietotājs spēj atlasīt datus no rezultatīvā skata katras virsotnes ietvaros, pēc tam kveriji tiks ģenerēti nevis no visiem rezultatīvā skata datiem, bet tikai no atlasītajām vērtībām. Tas nodrošina iespēju vienkārši un ātri saprast, kādam elementam vai elementu kopai ir saites ar citiem elementiem, ja ir novilkta saite starp virsotnēm. Ja nekādas vērtības nav atlasītas, tad visi dati ir Aktīvās atlases sastāvdaļa
Ievade
Tiek padota datu apakškopa no virsotnes rezultatīvā skata
Apstrāde
1. Saņem datu kopu, kas ir atlasīta 2. Kopu saglabā 3. Atjauno kverijus katrā virsotnē, ievērojot apakškopas datus, nevis visus datus, kas bija virsotnes rezultatīvajā skatā 4. Atjauno visu virsotņu rezultatīvos skatus, lai tiktu parādīti jaunie dati.

1.3.1.11. kolonnas izveidošana

Identifikators: create_column
Mērķis
Izveidot jaunu kolonnu un to pievienot virsotnei
Apraksts
Tiek izsaukta kolonnas izveidošanas funkcija, kolonna var parādīt datu atribūtus vai cik reizes tādi dati parādās RDF grafā. Kolonna tiek pievienota virsotnei, no kuras tika izsaukta šī funkcija.
Ievade
Kolonnas īpašību forma

Apstrāde
1. Lietotājs atver logu, no kura var izveidot kolonnu 2. Lietotājs aizpilda nepieciešamos laukus par kolonnu 3. Kolonna tiek izveidota 4. Tiek atjaunots rezultatīvais skats, kur var redzēt kolonnas

1.3.1.12. kolonnu dzēšana

Identifikators: delete_column
Mērķis
Tikt vaļā no nevajadzīgajām kolonnām
Apraksts
Kolonna tiek izņemta no virsotnes
Ievade
Lietotājs izvēlas kolonnu, no kuras vēlas tik vaļā
Apstrāde
1. Lietotājs atver logu, kur var rediģēt kolonnas 2. Lietotājs izvēlas izdzēst vienu kolonnu 3. Kolonna tiek izdzēsta 4. Atjauno skatus, lai parādās atlikušās kolonnas

1.3.1.13. kolonnas rediģēšana

Identifikators: edit_column
Mērķis
Rediģēt kolonnu, lai tajā parādītos vēlamie dati
Apraksts
Lietotājs manuāli var izmainīt kolonnas īpašības pēc paša vajadzībām.
Ievade
Formas aizpilde
Apstrāde
1. Lietotājs atver logu, kur parādās kolonnas īpašību forma 2. Lietotājs to aizpilda pēc savām iecerēm 3. Atjauno rezultatīvo skatu, lai parādās kolonnas jaunie iestatījumi
Kļūdu paziņojumi
Nav?
Komentāri
Jābūt aizpildītiem visiem laukiem, pirms tiek izdarītas citas operācijas, citādāk rodas problēmas

1.3.1.14. Strukturālā skata nodrošināšana

Identifikators: render_graph_view
Mērķis
Uztur strukturālo skatu, lai nerastos pārpratumi ar to, kas notiek ar elementu topoloģiju.
Apraksts
Strukturālo skatu atsvaidzina ar jauniem datiem tad, kad kādi dati Kverija modelī tiek mainīti. Šī funkcija tiek izsaukta pēc tādām metodēm kā virsotnes izveidošana, virsotnes dzēšana, atkarības izveidošana, virsotnes pārbīde un vairākām citām. Šī ir viena no centrālajām funkcijām, kas atbild par Kverija Grafa skata nodrošināšanu
Ievade
Saņem Kveriju Grafu
Apstrāde
1. Saņem Kveriju grafu 2. Apskata, kādus elementus ir jāizveido un kādi elementi jau ir izveidoti vai arī kādas īpašības tajos ir mainījušās 3. Ja kaut kādas izmaiņas ir veiktas, tad skats tiek atjaunots

1.3.1.15. Rezultatīvā skata nodrošināšana

Identifikators: render_node_view
Mērķis
Uztur rezultatīvo skatu, lai dati no kverija izsaukuma tiktu piemēroti attēloti virsotnēs
Apraksts
Tiek atjaunots rezultatīvo datu skats, kas attēlo datus, kas ir iegūti no ģenerētā kverija, kas tika izsaukts uz servera. Katru reizi, kad notiek izmaiņas rezultatīvajā skatā, piemēram, tiek izveidota jauna atlase kādā virsotnē vai tiek izdzēsta kāda atkarība, tad katras virsotnes rezultatīvais skats tiek atjaunots. Rezultatīvā skata atjaunošana darbojas kopā ar kveriju ģenerēšanu
Ievade
Kveriju dati
Apstrāde
1. Izsauc katram elementam kverija ģenerēšanu 2. Saņem jaunā kverija datus 3. Attēlo tos katras virsotnes rezultatīvajā skatā

1.3.2. Ārējās saskarnes prasības

Šajā sadaļā tiek specificētas programmatūras ārējās saskarnes prasības.

1.3.2.1. Lietotāja saskarne

Pievienojumprogrammas saskarnes logam vajag būt izveidotam tā, lai lietotājs viegli un ātri spētu saprast, kādas darbības ir pieejamas šajā programmā. Ja lietotājam tomēr sanāk izveidot kādu nepareizību, kas nav specificēta, tad programmatūra spēj ignorēt daļu no tām.

1.3.2.2. Programmatūras saskarne

Programmatūra izmanto interneta pārlūku, caur kuru lietotājs apskata datus. Programma pieslēdzas datubāzei, ko nodrošina lietotājs.

1.3.3. Atribūti

Šajā sadaļā tiek specificēti pievienojumprogrammas atribūti.

1.3.3.1. Pieejamība

Dati atrodas uz servera. Par datu uzglabāšanu atbild administratori.

Klients var pieslēgties datiem, izmantojot šo programmatūru, un tos redzēt savā interneta pārlūkā.

1.3.3.2. Pārnesamība

Pārnesamībai ir divas daļas.

Produkts pats ir tikai HTML lapa, kam vajag pieslēgties internetam. Pārnest šo daļu uz citām ierīcēm nav problemātiski. Tā darbojas uz jebkādas interneta lietotnes, kas atbalsta HTML5.

Otra ir servera daļa, par ko programmatūra neatbild. Tur vai nu ir jāpieslēdzas attālinātam serverim, vai arī lokāli jāuzinstalē *Stardog* serveris (no http://stardog.com/).

2. PROGRAMMATŪRAS PROJEKTĒJUMA APRAKSTS

2.1. Ievads

2.1.1. Nolūks

Programmatūras projektējuma apraksta izveidošanas iemesls ir nodrošināt veiksmīgu izstrādi programmatūrai "Interaktīvais datu apskates rīks, kas bāzēts uz ad-hoc koordinētu grafa apskati interneta pārlūkā" atbilstoši izvirzītajām prasībām, kas ir specificētas prasību specifikācijā. Dokuments ir paredzēts programmatūras izstrādātājiem kā palīgmateriāls [6].

2.1.2. Darba sfēra

Programmatūra ir RDF grafa vizualizācija, kas palīdz izprast datus un ļauj interaktīvi tos parādīt partneriem.

Programmatūra izmanto interneta pārlūku, lai vizualizētu datus. Lietotājs izveido grafa struktūru no virsotnēm un atkarībām un virsotņu ietvaros redz, kādi dati tiek atlasīti. Izveidotā grafa struktūra darbojas kā kverijs.

2.1.3. Saistības ar citiem dokumentiem

Dokuments ir izstrādāts pēc LVS 72:1996 „Ieteicamā prakse programmatūras projektējuma aprakstīšanai" standarta un ir saistība ar programmatūras prasību specifikāciju [2].

2.2. Dekompozīcijas apraksts

Dekompozīcijas apraksta mērķis ir demonstrēt programmatūras sadalījumu vienumos. Tiek aprakstīts katra vienuma nolūks.

2.2.1. Vienumu dekompozīcija

2.2.1.1. Virsotnes skats (Identifikators NV.1)

Nolūks	
Nodrošināt visas funkcijas, kas būs atbildīgas par virsotnes darbībām	
Funkcija	
Veic noteiktas darbības ar virsotni.	
Funkcijas apraksts	**Funkcijas nosaukums**
Funkcija izveido virsotni (Atsauce uz lielikumu. [2] koda fragmets).	make_node
Funkcija izdzēš virsotni no Kveriju modeļa un izdzēš visas atkarības, kas ir saistītas ar šo virsotni	delete_node
Funkcija nodrošina virsotnes pārvietošanu pa *htmp* konteineri	node_drag
Funkcija nodrošina virsotnes izmēru maiņu	resize_node
Izvēlēties tipu virsotnes datiem, kas tiks parādīti. Tips darbojas kā filtrs.	select_type

2.2.1.2. Atkarību skats (Identifikators EV.2)

Nolūks	
Nodrošina visas funkcijas, kas ir atbildīgas par atkarību apstrādi	
Funkcija	
Izveido, dzēš un darbojas ar atkarībām	
Funkcijas apraksts	**Funkcijas nosaukums**
Funkcija izveido atkarību Kveriju Grafā no elementiem, kas tika padoti	draw_edge
Funkcija izdzēš atkarību no visiem saistītajiem elementiem	delete_edge
Plūsmas virziena maiņa, plūsmas mainās cikliski. Ir četri atšķirīgi plūsmu veidi (Atsauce uz pielikumu, [3] koda fragmets).	toggle_edge_flow

2.2.1.3. Kverija grafa skats (Identifikators: QGV.3)

Nolūks	
Saturēt visas funkcijas, kas atbild par skatu uzturēšanu	
Funkcija	
Skatu atjaunošana, un kverija grafa skata inicializēšana	
Funkcijas apraksts	**Funkcijas nosaukums**
Funkcija atjauno strukturālo skatu(Atsauce uz pielikumu [1] koda	render_graph

fragmets).	
Funkcija atjauno rezultatīvo skatu	fill_node_view

2.2.1.4 Datu atlases apstrādātājs (Identifikators:DSH.4)

Nolūks	
Apkopot funkcijas, kas atbild par datu atlasi	
Funkcija	
Savāc atlasi, saglabā to un apstrādā to, lai citas kverijs to var izmantot.	
Funkcijas apraksts	**Funkcijas nosaukums**
Funkcija izveido jaunu atlases kopu	make_selection
Funkcija izdzēš atlasi no virsotnes	delete_selection

2.2.1.5. Kveriju ģenerētājs (Identifikators: QG.5)

Nolūks	
Nodrošina visas funkcijas, kas piedalās kverija veidošanā	
Funkcija	
Izveido kverija iekšējo struktūru un citas lietas, ko izmanto kveriju ģenerētājs	
Funkcijas apraksts	**Funkcijas nosaukums**
Izveido kveriju ar visām tā komponentēm	generate_query
Savāc ierobežojumus kverijam no elementiem, kas nav saistīti ar virsotni, kam tiek veidots kverijs	constraint
Savāc ierobežojumus kverijam no virsotnes iekšējiem datiem, piem., tipa izvēles vai datu atlases	local_filter

2.2.1.6. Kolonnu apstrādātājs (Identifikators CH.6)

Nolūks	
Apkopo visas funkcijas, kas ir saistītas ar kolonnu apstrādi	
Funkcija	
Izveidot, dzēst un mainīt kolonnu īpašības	
Funkcijas apraksts	**Funkcijas nosaukums**

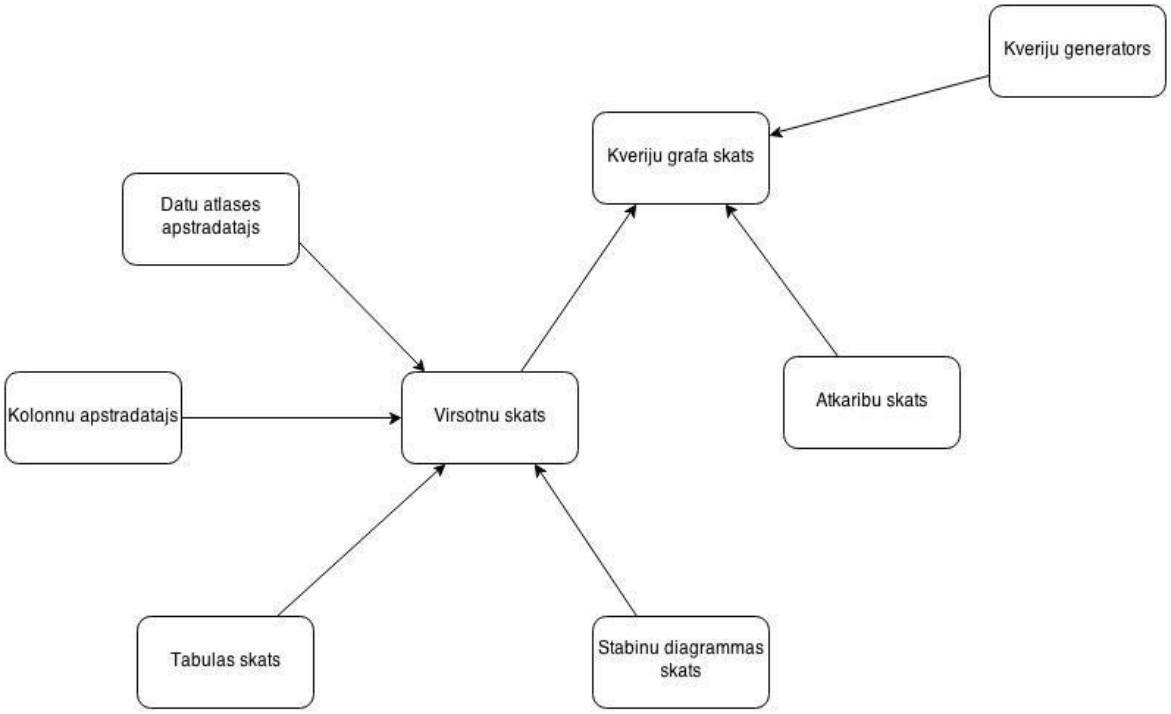
Funkcija izveido jaunu kolonnu virsotnei	create_column
Funkcija izdzēš kolonnu no virsotnes	delete_column
Funkcija izmaina kolonnas īpašības	edit_column_parameters

2.2.1.7. Tabulas skats (Identifikators TV.7)

Nolūks	
Nodrošināt virsotnes rezultatīvā skata attēlojumu tabulas formā	
Funkcija	
Apstrādāt datus rezultatīvajā skatā, lai tie tiktu attēloti tabulas formā	
Funkcijas apraksts	**Funkcijas nosaukums**
Funkcija izveido tabulas skatu (Atsauce uz pielikumu [4] koda fragmets).	init_table_view
Funkcija ievieto kverija datus tabulā.	set_data
Funkcija izdzēš visus datus par tabulas skatu	destroy
Funkcija atjauno tabulas skatu, ja kas ir mainījies	refresh_view_from_node

2.2.1.8. Stabiņu diagrammas skats (Identifikators BCV.8)

Nolūks	
Nodrošināt virsotnes rezultatīvā skata attēlojumu stabiņu diagrammas formā	
Funkcija	
Apstrādāt datus rezultatīvajā skatā, lai tie tiktu attēloti stabiņu diagrammas formā	
Funkcijas apraksts	**Funkcijas nosaukums**
Funkcija izveido stabiņu diagrammas skatu	init_table_view
Funkcija ievieto kverija datus diagrammā.	set_data
Funkcija izdzēš visus datus par diagrammas skatu	destroy
Funkcija atjauno stabiņu diagrammas skatu, ja kas ir mainījies	refresh_view_from_node



2.1. att. **Vienumu atkarības.**

2.2.2. Datu dekompozīcija

Kverija modelis satur visus elementus, kas tiek attēloti Kverija Grafa skatā.

Kverija modeļa sastāvs :

1. Prefiksi
2. Servera konekcijas parametri
3. Virsotnes
4. Atkarības

2.2.2.1. Prefiksu dekompozīcija

Prefiksi

- prefiksa identifikators - simbolu virkne, ko saīsinām ar prefiksa aizstājēju
- identifikatora aizstājējs - vārds, kas tiek pielikts pirms vērtības

2.2.2.2. Servera konekcijas parametru dekompozīcija

Servera konekcijas parametri

- servera vārds - servera nosaukums, ko izmanto, lai programmatūra zinātu ar kuru datubāzi jāsazinās
- adrese - adrese un URL vārds, kas norāda, caur kurieni piekļūt datubāzei

- lietotāja vārds - lai izveidotu sakarus ar datubāzi ir nepieciešams autorizēties lielākoties gadījumos, šeit tiek turēts autentifikācijas vārds
- parole - autentifikācijas parole
- secināšanas režīms (*reasoning mode*)

2.2.2.3. Virsotnes dekompozīcija

Virsotnes

- vārds - virsotnes unikāls nosaukums, kas tiek ģenerēts automātiski
- ienākošās līnijas – uzskaitījums, ar kādām atkarībām virsotne ir saistīta
- kverija parametri - satur tādus datus kā atlase, tipi
- ģeometrija - x un y koordinātas, kur atradīsies virsotne un pašas virsotnes augstums un platums
- aktīvais vizualizācijas tips - tipa nosaukums, kādā virsotnes skatā rezultatīvie dati tiks attēloti
- vizualizācijas definīcijas -
- kolonnas - satur kolonnu objektus, kolonnu īpašības un citus parametrus, kas ar kolonnām ir saistīti

2.2.2.4. Atkarības dekompozīcija

Atkarība

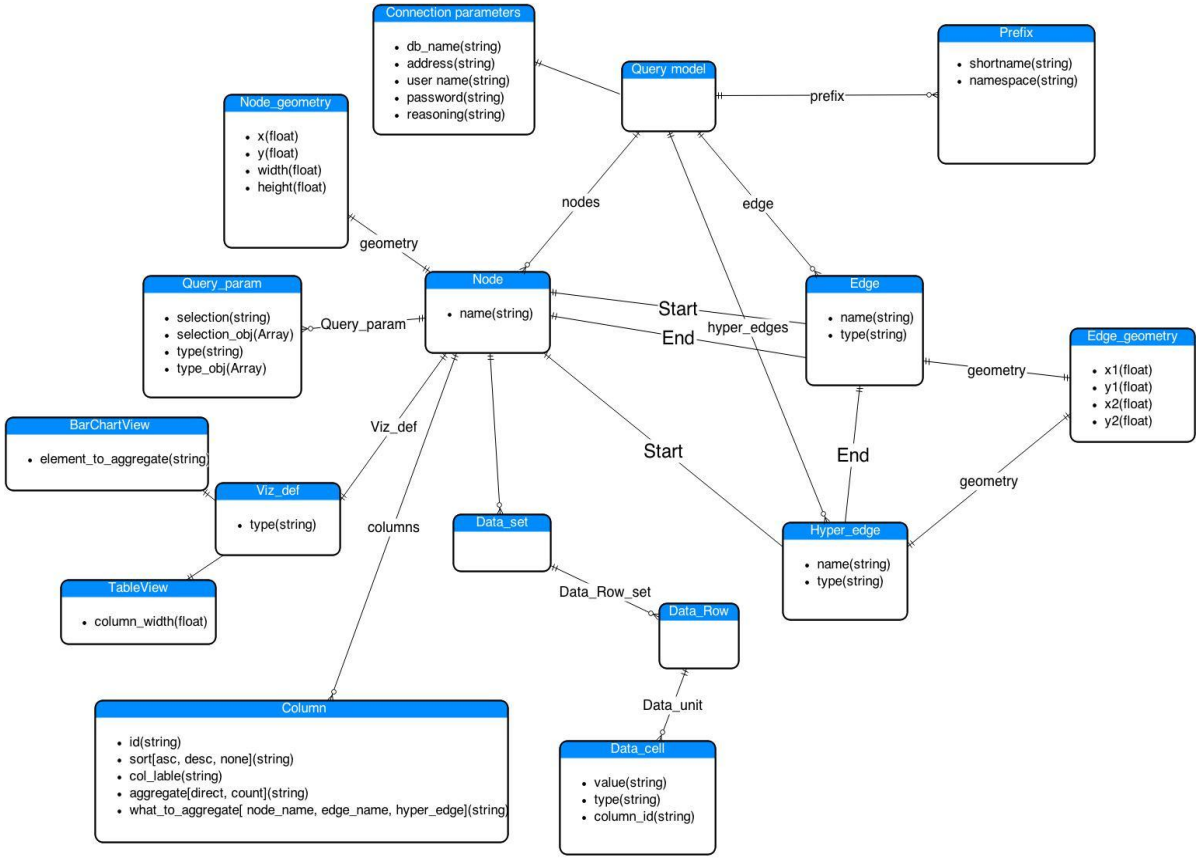
- vārds - atkarības unikāls nosaukums
- ienākošās līnijas - ar kādām hiperatkarībām atkarības ir saistītas
- ģeometrija - atkarības ģeometriskais novietojums
- sākums - elementa nosaukums, no kurienes atkarība sākas
- beigas - elementa nosaukums, no kurienes atkarība beidzas
- tips - nosaka vai elements ir atkarība vai hiperatkarība

2.2.2.5. Hiperatkarības dekompozīcija

Hiperatkarība

- vārds - atkarības unikāls nosaukums
- ienākošās līnijas - ar kādām hiperatkarībām atkarības ir saistītas
- ģeometrija - atkarības ģeometriskais novietojums
- sākums - elementa nosaukums, no kurienes atkarība sākas
- beigas - elementa nosaukums, no kurienes atkarība beidzas
- tips – nosaka, vai elements ir atkarība vai hiperatkarība

2.2. attēlā tiek attēlota Kverija modeļa struktūra.



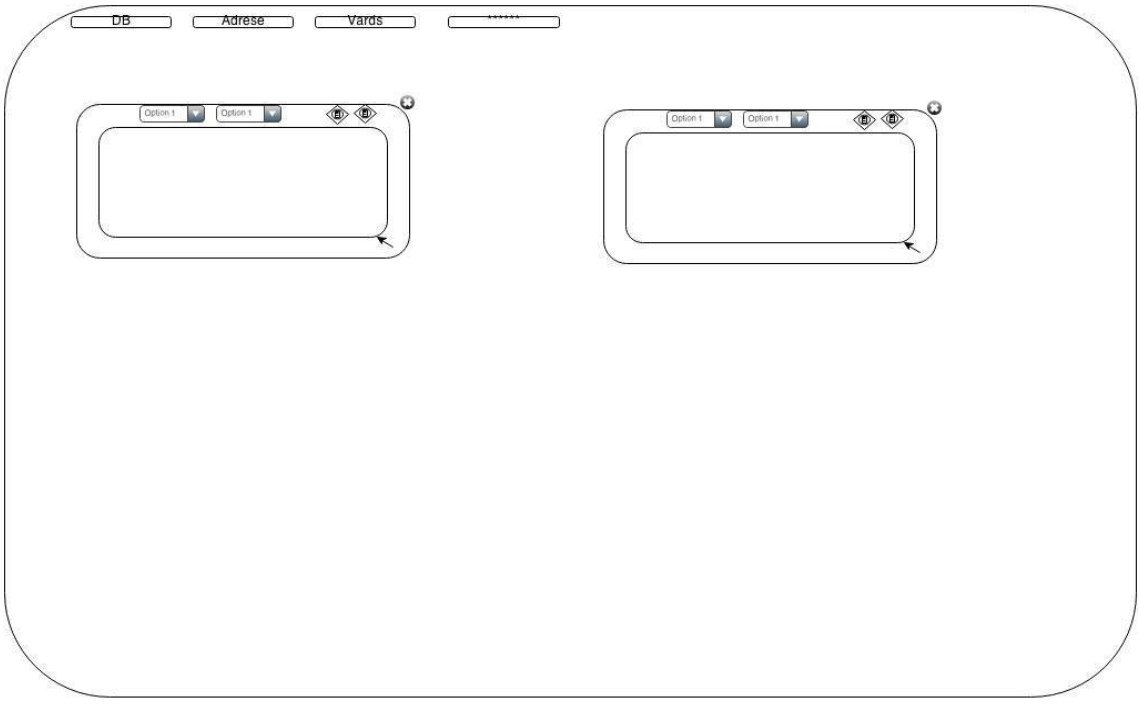
2.2. att. **Datu dekompozīcijas modelis.**

2.2.3. Lietotāju Saskarne

2.3. attēlā tiek parādīts lietotāja saskarnes logs, kur lietotājs var ievadīt datubāzes parametrus, lai sistēma varētu izveidot tiešsaisti ar servera datubāzi . Atlikušajā konteinerī tiks parādīti lietotāja izveidotie elementi.

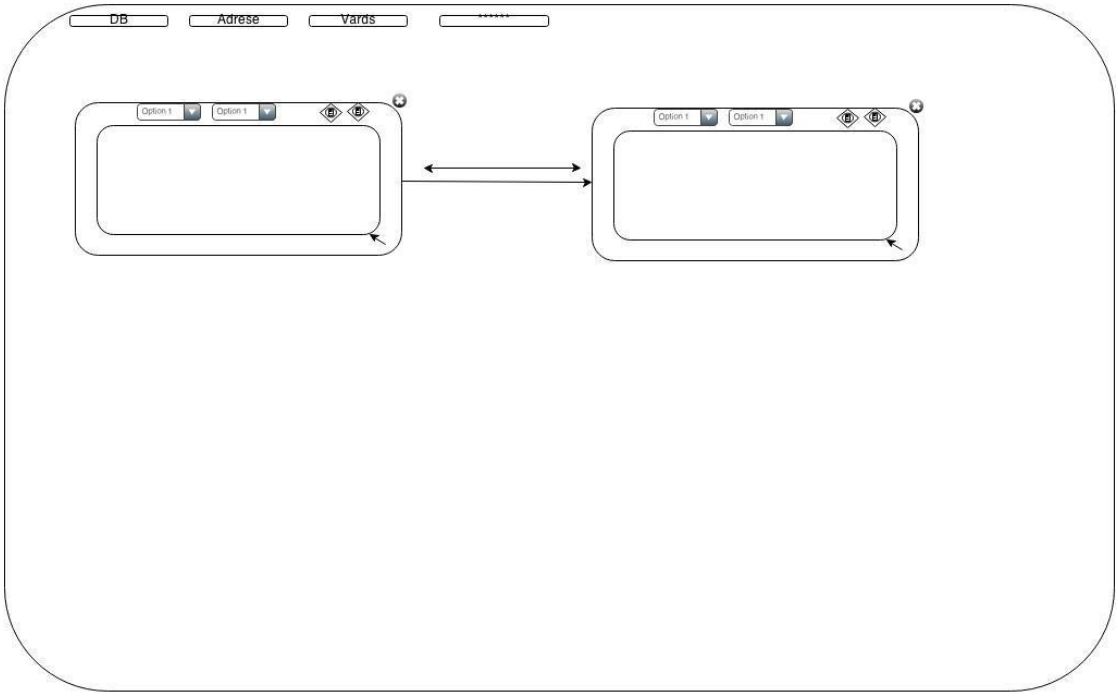
2.3. att. **Lietotāja saskarsmes logs**

2.4. attēlā ir parādīta vienkārša skice, kā izskatās konteineris, kad ir pievienotas divas virsotnes, bet nav atkarības, kas tās saistītu.



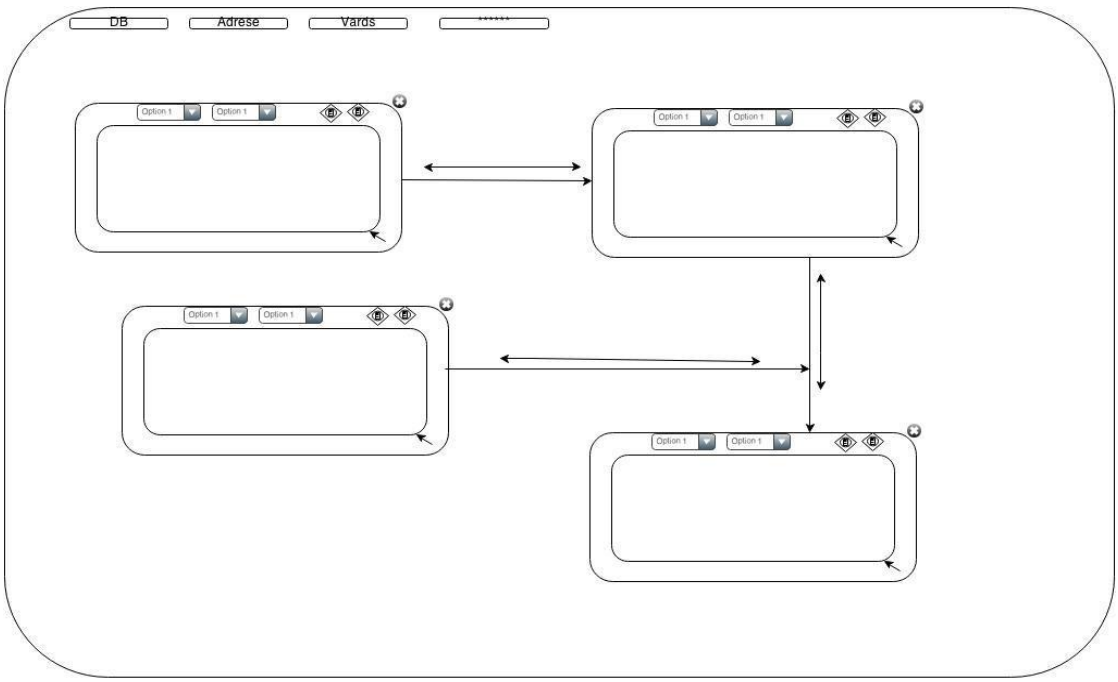
2.4. att. **Lietotāja saskarsmes logs ar izveidotām virsotnēm**

2.5. attēlā parādīts, kā atkarība savieno divas virsotnes. Virsotnes iekšējais konteineris atbild par datu parādīšanu, kas šeit nav attēloti.



2.5. att. **Lietotāja saskarsmes logs ar izveidotu vienkāršu struktūru**

2.6. attēlā parādīts kompleksas struktūras modelis. Modelis sastāv no četrām virsotnēm, divām atkarībām un vienas hiperatkarības. Hiperatkarība ar saistīto virsotni darbojas kā filtrs atkarībai, pie kā hiperatkarība ir pievienota.

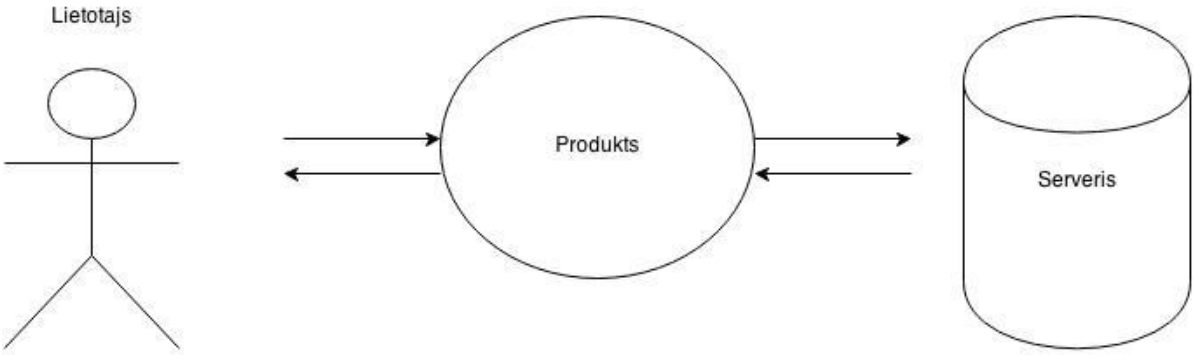


2.6. att. **Lietotāja saskarsmes logs ar izveidotu kompleksu struktūru**

2.3 Atkarību apraksts

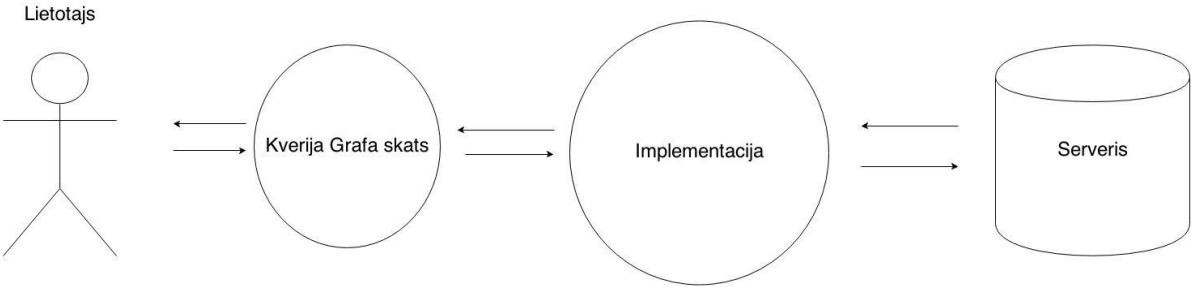
2.3.1. Datu plūsmu diagrammas

2.7. attēlā ir redzams vienkāršs modelis, kas parāda izstrādātā produkta (programmatūras) saistību ar lietotāju un serveri.



2.7. att. **0. Datu plūsmas līmenis**

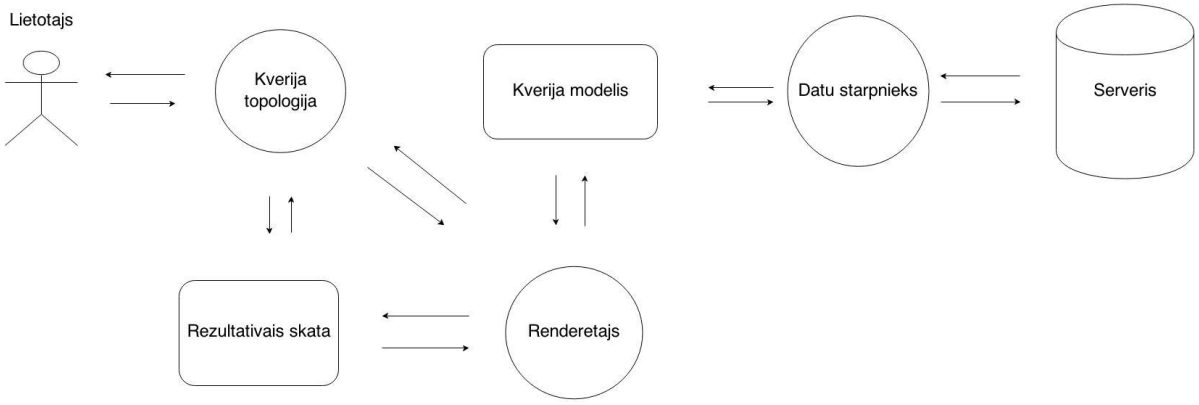
2.8. attēlā ir parādīts, ka 2.7. attēla produktam ir divi pamatelementi: Kverija grafa skats, kas atbild par lietotāja saskarni, un implementācija, kas atbild par datu apstrādi starp vizualizāciju un serveri.



2.8. att. **1. Datu plūsmas līmenis**

2.9. attēlā ir parādīta vēl detalizētāks datu plūsmas modelis, kur 2.8. attēla Kverija grafa skats tiek sadalīts Kverija topoloģijā, ko izveido lietotājs, pievienojot elementus topoloģijai un tos saistot, kā arī Rezultatīvais skats, kas parāda datus, kuri ir iegūti no servera.

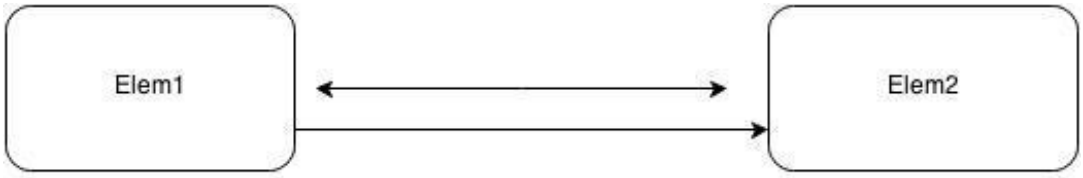
2.9. attēlā labajā pusē 2.8. attēla implementācija ir sadalīta trīs daļās: renderētājā, kas datus no Kverija modeļa apstrādā un padod rezultatīvajam skatam un Kverija topoloģijai, Kverija modelī, kas satur visus elementus, kas tiks parādīti Kverija topoloģijā (sīkāks apraksts par Kverija modeli 2.2.2. nodaļā), kā arī Datu starpniekā, kas iegūst datus no servera un caur Kverija modeli padod tos Rezultatīvajam skatam.



2.9. att. **2. Datu plūsmas līmenis**

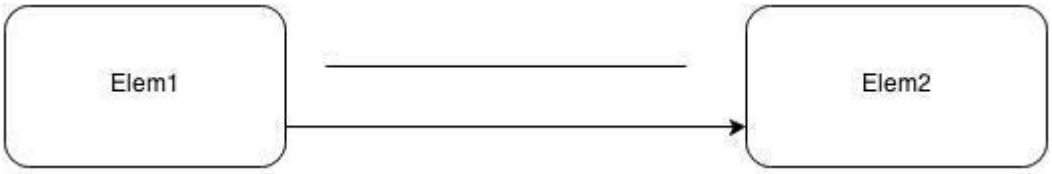
2.3.2. Atkarības veidi un to plūsmu atšķirības

2.10. attēlā parādīta abpusējā datu plūsma no Elem1 uz Elem2. Šis ir noklusētais atkarības stāvoklis pēc izveidošanas (apraksts par plūsmām 2.4.2.1. punktā).



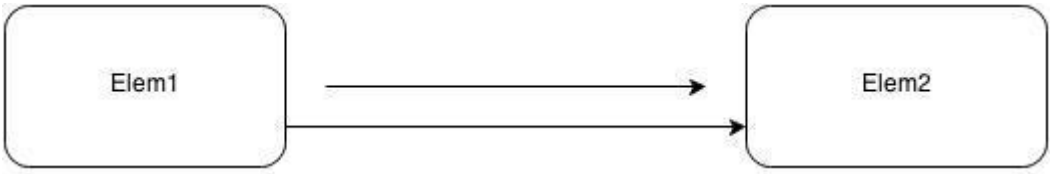
2.10. att. **1. Atkarības plūsmas veids**

2.11. attēlā parādīta bezplūsmas atkarība, datu kopas nesamazinās nevienā virsotnē (apraksts par plūsmām 2.4.2.1. punktā).



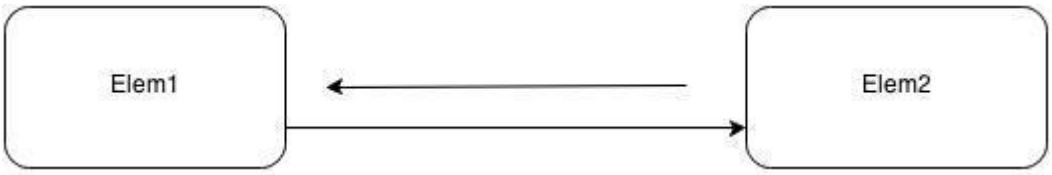
2.11. att. **2. Atkarības plūsmas veids**

2.12. attēlā parādīta vienvirziena plūsma no Elem1 uz Elem2, Elem2 tiek parādīti dati, kam ir saites ar Elem1 datu kopu (apraksts par plūsmām 2.4.2.1. punktā).



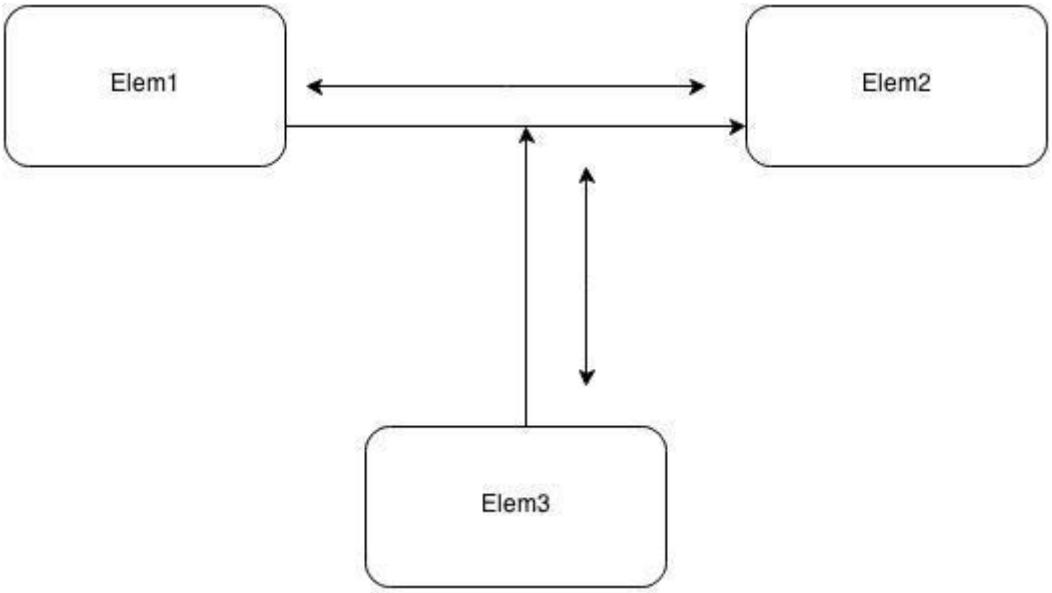
2.12. att. **3. Atkarības plūsmas veids**

2.13. attēlā parādīta inversā atkarības plūsma no Elem2 uz Elem1 (apraksts par plūsmām 2.4.2.1. nodaļā).



2.13. att. **4. Atkarības plūsmas veids**

2.14. attēlā parādīts hiperatkarības elementa izmatošanas veids. Hiperatkarības virsotnē var izvēlēties vērtības, kas atbild par atkarības plūsmas vērtību, kas saista Elem1 un Elem2. Hiperatkarība saista vienu virsotni un vienu atkarību.



2.14. att. **Hiperatkarības modelis**

2.4. Vienumu detalizēts projektējums

2.4.1. Virsotnes skats (NV.1)

2.4.1. 1. Virsotnes izveide

Identifikators: make_node
Apraksts

Funkcija izveido virsotni pēc 2.2.2.3. izveidota šablona un virsotnes datus pievieno Kveriju Grafam.
Ievade
Koordinātas Kverija grafu skata konteinerī, no kurienes sāks zīmēt virsotni.
Apstrāde
1. Padod x un y koordinātas un Kverija Grafu, 2. Funkcija iziet cauri visām iepriekš izveidotajām virsotnēm Kveriju Grafā un izveido unikālu identifikatoru jaunajai virsotnei 3. Virsotnes ģeometrijai pievieno koordinātas, no kurienes virsotne tiks sākta zīmēt 4. Virsotnes vārdam tiek pielīdzināts iepriekš izveidotais identifikators 5. Jaunizveidoto virsotni pievieno Kveriju Grafam 6. Atjauno visus skatus, lai tiek uzzīmēta virsotne un tajā parādās pareizie kverija dati
Izvade
Nav

2.4.1.2 Virsotnes dzēšana

Identifikators: delete_node
Apraksts
Funkcija izdzēš visus virsotnes datus Kverija Grafā no visiem elementiem, kas ir saistīti ar virsotni, pēc pogas nospiešanas, kas inicializē šo funkciju. Izdzēšot virsotni tiek izdzēstas visas ar to saistītās atkarības, jo no vienpusējām atkarībām nav jēgas. Funkcija tiek piesaistīta viegli pamanāmai pogai
Ievade
Tiek inicializēta notikumfunkcija, kas saņem virsotnes datus un Kverija Grafu
Apstrāde
1. No virsotnes datiem tiek uz visiem elementiem, kas ir saistīti ar šo virsotni 2. Savāc visas atkarības sarakstā, kas ir saistītas ar virsotni 3. Izsauc 2.4.2.1 punktā aprakstīto funkciju uz visām saistītajām atkarībām 4. Izdzēš visu informāciju par virsotni no Kveriju Grafa 5. Inicializē rezultatīvā un strukturālā skata atjaunošanu
Izvade
Nav

2.4.1.3 Virsotnes pārbīde

Identifikators: drag_node
Apraksts
Funkcija izmaina virsotnes ģeometrijas sākumpunktus.

Ievade
Virsotnes dati, jaunās koordinātas(uz x un y asīm)
Apstrāde
1. Saņem virsotnes elementu no Kveriju Grafa 2. Izvēlas virsotnes ģeometrijas sadaļu 3. Virsotnes ģeometrijā ievieto jaunās vietas koordinātas 4. Izsauc funkciju, kas atjauno strukturālo skatu
Izvade
Nav

2.4.1.4 Virsotnes izmēra izmaiņa

Identifikators: resize_node
Apraksts
Lietotājam nospiežot uz virsotnes noteiktas sadaļas tiek izsaukt funkcija, kas sāk izmainīt virsotnes ģeometrijas platumu un augstumu pēc lietotāja peles kustībām, funkcijas beidzas, kad lietotājs atlaiž taustiņu, kas inicializēja šo funkciju. Virsotnei ir minimālais platums un augstums, kuru lietotājs nespēj pārsniegt.
Ievade
Jaunā izmēra platums un augstums, virsotnes dati no Kverija Grafa
Apstrāde
1. Iegūst virsotnes datus no Kveriju Grafa 2. Izvēlas virsotnes ģeometrijas sadaļu 3. Vecos izmērus aizstāj ar jaunajiem 4. Izsauc funkciju, kas atjauno strukturālo skatu(...)
Izvade
Nav

2.4.1.5 Virsotnes skata maiņa

Identifikators: toggle_node_view
Apraksts
Mainot virsotnes skatu, dati, kas iegūti no servera tiek attēloti citādāk. Mainot skatus, varam iegūt lielāku izpratni par datiem, paskatoties uz tiem no cita skatapunkta. Skata maiņa tiek inicializēta pēc pogas nospiešanas.
Ievade
Virsotnes dati no Kveriju Grafa
Apstrāde
1. Virsotnes datos izmaina aktīvā vizualizācijas tipa datus 2. Izsauc funkciju, kas pārģenerē kverijus 3. Saņem datus no servera

4. Apstrādā datus specifiski izvēlētajam skatam 5. Atjauno rezultatīvo skatu, lai parādītos jaunais skats interneta pārlūkā
Kļūdu paziņojumi
„Error…..”
Izvade
Nav

2.4.1.6 Virsotnes tipa izvēle

Identifikators: select_type
Apraksts
Tiek izvēlēts tips, kas filtrēs rezultatīvajā skatā parādītos datus. Tipi tiek iegūti no rezultatīvā skata datiem. Tipus var izvēlēties no *combobox,* kur visi tipi tiek glabāti
Ievade
Virsotnes kverijs
Apstrāde
1. Modificējam padoto kveriju uz tādu kveriju, kas atdos visus tipus par datiem, ko atdeva sākotnējais kverijs 2. Iegūst tipu kopu, ko saglabājam sarakstā 3. Sarakstu sasaistām ar *combobox*, lai katru reizi, kad izmainās tipu kopa, nav jāizdzēš elements, kurā turam tipus 4. Ja lietotājs izvēlās kādu no tipiem, tad šo tipu pieglabājam pie kverija parametriem 5. Izsauc funkciju, kas atjauno rezultatīvo skatu
Kļūdu paziņojumi
„*Error...*” rezultatīvajā skatā.

2.4.2. Atkarības skats (EV.2)

2.4.2.1. Atkarības izveidošana

Identifikators: make_edge
Apraksts
Izveido saiti starp divām virsotnēm vai virsotni un atkarību, kas piedalās kveriju veidošana. Atkarībai ir divi virzieni un četras plūsmas. Virziens nosaka, no kura elementa iziet atkarība un uz kuru elementu tā iet. Piemēram, ja pirmo elementu uztveram kā personu Paula un otro elementu kā Jānis, tad atkarība ir kā saite ar nosaukumu māte. Paula ir māte Jānim, tāds ir virziens, bet Jānis nav māte Paulai, uz pretējo pusi ir pilnīgi cita atkarība kā Jānis ir bērns Paulai. Ja ir datubāze par ģimeni un mēs esam izveidojuši divas tabulas un atkarību māte

no tabulas A uz tabulu B. Ar **abpusējo plūsmu** Tabulā A parādās visi bērni, kam ir mātes, un tabulā parādās visas mātes kam ir bērni. Ir arī **bezplūsmu atkarība**, kas nesamazina datus nevienā virsotnē, bet šo atkarību var izmantot kolonnu izveidē. Tad ir divas **vienvirziena plūsmas**, kas parāda, kādi bērni vai kādas mātes atrodas visā datubāzē. Cikls iet no abpusējās uz bezplūsmu un tad pa abām vienvirziena plūsmām atkal uz abpusējo plūsmu, ja uzspiež uz plūsmas līnijas. Šīs plūsmas var mainīt ar 2.2.1.4. punkta trešās funkcijas palīdzību (*toggle_edge_flow*) Atkarību starp vienu virsotni un atkarību sauc par hiperatkarību.
Ievade
Tiek padoti divi elementi un Kveriju Grafs
Apstrāde
1. Pārbauda, vai abi elementi ir virsotnes vai viena virsotne un viena atkarība. Ja nav tā, funkcija beidz darbu 2. Izveido atkarības unikālu identifikatoru 3. Identifikatoru izmanto, lai izveidotu unikālu atkarības elementu 4. Pirmais padotais elements kļūst par atkarības sākumu un otrais padotais par atkarības beigām, jo atkarība ir orientēta 5. Izveidoto atkarību ievieto Kveriju Grafā 6. Atjauno strukturālo skatu, lai parādās jaunā atkarība
Izvade
Nav

2.4.2.1 Atkarības izdzēšana

Identifikators: delete_edge
Apraksts
Izdzēš visu informāciju par atkarību no Kveriju Grafa
Ievade
Atkarības nosaukums un Kveriju Grafs
Apstrāde
1. No atkarības galapunktu elementiem tiek izdzēsta jebkāda informācija par atkarību 2. Atkarības dati tiek izņemti no Kveriju Grafa 3. Izsauc funkcijas, kas atjauno strukturālo un rezultatīvo skatu
Izvade
Nav

2.4.3. Datu atlases apstrāde (DSH.4)

2.4.3.1. Vērtību atlasīšana

Identifikators: data_selection
Apraksts
Atlasa datu apakškopu no rezultatīvā skata datiem, kas darbojas kā iekšējais un ārējais ierobežojums kveriju veidošanā
Ievade
Tiek padota datu apakškopa no virsotnes rezultatīvā skata
Apstrāde
1. Lietotājs spēj atlasīt datus no rezultatīvā skata, uz tiem uzspiežot. Klasiskais ar *Shift* taustiņa turēšanu tiek atlasīti visi dati, kas ir starpā starp jau izvēlēto un jauno, bet, nospiežot kreiso *Ctrl* taustiņu, var izvēlēties pa vienam kurus datus atlasīt 2. Šos datus saglabājam kverija parametros, kas atrodas virsotnē 3. Izsauc funkciju, kas pārģenerē kverijus, jo tagad *aktīvā datu kopa* šajā virsotnē ir izmainījusies un kverijs to ņem vērā

2.4.4. Kveriju Grafa skati(QGV.3)

2.4.4.1. Strukturālā skata atsvaidzināšana

Identifikators: render_graph_view
Apraksts
Atjauno strukturālo skatu, kur atrodas vizuālie elementi kā virsotnes, atkarības un hiperatkarības. Kā arī visi elementi, kas ir pamatelementu ietvaros, kas ir atbildīgi par dažādu citu funkciju izpildi.
Ievade
Saņem Kveriju Grafu
Apstrāde
1. Iziet cauri katram Kveriju Grafa elementam 2. Apskatās, vai tāds elements jau eksistē a) ja eksistē – skatās, vai kaut kas ir mainījies tajā kopš iepriekšējās reizes; b) ja neeksistē, tad izveido to ar visiem tā atribūtiem 3. Izveido nepieciešamos atribūtus elementiem kā tipu *combobox* un izmēra maiņas lauks virsotnēm 4. Attēlo tos visus grafiski ar bibliotēkas *d3* palīdzību
Izvade
Nav

2.4.4.2. Rezultatīvā skata atsvaidzināšana

Identifikators: render_node_view

Apraksts
Atjauno rezultatīvo skatu visās virsotnēs, datiem, kas ir iegūti no servera, rezultatīvais skats tiek atjaunots pēc katras darbības ar rezultatīvā skata datiem un elementu pievienošanas, piemēram, tipu izvēles maiņa, atlases izveide, atkarības izveidošanas un dzēšanas, virsotnes izveidošanas un dzēšanas.
Ievade
Kverija grafs
Apstrāde
1. Katrai virsotnei tiek izsaukta funkcija, kas pārģenerē kveriju 2. Saņemtie kveriji tiek padoti kā parametrs, sazinoties ar serveri 3. Saņem datus no Servera 4. Dati tiek iedoti katrai virsotnei 5. Katra virsotne apstrādā datus pēc iestatītā skata 6. Skats arī nodrošina atlases elementu iezīmēšanu, lai netiktu pārprasts, kuri elementi ir aktīvajā atlasē

2.4.5. Kverija Ģenerēšana (QG.5)

Identifikators: query_generation
Apraksts
Apstrādājot strukturālā skata topoloģiju, no vienas virsotnes skatapunkta tiek uzģenerēts kverijs, kas tiks izmantots, lai iegūtu datus no servera, kas tiks ievietoti virsotnēs. Kveriju ģenerēšanas laikā tiek ņemti vērā gan lokālie ierobežojumi, virsotnes tipa izvēle un datu atlase, kā arī ārējie ierobežojumi, atkarības, kas ir saistītas ar virsotni un citu virsotņu lokālie ierobežojumi, kas arī ierobežo virsotni, kam tiek veikts kverijs
Ievade
Tiek padots Kverija Grafs un virsotne
Apstrāde
1. Vispirms tiek sagatavots kverija karkass, kas sastāv no galvas, kurā atrodas tabulas kolonnu vārdi, kurus būs jāparāda, un ķermeņa, kurā atrodas visi ierobežojumi un filtri 2. Savāc lokālo ierobežojumus, kas ir atrodami virsotnes kveriju parametros, tie ir tipa izvēle un datu atlases saraksts 3. Tad savāc visus ārējos ierobežojumus, no elementiem, kas ir saistīti ar virsotni 4. Visus ierobežojumus apstrādā pēc izvēlētā kverija sintakses 5. Saliek visus pārējos elementus kverijā, kā kārtošanu un citus 6. Atdod gatavo kveriju
Izvade
Kverijs *string* formā.

Konekcijas izveide

Identifikators: Connection_data_input

Apraksts
Tiek izveidota konekcija ar datubāzi, kuras datus parādīs programmatūra. Interneta pārlūkā jābūt ievades laukiem, kas saņems konekcijas parametrus.
Ievade
Datubāzes vārds (*string*), URL adrese (*string*), autentifikācijas vārds (*string*) un autentifikācijas parole (*string*)
Apstrāde
1. Lietotājs ievada konekcijas parametrus attiecīgajos ievades laukos 2. Programma apstrādā un saglabā tos Kverija modelī 3. Katru reizi tiek izmantoti šie parametri, kad tiek izsaukta 2.4.4.2. punktā aprakstītā funkcija, kas atjauno rezultatīvos skatus. 4. Programmatūra ar šīs funkcijas palīdzību spēj sazināties ar izvēlēto datubāzi
Komentāri
Programmatūra nepārbauda ievade pareizību.

2.4.6. Kolonnu apstrādātājs (CH.6)

2.4.6.1. Kolonnas izveidošana

Identifikators: create_column
Apraksts
Izveido kolonnu, ko pievieno virsotnes datiem. Kolonnas izveide tiek inicializēta cauri formu, ko aizpilda lietotājs. Pie formas tiek klāt nospiežot formas pogu. Forma sastāv no vairākiem laukiem.
Ievade)
Formā ievada kā kolona būs sakārtota (*sort*) (*combobox*), kolonnas nosaukums (*string*), ko kolonna parāda (*string*) un pēc kā kolonna tiek agregēta (*combobox*)
Apstrāde
1. Lietotājs atver formas logu 2. Tiek aizpildīti nepieciešamie lauki 3. Datus apstrādā un tos ievada kolonnas šablonā 4. Jaunizveidoto kolonnu pievieno virsotnei Kveriju Grafā 5. Izsauc funkciju, kas atjauno rezultatīvos skatus katrā virsotnē

2.4.6.2. Kolonnu dzēšana

Identifikators: delete_column
Apraksts
Kolonna tiek izņemta no Kverija Grafa. Šo notikumu var inicializēt no kolonnu formas loga, tur jāizveido poga, kas izdzēš izvēlēto kolonnu.

Ievade
Pogas nospiešana, kas inicializē kolonnas izdzēšanu
Apstrāde
1. Atveros formas logs, kas atbild par kolonnu 2. Nospiežot uz kolonnas izdzēšanas pogu, attiecīgā kolonna tiek izdzēsta 3. Izsauc funkciju, kas atjauno visus rezultatīvos skatus

2.4.6.3. Kolonnas rediģēšana

Identifikators: Render column
Apraksts
Kolonnas īpašības ir maināmas pat pēc kolonnas izveides. Pie tām var tikt cauri kolonnas īpašību formas. Formu aizpilda lietotājs
Ievade
Saņem jaunās kolonnas īpašības no formas, ko lietotājs ir aizpildījis
Apstrāde
1. Saņem formas ievadītos datus 2. Apskata kuri dati ir izmainījušies 3. Izmaina kolonnas datus Kveriju Grafā 4. Izsauc funkciju, kas atjauno visus rezultatīvos skatus
Kļūdu paziņojumi
Nav?
Komentāri
Šobrīd ievadītie dati netiek pārbaudīti un validēti, tādēļ, ievadot nepareizus datus vai neaizpildod dažus laukus, programma var nestrādāt.

3. TESTĒŠANAS DOKUMENTĀCIJA

3.1. Testēšanas plāns

Testēšanas plāns sastāv no visu vienumu testēšanas. Veicot testēšanu ir jāpārliecinās, ka ir ievērotas visas prastības, kas tika izvirzītas programmatūras prasību specifikācijā [7].

Testēšanā tiks pielietoti Latvijas testēšanas standarti [3].

Testēšanas procesā tiks veiktas tālāk minētās pārbaudes:

- vai vienuma izpilda tikai un vienīgi tam paredzēto darbu;
- vai vienums nepieciešamības gadījumā korekti izsauc cita vienuma darbību;

Testēšanas procesā netika iekļautas ātrdarbības un noslodzes pārbaudes. Tāpat netika ņemtas vērā kļūdas, ja tās ir RDF grafā. Tā kā šī programmatūra ir prototips, tad visas iespējamās kļūdas netika apskatītas.

Testēšana tika veikta, izmantojot pelēkās kastes metodi – testpiemēri tiek veidoti, ņemot vērā vienumu uzbūvi, un testēšana notiek no lietotāja skata punkta.

Testēšanu veica autors 2014. gada aprīlī un maijā.

Testēšana tika veikta Google Chrome interneta pārlūkā, kurā programmatūra ir izstrādāta.

Testējot saskarne ar programmu notiek ar peles kreiso taustiņu.

3.2. Testēšanas žurnāls

Testēšanas žurnālā iekļautas astoņas sadaļas, katram vienumam sava sadaļa :

- Virsotnes skats
- Atkarības skats
- Kverija grafa skats
- Skatu atlases apstrādātājs
- Kveriju ģenerētājs
- Kolonnu apstrādātājs
- Tabulas skats
- Stabiņu diagrammas skats

3.2.1. Virsotnes skats (NV.1)

3.1. tabula.

Vienuma NV.1 testēšanas rezultāti

Testa nr.	Testa apraksts	Izpildās	Komentārs
1.	Izveidot jaunu virsotni, kas būs redzama lietotājam : Lietotājs noklikšķina uz interneta pārlūka konteinera.	Jā	Nav
2.	Izveidot vairākas jaunas virsotnes, kas būs redzamas lietotājam : Lietotājs vairākas reizes noklikšķina uz interneta pārlūka konteinera dažādās vietās, bet ne uz izveidotajām virsotnēm.	Jā	Ja virsotnes atrodas pārāk tuvu viena otrai, tad tās daļēji pārklājas.
3.	Izdzēst virsotni no Kverija modeļa, lai tā vairs netiktu attēlota Kveriju Grafa skatā: Lietotājs noklikšķina uz virsotnes izdzēšanas pogas, kas atrodas virsotnes augšējā labajā stūrī.	Jā	Nav
4.	Izdzēst virsotni, ko referencē kādas citas virsotnes kolonna, no Kverija modeļa, lai tā vairs netiktu attēlota Kveriju Grafa skatā: Lietotājs noklikšķina uz virsotnes izdzēšanas pogas.	Nē	Kolonna netiek izdzēsta un tā nograuj sistēmu. **Izlabots** - kolonnas, kas ir saistītas ar izdzēšamo virsotni, arī tiek izdzēstas. Ja virsotni referencē kāda kolonna, tad tiek izsaukts CH.6 vienuma 2.4.8.2. aprakstītā kolonnas dzēšanas funkcija.
5.	Pārvietot virsotni pa interneta pārlūka konteineri: Lietotājs tur kursoru uz virsotnes augšējās ārējās malas (atrodas virs rezultatīvā skata datiem) un velk virsotni pa konteineri.	Jā	Nav
6.	Izmainīt virsotnes izmērus uz mazākām vērtībām: Lietotājs tur kursoru uz izmēra izmaiņu apgabala, kas atrodas virsotnes labajā apakšējā stūrī, un virza to uz augšējo kreiso stūri, kā arī pa kreisi vai uz augšu.	Jā	Nav
7.	Izmainīt virsotnes izmērus uz mazākām vērtībām, pārbaudot, lai virsotne nepaliek mazāka par noteikto minimumu: Lietotājs tur kursoru uz samazināšanas apgabala, kas atrodas virsotnes labajā	Jā	Nav

	apakšējā stūrī, un virza to uz augšējo kreiso stūri līdz process apstājas.		
8.	Izmainīt virsotnes izmērus uz lielākām vērtībām: Lietotājs tur kursoru uz izmēra izmaiņu apgabala, kas atrodas virsotnes labajā apakšējā stūrī, un virza to prom no augšējā kreisā stūra, kā arī pa labi vai uz leju.	Nē	Virsotni var palielināt bezgalīgi, **Izlabots** - virsotnei tagad ir iespējamie lielākie lielumi.
9.	Izvēlēties tipu, kas darbosies kā filtrs uz Rezultatīvā skata datiem: Lietotājs izvēlas vērtību no pirmās izvēlnes, kas atrodas virsotnes augšējā apgabalā kreisajā pusē.	Nē	Pie dažiem tipiem tipu izvēle pazūd, ja ir tikai viena virsotne, kas nav saistīta ar citām virsotnēm. **Izlabots** - problēma tika novērsta, uzlabojot kveriju ģenerēšanas funkciju.

3.2.2. Atkarības skats (EV.2)

3.2. tabula.

Vienuma EV.2 testēšanas rezultāti

Testa nr.	**Testa apraksts**	**Izpildās**	**Komentārs**
1.	Izveidot atkarību starp divām virsotnēm: Lietotājs, turot nospiestu S*hift* taustiņu, izvēlas pirmo virsotni un pēc tam otro virsotni, kurai izveidot atkarību.	Nē	Lietotājs var izveidot vairākas atkarības starp divām vienādām virsotnēm. **Izlabots** - starp divām virsotnēm iespējama tikai viena atkarība.
2.	Izveidot atkarību starp virsotni un atkarību: Lietotājs, turot nospiestu S*hift* taustiņu, izvēlas pirmo elementu no virsotnes vai atkarības un pēc tam atlikušo elementu.	Nē	Lietotājs var izveidot vairākas atkarības starp minētajiem diviem elementiem, kā arī atkarības uz hiperatkarībām Izlabots - vairs nav iespējam izveidot starp diviem elementiem vairākas vienādas attiecības. Uz hiperatkarību nevar izveidot atkarību, jo tas nav specificēts gadījums.
3.	Izdzēst atkarību: Lietotājs novieto kursoru uz atkarības izdzēšanas lauka, kas atrodas tuvu	Nē	Atkarības tiek izdzēstas tikai tad, kad saistītā virsotne tiek

	atkarības viduspunktam, un noklikšķina uz tā.		izdzēsta. **Izlabots** - atkarības var izdzēst neizdzēšot nevienu virsotni.
4.	Mainīt atkarības plūsmas virzienu: Lietotājs novieto kursoru uz atkarības plūsmas virziena bultas, kas ir šaurāka par atkarības līniju, un noklikšķina uz tās.	Jā	Cikls mainās pēc 2.4.2.1. apakšpunktā norādītā apraksta.

3.2.3. Kverija grafa skats (QGV.3)

3.3. tabula.

Vienuma QGV.3 testēšanas rezultāti

Testa nr.	Testa apraksts	Izpildās	Komentārs
1.	Atjaunot rezultatīvo skatu pēc jaunas virsotnes izveidošanas: Notiek automātiski bez lietotāja palīdzības.	Jā	Nav
2.	Atjaunot rezultatīvo skatu pēc jaunas atkarības izveidošanas: Notiek automātiski bez lietotāja palīdzības.	Jā	Nav
3.	Atjaunot rezultatīvo skatu pēc virsotnes izdzēšanas: Notiek automātiski bez lietotāja palīdzības.	Jā	Nav
4.	Atjaunot rezultatīvo skatu pēc atkarības izdzēšanas: Notiek automātiski bez lietotāja palīdzības.	Jā	Nav
5.	Atjaunot rezultatīvo skatu pēc jaunas atlases izveidošanas virsotnē: Notiek automātiski bez lietotāja palīdzības.	Jā	Nav
6.	Atjaunot rezultatīvo skatu pēc atkarības plūsmas virziena maiņas: Notiek automātiski bez lietotāja palīdzības.	Jā	Nav
7.	Atjaunot rezultatīvo skatu pēc tipa izvēles virsotnē: Notiek automātiski bez lietotāja palīdzības.	Jā	Nav
8.	Atjaunot rezultatīvo skatu pēc kolonnas pievienošanas: Notiek automātiski bez lietotāja palīdzības.	Jā	Nav
9.	Atjaunot rezultatīvo skatu pēc kolonnas izdzēšanas:	Jā	Nav

	Notiek automātiski bez lietotāja palīdzības.		
10.	Atjaunot rezultatīvo skatu pēc kolonnas īpašību izmaiņas: Notiek automātiski bez lietotāja palīdzības.	Nē	Ir konstatētas atsevišķas problēmas. Piemēram, ja visi lauki netiek aizpildīti.
11.	Atjaunot strukturālo skatu pēc jaunas virsotnes izveidošanas: Notiek automātiski bez lietotāja palīdzības.	Jā	Nav
12.	Atjaunot strukturālo skatu pēc jaunas atkarības izveidošanas: Notiek automātiski bez lietotāja palīdzības.	Jā	Nav
13.	Atjaunot strukturālo skatu pēc virsotnes izdzēšanas: Notiek automātiski bez lietotāja palīdzības.	Jā	Nav
14.	Atjaunot strukturālo skatu pēc atkarības izdzēšanas: Notiek automātiski bez lietotāja palīdzības.	Nē	Problēma atsaucas uz 5.1. tabulas ceturto ierakstu. **Izlabots**
15.	Atjaunot strukturālo skatu pēc virsotnes pārbīdes: Notiek automātiski bez lietotāja palīdzības.	Jā	Nav
16.	Atjaunot strukturālo skatu pēc virsotnes izmēru izmaiņām: Notiek automātiski bez lietotāja palīdzības.	Jā	Nav

3.2.4. Datu atlases apstrādātājs (DSH.4)

3.4. tabula.

Vienuma DSH.4 testēšanas rezultāti

Testa nr.	**Testa apraksts**	**Izpildās**	**Komentārs**
1.	Izveidot jaunu atlasi no virsotnes rezultatīvā skata datiem: Lietotājs izvēlas datu vienības, uzklikšķinot ar kursoru uz tām, vairākas vienības var atlasīt, turot nospiestu *Shift* vai *Ctrl* taustiņu.	Jā	Ir kolīzija starp atlases izveidi un atkarības izveidošanu, jo abos gadījumos izmanto S*hift* taustiņu
2.	Izdzēst atlasi: Lietotājs ar kursoru uzklikšķina uz atlases dzēšanas pogas, kas atrodas virsotnes apakšējā sadaļā, otrā poga no kreisās puses.	Jā	Nav
3.	Pārliecināties vai izveidotā atlase saglabājas, ja datu kārtība tiek mainīta:	Jā	Ja mainās datu kārtība, programma

	Lietotājs veic dažādas darbības, kas maina rezultatīvā skata datu kārtību.		atceras atlasītos elementus, kamēr tie ir rezultatīvajā skatā. Ja atlase vairs nav rezultatīvajā skatā vai rezultatīvais skats ir tukšs, tad atlase tiek izdzēsta.

3.2.5. Kveriju ģenerētājs (QG.5)

3.5. tabula.

Vienuma QG.5 testēšanas rezultāti

Testa nr.	Testa apraksts	Izpildās	Komentārs
1.	Izveidot kveriju attiecīgi virsotnes ierobežojumiem, ar kuru iegūt datus no servera, un ievietot virsotnes rezultatīvajā skatā: Notiek automātiski bez lietotāja palīdzības.	Jā	Šis tests tiek izsaukts katru reizi, kad tiek iedarbināta vienuma QGV.3. rezultatīvā skata atjaunināšanas funkcija. Kveriju ģenerēšana no autora puses tiek uzskatīta par melnās kastes modeli.

3.2.6. Kolonnu apstrādātājs (CH.6)

3.6. tabula.

Vienuma CH.6 testēšanas rezultāti

Testa nr.	Testa apraksts	Izpildās	Komentārs
1.	Atvērt kolonnu apstrādes logu: Lietotājs ar kursoru noklikšķina uz pogas, kas atrodas virsotnes augšējajā sadaļā, otrā poga no labās puses.	Jā	Nav
2.	Izveidot kolonnu: Lietotājs atver kolonnu apstrādes logu, apstrādes logā ieklikšķina dzeltenajā laukā. Sistēma izveido kolonnu, kuru pieliek pie virsotnes.	Jā	Nav
3.	Izdzēst kolonnu: Lietotājs atver kolonnu apstrādes logu, noklikšķina uz kolonas izdzēšanas pogas.	Jā	Nav

4.	Izmainīt kolonnas īpašības: Lietotājs atver kolonnu apstrādes logu, ieraksta kolonnu īpašību laukos izvēlētos parametrus.	Nē	Nedarbojas izvēles ievadlauki. **Izlabots** - tagad īpašības var izmainīt. Pirmais ievadlauks ir kolonnas agregācijas tips, otrais ir kārtošanas secība, trešais ir kolonnas vārds, ceturtais ir šīs kolonnas īpašības vārds.

3.2.7. Tabulas skats (TV.7)

3.7. tabula.

Vienuma TV.7 testēšanas rezultāti

Testa nr.	Testa apraksts	Izpildās	Komentārs
1.	Inicializēt tabulu skatu: Ja virsotnē ir parādīta stabiņu diagramma, lietotājs noklikšķina virsotnes augšējā daļā uz pirmās pogas no labās puses.	Jā	Rezultatīvais skats tiek nomainīts no stabiņu diagrammas skata uz tabulu skatu.
2.	Veiksmīgi nomainīt izmatoto datubāzi: Lietotājs ieraksta datubāzes parametrus datubāzes konekcija ievades laukos, kas atrodas interneta pārlūka konteinera pašā augšējajā sadaļā.	Jā	Visi ievades dati ir atbilstoši eksistējošai datubāzei.

3.2.8. Stabiņu diagrammas skats (BCV.8)

3.8. tabula.

Vienuma BCV.8 testēšanas rezultāti

Testa nr.	Testa apraksts	Izpildās	Komentārs
1.	Inicializēt stabiņu diagrammas skatu: Ja virsotnē ir parādīta tabula, lietotājs noklikšķina virsotnes augšējā daļā uz pirmās pogas no labās puses. Poga darbojas kā 2.4.1.5. apakšpunktā aprakstītā funkcija.	Nē	Nav iespēja izvēlēties kolonnu, kuru parādīt stabiņu diagrammas skatā. **Izlabots** - pielikts izvēlnes logs virsotnes augšējajā sadaļā, kurā var izvēlēties kolonnu,

			kuru parādīt stabiņu diagrammas skatā.
2.	Izvēlēties kolonnu, kuru parādīt stabiņu diagrammas skatā: Lietotājs izvēlnes logā, kas atrodas virsotnes augšējā sadaļā labajā pusē, ieklikšķina kolonnas vārdu, kuru parādīt.	Jā	Izvēlnes logā parādās tikai kolonnas ar agregācijas tipu *count.* Izvēlne neinicializē skata maiņu, tas jāveic lietotājam, nospiežot skata maiņas pogu.
3.	Atlasīt datus no stabiņu diagrammas skata kā aktīvo atlasi: Lietotājs ar kursora vilkšanu ārpus stabiņiem izvēlas, kurus atlasīt.	Jā	Atlasi ir iespējams izveidot tikai no blakus esošiem stabiņiem.

3.3 Testēšanas kopsavilkums

Autors ir testējis visus astoņus vienumus.

Programmatūras testēšanas laikā tika atklātas dažas būtiskas kļūdas, kas tika novērstas, programmatūras kodā veicot attiecīgus labojumus, un testējot atkārtoti.

Liela daļa no atklātajām mazāk būtiskajām kļūdām tika novērstas testēšanas laikā.

Pie atsevišķu konstatēto kļūdu novēršanas darbs vēl jāturpina, piemēram, vienumam - Kveriju grafa skats (QGV.3).

Tā kā programmatūra ir prototips, kas vēl ir izstrādāšanas fāzē, tad testēšana nākotnē būs jāatkārto un jāveic citos interneta pārlūkos.

Kopējais testēšanas vērtējums ir labs, tā kā tika atklātas nepilnības, un tika apstiprināts, ka izpildās programmatūrai izvirzītās prasības.

4. PROJEKTĒJUMA ORGANIZĀCIJA

Programmatūras izstrādes laikā tika izmantots spirālveida dzīves cikla modelis. Ik pa pāris nedēļām tika uzdotas prasības, kas ir jārealizē, un regulāri ar darba vadītāju tika apspriests, kā prasība izpildās un ko var uzlabot. Pēc tam šīs prasības tika dokumentētas. Tika veidota testēšana uz piemēriem, kas parādīja, vai programmatūra veic nepieciešamās prasības. Ja parādījās kādas kļūdas, tad tās tika lielākoties novērstas. Pēc tam notika koda neliela optimizācija un refaktorēšana.

Darba vadītājs palīdzēja apgūt projekta daļas, kas autoram patstāvīgi nebija pa spēkam. Vadītājs palīdzēja apgūt vidi, bibliotēkas un rīkus, lai autors varētu veiksmīgi realizēt projektu.

Programmatūras izstrādē tika izmantotas tādas valodas kā *javascript, d3.js* bibliotēka un *underscore.js* bibliotēka, kas lielākoties veidoja visa projekta karkasu. Projektēšana un testēšana notika Google Chrome interneta pārlūkā.

Programmatūra tomēr ir prototips, tādēļ ar tādām programmatūras daļām kā kveriju ģenerēšana un skatu apstrādāšana strādāja darba vadītājs.

5. KVALITĀTES NODROŠINĀŠANA

Lai izstrādātu kvalitatīvu programmatūru, pirms darba sākšanas tika definēti noteikumi, kas atbildētu par kvalitātes nodrošināšanu. Programmatūras kods tika rakstīts vienotā stilā. Mainīgo vārdi, funkciju vārdi un komentāri tika rakstīti angļu valodā.

Mainīgo un funkciju vārdi tika veidoti tā, lai atvieglotu lasītāja darbu, lai funkciju darbība būtu saprotama no mainīgo nosaukumiem. Lai programmatūras kods būtu pēc iespējas pārskatāmāks, autors lietoja atkāpes un tukšas rindas starp koda blokiem.

Ja testēšanas laikā tika konstatētas kļūdas programmatūras darbībā, kļūdainās vietas tika labotas un testētas atkārtoti.

Programmatūras prasību specifikācija un programmatūras projektējuma apraksts tika izstrādāti atbilstoši attiecīgajiem Latvijas Valsts standartiem.

6. KONFIGURĀCIJU PĀRVALDĪBA

Konfigurāciju pārvaldībai tika izmantota GitHub interneta lietotne.

Programmatūras kods pārsvarā tika glabāts autora datora cietajā diskā. Pēc ievērojamu izmaiņu veikšanas, kods tika saglabāts GitHub interneta lietotnē.

Tādā veidā autors panāca, ka bija pilnīga kontrole pār izstrādi – darba vadītājs spēja viegli apskatīt darba efektivitāti, progresu un izmaiņas, un autoram gandrīz vienmēr bija pieejama strādājošas programmatūras koda versija. Ja tika konstatētas kļūdas, kuras nevarēja pietiekami ātri novērst, tad varēja vienkārši atkāpties pāris soļus atpakaļ, un turpināt darbu no tās vietas, kad kods strādāja pareizi.

7. DARBIETILPĪBAS NOVĒRTĒJUMS

Lai novērtētu programmatūras darbietilpību, autors izmantoja Basic COCOMO modeli. Šī modeļa formula darbietilpības novērtēšanai ir [a*(KLOC)^b], kur „a" un „b" ir koeficenti, kas ir atkarīgi no izstrādātā projekta veida un KLOC ir koda rindiņu skaits tūkstošos.

Basic COCOMO [4] modelim ir trīs iestatījuma veidi: maziem projektiem, vidējiem un lieliem. Šis projekts ir mazas kapacitātes, tādēļ izvēlamies *Organic* veidu, kas atbild par maza lieluma projektiem. Tas nosaka, ka projektā ir iesaistīta neliela izstrādātāju grupa, kurai ir pietiekami laba pieredze šāda veida projektu izstrādē. Kaut arī autors nebija ar pietiekamu pieredzi šādā sfērā, autoram palīdzēja projekta vadītājs, kas darba gaitu pietuvināja modelī specificētajiem noteikumiem. *Organic* projekta „a" un „b" koeficienti ir attiecīgi 2,4 un 1,05, bet šajā projekta, kur autoram nav pieredze, koeficents a būs 1,9. Autora darba ieguldījums ir tikai 4/5 no visa darba.

Grūtākā daļa, lai novērtētu darbietilpību, bija izstrādāto koda rindiņu skaita prognoze. Parasti šādā gadījumā lieto funkciju skaitīšanu, bet autors to nedarīja, jo šis skaitīšanas veids ņem vērā lietotāja ievades un programmatūras izvades, kas nav projekta galvenais mērķis. Darba autors izvēlējās prognozēt koda rindu skaitu pēc līdzīgu darbu izstrādes un kolēģu ieteikumiem un atsauksmēm.

Nonācām pie tā, ka vienas funkcionālās prasības realizēšanai vajadzētu aizņemt ap vidēji 120 rindiņām. Vēl jāpieskaita melnās kastes pielikums, kas atbildēs par kveriju ģenerēšanu un skatu struktūru realizāciju, un tas varētu veidot vēl kādas 4 funkcionālās prasības. Tādā veidā tika prognozēts, ka kopējais rindiņu skaits = 15 * 120 + 4 * 120 = 1800 + 480 = 2280 koda rindas (2.28 KLOC). Darbietilpība būtu = 1,9 * 2,28 ^ 1,05 = 4,51 personmēneši, no kuriem 4/5 bija autora ieguldījums = 4,51 * 4/5 = 3,61 personmēneši.

Pēc programmatūra izstrādes ar SLOC Metrics [8] rīku tika aprēķināts reālais koda rindu skaits, sasniedzot 2557 rindas (bez komentāriem un tukšajām rindām) jeb 2,56 KLOC. Tātad darbietilpība pēc Basic COCOMO modeļa formulas būtu = 1,9 * 2,58 ^ 1,05 = 5,14 personmēneši. Viena piektdaļa no šā darba izstrādāja vadītājs, tādēļ autoram paliek 5,14 * 4/5 = 4,11 personmēneši, kur kodu optimizējot un refaktorējot sanāktu tuvāk 3 personmēnešu slieksnim.

COCOMO modelis tomēr ir domāts lielākoties lieliem un nepazīstamiem projektiem, vai arī maziem un pazīstamiem. Autoram šis bija mazs projekts, kas īsti nebija pazīstams, un bija nepieciešams vairāk laika nepieciešamo metožu apguvei, tāpēc darbietilpības novērtējums iznāca lielāks par prognozēto.

Patērētais laiks bija tuvu 3,3 personmēnešiem programmatūras izstrādē. Koda rindiņu daudzums apmierina kvalifikācijas darbā specificētās prasības. Darbs tika sākts laicīgi, jo bija iespēja programmatūru izstrādāt prakses ietvaros. Prakses vadītājs un darba vadītājs bija viena persona, kas atviegloja konsultēšanās iespējas.

SECINĀJUMI

Izvirzītais mērķis ir sasniegts - tika izstrādāta programmatūra, ar kuras palīdzību var interaktīvi apskatīt RDF grafa datus.

Autors secina, ka izstrādes sākumā precīzi izstrādāta prasību specifikācija un programmatūras projektējuma apraksts atvieglo programmēšanu un testēšanu, un izpratni par kopējo projekta būtību. Vienota stila ievērošana, rakstot kodu, būtiski atvieglo nākotnē nepieciešamo koda uzturēšanu un izmainīšanu. Koda komentēšana ir noderīga turpmākai koda refaktorēšanai un optimizēšanai.

Autors ieguva ļoti noderīgu pieredzi, strādājot eksperimentālā projektā, iegūstot prasmes programmēt un domāt radoši. Autors iepazinās ar kveriju valodu SPARQL un datu vizualizācijas rīkiem.

Izstrādāto programmatūru var vēl izvērst un pilnveidot ar mērķi efektīvāk prezentēt grafisko kveriju interfeisu iespējas industrijas locekļiem.

Projekta tapšanas laikā autoram bija iespēja iepazīt daudzveidīgo Latvijas Universitātes Matemātikas un informātikas institūta profesionālo darbību saistībā ar projektu izstrādi. Darba izstrāde autoram bija interesanta un aizraujoša, autors labprāt turpinātu strādāt līdzīgā jomā.

Programmatūras izstrāde un testēšana notika Google Chrome 35 versijas ietvaros, tāpēc tās pielāgošana darbam citos interneta pārlūkos būtu uzskatāms kā tālāks programmatūras pilnveidošanas etaps.

LITERATŪRAS UN AVOTU SARAKSTS

1. LVS 68:1996 „Programmatūras prasību specifikācijas ceļvedis”

[tiešsaiste]. [atsauce 29.05.2014].

2. LVS 72:1996 „Ieteicamā prakse programmatūras projektējuma aprakstīšanai”

[tiešsaiste]. [atsauce 30.05.2014].

3. LVS 70:1996 „Programmatūras testēšanas dokumentācija”

[tiešsaiste]. [atsauce 29.05.2014].

4. Basic COCOMO [tiešsaiste]. [atsauce 30.05.2010].
Pieejams: http://en.wikipedia.org/wiki/COCOMO#Basic_COCOMO

5. Dainis Dosbergs, „Programmatūras prasību specifikācija”:

http://estudijas.lu.lv/mod/resource/view.php?id=131436

[tiešsaiste]. [atsauce 27.05.2014].

6. Dainis Dosbergs, „Programmatūras projektējuma apraksts”:

http://estudijas.lu.lv/mod/resource/view.php?id=131444

[tiešsaiste]. [atsauce 29.05.2014].

7. Dainis Dosbergs, „Testēšanas process un testēšanas dokumentācija”

http://estudijas.lu.lv/mod/resource/view.php?id=131450

[tiešsaiste]. [atsauce 28.05.2014].

8. SLOC Metrics [tiešsaiste]. [atsauce 30.05.2014].
Pieejams: http://microguru.com/sloc/[tiešsaiste]. [atsauce 31.05.2014].

9. Terminu vārdnīca: http://termini.lza.lv/term.php

[tiešsaiste]. [atsauce 31.05.2014].

PIELIKUMS

Programmatūras koda fragmenti

Grafisko elementu uzturēšanas funkcija

```
function render_graph(graph) {

  // handles node draging
  var drag_type;
  var ultimate_drag = d3.behavior.drag()
    .origin(function(d) {

      var srcElement = d3.select(d3.event.srcElement);
      if (srcElement.classed('resize-sprite')) {
        drag_type = 'resize';
        console.log(drag_type);
        return {
          x: d.geometry.width,
          y: d.geometry.height
        };
      } else if (srcElement.classed('top-container')) {
        drag_type = 'move';
        return d.geometry;
      } else {

        return {
          x: 0,
          y: 0
        };
      }
    })
    .on('drag', function(d) {
      if (drag_type == 'resize') {

        d.geometry.width = Math.max(d3.event.x, 600);
        d.geometry.height = Math.max(d3.event.y, 300);
        d.geometry.width = Math.min(d.geometry.width, 1500);
        d.geometry.height = Math.min(d.geometry.height, 1000);
      } else if (drag_type == 'move') {
        d.geometry.x = d3.event.x;
        d.geometry.y = d3.event.y;
      } else {
```

```
            return;
        }
        render_graph(graph);

    })
    .on('dragend', function(d) {
        drag_type = null;
    });


var container = d3.select('#query-graph-container')
    .style('position', 'relative');

// make arrays of different elements
var node_arr = _.filter(graph, function(obj) {
    return obj['type'] == 'node';
});

var edge_arr = _.filter(graph, function(obj) {
    return obj['type'] == 'edge' || obj['type'] == 'hyper_edge';
});

// adds database connection values to the inputfields, so
// the user can see to what database is the connection is enstablished
var db_name = d3.select('#db_name_input')
    .attr('value', function(d) {
        return graph.database;
    })

db_name.node().value = graph.database;

db_endpoint = d3.select('#endpoint_input')
    .attr('value', function(d) {
        return graph.endpoint;
    })

db_endpoint.node().value = graph.endpoint;

db_reasoning = d3.select('#reasoning_input')
    .attr('value', function(d) {
        return graph.reasoning;
```

```
    })

db_reasoning.node().value = graph.reasoning;

db_user = d3.select('#user_input')
    .attr('value', function(d) {
        return graph.credentials.user;
    })

db_user.node().value = graph.credentials.user;

db_pass = d3.select('#pass_input')
    .attr('value', function(d) {
        return graph.credentials.pass;
    })

db_pass.node().value = graph.credentials.pass;

//
// DRAW NODES
//
var edge_canvas = container.select('.edge_canvas');
// bind
var nodes = container.selectAll('.query-node')
    .data(node_arr, function(d) {
        return d.name;
    });

// create
nodes.enter()
    .append('div')
    .classed('query-node', true)
    .style('position', 'absolute')
    .attr('id', function(node) {
        return node['name'];
    })
    .call(ultimate_drag)
    .each(function(node) {

        var node_div = d3.select(this);
```

```
node.node_div = node_div;

// at start greate the three major node sections
var top_div = node_div.append('div')
    .attr('class', 'top-container')
    .style('position', 'absolute')
    .style('height', 40 + 'px')

var middle_node_div = node_div
    .append('div')
    .attr('class', 'middle-container')
    .attr('id', node.name + '1')
    .style('position', 'absolute')
    .style('top', '40px')

var bottom_div = node_div.append('div')
    .attr('class', 'bottom-container')
    .style('position', 'absolute')
    .style('height', 40 + 'px')



// after the major sections are created the buttons and actions
// are put in into these sections
var type_field = top_div.append('div')
    .style('position', 'absolute')
    .style('left', 50 + 'px')
    .style("font-family", "sans-serif")
    .style("font-size", '30px')


var type_selection_container = type_field.append('div')
    .append('select')
        .style('width', 400 + 'px')
        .classed('input-group', true)
        .classed('form-control', true)
        .classed('type_combobox', true)
        .classed('type_ul', true)
        .on('change', function(node) {

            select = d3.select(this);
```

```
            options = select.selectAll('option');

            var selectedIndex = select.property('selectedIndex');

            var data = d3.select(options[0][selectedIndex]).datum();
            node.query_param.current_type = data;

            render_graph(graph);
            fill_tables(graph);
        });

var barchart_container = top_div.append('div')
    .style('position', 'absolute')
    .style('left', 450 + 'px')
    .style("font-family", "sans-serif")
    .style("font-size", '30px')

var barchart_selection_container = barchart_container.append('div')
        .append('select')
            .style('width', 400 + 'px')
            .classed('input-group', true)
            .classed('form-control', true)
            .classed('barchart_combobox', true)
            .classed('type_ul', true)
            .on('change', function(node) {

                select = d3.select(this);
                options = select.selectAll('option');

                console.log('bar')
                get_all_barchart_columns(graph, node);
                var selectedIndex = select.property('selectedIndex');

                var data = d3.select(options[0][selectedIndex]).datum();
                node.query_param.current_type = data;
                console.log(node, data, graph)

                if (data != undefined) {
                    column_id = _.chain(node.columns)
                        .map(function(column) {
                        if (column.column_label == data) {
```

```
                    return column.id
                }
                })
                .compact()
                .value();

                node.visualization_defs.BarChartView.properties.y_axis_column =
column_id[0];
            } else {
                node.visualization_defs.BarChartView.properties.y_axis_column =
null;
            }

            fill_tables(graph);
            render_graph(graph);
        });



    top_div.append('div')
        .classed('button', true)
        .classed('glyphicon glyphicon-info-sign type_query', true)
        .style('height', 40 + 'px')
        .style('width', 40 + 'px')
        .on('click', function(node) {
            console.log('type query\n', node.query_param.type_query);
        });

    top_div.append('div')
        .classed('glyphicon glyphicon-signal', true)
        .classed('button', true)
        .style('position', 'absolute')
        .style('height', 40 + 'px')
        .style('width', 40 + 'px')
        .style('top', 9 + 'px')
        .style('left', 5 + 'px')
        .on('click', function(node) {
            if (node['active_visualization_type'] == 'TableView') {
                node['active_visualization_type'] = 'BarChartView';
            } else {
                node['active_visualization_type'] = 'TableView';
            }
            console.log(node['active_visualization_type']);
```

```
            render_graph(graph);
            fill_tables(graph);
        });


    top_div.append('div')
        .attr('class', 'delete-node')
        .style('position', 'absolute')
        .style('top', '-11px')
        .on('click', function(node) {
            delete_node(graph, node);
        })

    top_div.append('div')
        .attr('class', 'input-window')
        .classed('glyphicon glyphicon-align-justify', true)
        .classed('button', true)
        .attr('data-toggle', 'modal')
        .attr('data-target', '.bs-example-modal-lg')
        .style('position', 'absolute')
        .style('top', 9 + 'px')
        .style('left', 5 + 'px')
        .style('height', 40 + 'px')
        .style('width', 40 + 'px')
        .on('click', function(node) {
            render_columns(graph, node);
        })

    bottom_div.append('div')
        .classed('button', true)
        .attr('id', node.name)
        .style('position', 'absolute')
        .style('height', 40 + 'px')
        .style('width', 40 + 'px')
        .call(ultimate_drag)
        .append('div')
            .style('position', 'absolute')
            .classed('glyphicon glyphicon-resize-full resize-sprite', true)
            .style('top', 5 + 'px')
            .style('left', 5 + 'px')
            .call(ultimate_drag);
```

```
bottom_div.append('div')
    .attr('class', 'query-button')
    .classed('glyphicon glyphicon-info-sign', true)
    .classed('button', true)
    .attr('id', node.name)
    .style('position', 'absolute')
    .style('top', 5 + 'px')
    .style('left', 5 + 'px')
    .style('height', 40 + 'px')
    .style('width', 40 + 'px')
    .on('click', function(node) {
        var query = generate_query(graph, node);
        console.log('query: \n', query, '\n', node.query_param);
    });

bottom_div.append('div')
    .attr('class', 'refresh-button')
    .classed('glyphicon glyphicon-refresh', true)
    .classed('button', true)
    .attr('id', node.name)
    .style('position', 'absolute')
    .style('top', 5 + 'px')
    .style('left', 5 + 'px')
    .style('height', 40 + 'px')
    .style('width', 40 + 'px')

    .on('click', function(node) {
        fill_tables(graph);
    });


bottom_div.append('div')
    .attr('class', 'delete-selection')
    .classed('glyphicon glyphicon glyphicon-repeat', true)
    .classed('button', true)
    .attr('id', node.name)
    .style('position', 'absolute')
    .style('top', 5 + 'px')
    .style('left', 5 + 'px')
    .style('height', 40 + 'px')
    .style('width', 40 + 'px')
```

```
            .on('click', function(node) {
                // delete selections
                node['query_param']['selection'] = [];
                fill_tables(graph);
            });

    });

// at changes the element properties are just updated with d3, not deleted
// update
nodes
    .style('left', function(node) {
        return node.geometry.x + 'px';
    })
    .style('top', function(node) {
        return node.geometry.y + 'px';
    })
    .style('width', function(node) {
        return node.geometry.width + 'px';
    })
    .style('height', function(node) {
        return node.geometry.height + 'px';
    })
    .call(function(node) { // updating sizes and positions

        var node_div = this;


        var top_div = node_div.select('.top-container')
            .style('width', function(node) {
                return node.geometry.width + 'px';
            });

        var middle_node_div = node_div.select('.middle-container')
            .style('width', function(node) {
                return node.geometry.width + 'px';
            })
            .style('height', function(node) {
                return node.geometry.height - 100 + 'px';
            });

        var bottom_div = node_div.select('.bottom-container')
```

```
    .style('top', function(node) {
        return node.geometry.height - 40 + 'px';
    })
    .style('width', function(node) {
        return node.geometry.width + 'px';
    });



//bind
var type_options = top_div.select('.type_ul').selectAll('option')
    .data(function(d) {
        return d.query_param.type_arr;
    })

//create
type_options.enter()
    .append('option')
    .classed('option', true)



//update
type_options.text(function(d) {

        return d;
    });



//exit
type_options.exit().remove();

// bind
var bar_options = top_div.select('.barchart_combobox').selectAll('option')
    .data(function(node) {
        get_all_barchart_columns(graph, node);
        return get_all_barchart_columns(graph, node);
    })

//create
bar_options.enter()
    .append('option')
    .classed('option', true)
```

```
//update
bar_options.text(function(d) {
        return d;
    });


//exit
bar_options.exit().remove();

top_div.select('.glyphicon-signal')
    .style('left', function(node) {
        return node.geometry.width - 100 + 'px';
    });


top_div.select('.delete-node')
    .style('left', function(node) {
        return node.geometry.width - 25 + 'px';
    });

top_div.select('.input-window')
    .style('left', function(node) {
        return node.geometry.width - 150 + 'px';
    });

bottom_div.select('.resize-sprite')
    .style('left', function(node) {
        return node.geometry.width - 40 + 'px';
    });

bottom_div.select('query-button')
    .style('left', function(node) {
        return node.geometry.width - 280 + 'px';
    });

bottom_div.select('.refresh-button')
    .style('left', function(node) {
        return node.geometry.width - 320 + 'px';
    });
```

```
            bottom_div.select('.delete-selection')
                .style('left', function(node) {
                    return node.geometry.width - 390 + 'px';
                });

            bottom_div.select('.make-something')
                .style('left', function(node) {
                    return node.geometry.width - 490 + 'px';
                })
        })
        .each(function(node) { // updating container visualization

            var view_construct_for_view_type = {
                'TableView': TableView,
                'BarChartView': BarChartView
            }

            var node_div = d3.select(this);

            var gridContainer = node_div.select('.middle-container');
            // check what visualization type is set
            if (!node.current_visualization_view) { // if it is not defined then it is made and the view are refreshed

                var view_constructor = view_construct_for_view_type[node.active_visualization_type];

                node.current_visualization_view = view_constructor().init(gridContainer.node(), node, graph);

            } else if (node.current_visualization_view.type == node.active_visualization_type) {

                // if the rigth type is set, then only view refreshing is needed
                node.current_visualization_view.refresh_view_from_node();

            } else { // if a undefined type is defined then it is deleted and there is a good one made

                node.current_visualization_view.destroy();

                var view_constructor = view_construct_for_view_type[node.active_visualization_type];
```

```
                node.current_visualization_view = view_constructor().init(gridContainer.node(),
node, graph);

            }

            console.assert(node.current_visualization_view)

        });

        //deleting nodes that are no more in the graph
        // remove
        nodes.exit()
            .remove();

        // start making edge elements and their properties
        //
        // DRAW EDGES
        //

        var edge_delete_field = edge_canvas.selectAll('.delete_edge')
            .data(edge_arr, function(d) {
                return d.name;
            });
        // create
        edge_delete_field.enter()
            .append('rect')
                .classed('delete_edge', true)
                .attr('height', 40 + 'px')
                .attr('width', 40 + 'px')
                .on('click', function(edge) {
                    delete_edge(graph, edge);
                    render_graph(graph);
                    fill_tables(graph);
                })

        //update
        edge_delete_field
            .attr('x', function(d) {
                return (d.geometry.x1 + d.geometry.x2 - 100) / 2;
            })
            .attr('y', function(d) {
                return (d.geometry.y1 + d.geometry.y2 - 100) / 2;
```

```
        })


    edge_delete_field.exit().remove();

    // bind
    var edges = edge_canvas.selectAll('.query-edge')
        .data(edge_arr, function(d) {
            return d.name;
        });

    //create
    edges.enter()
        .append('path')
        .classed('query-edge', true)
        .attr('marker-end', "url(#link_path_end)")
        .attr('id', function(node) {
            return node['name'];
        });

    // update

    edges.each(function(edge) {

        updateEdgeCoordinates(graph, graph[edge.start]['geometry'],
graph[edge.end]['geometry'], 5, edge.name);
    })
        .attr('d', function(d) {
            var path = 'M' + d.geometry.x1 + ',' + d.geometry.y1 + 'L' + d.geometry.x2 + ',' +
d.geometry.y2;

            return path;
        })

    // delete removed
    edges.exit().remove()

    // create paths for flow lines that show the flow of the edge
    // bind data
    var flow_edges = edge_canvas.selectAll('.query-edges_red')
        .data(edge_arr, function(d) {
            return d.name;
```

```
        })

    // create new
    flow_edges.enter()
        .append('path')
        .classed('query-edges_red', true)
        .attr('stroke', "red")
        .attr('stroke-width', "9px")
        .attr('id', function(node) {
            return node['name'];
        })
        .on('click', function(d) {
            toggleWay(graph, d);
            return;
        });

    // update all
    flow_edges
        .attr('d', function(d) {

            if (d.geometry.x1 == d.geometry.x2) {
                var path = 'M' + (d.geometry.x1 + 20) + ',' + d.geometry.y1 + 'L' + (d.geometry.x2
+ 20) + ',' + d.geometry.y2;
            } else {
                var path = 'M' + d.geometry.x1 + ',' + (d.geometry.y1 - 20) + 'L' + d.geometry.x2
+ ',' + (d.geometry.y2 - 20);
            }

            return path;
        })
        .attr('marker-end', function(edge) {

            if (_.contains(graph[edge.end]['incoming_lines'], edge.name)) {
                return 'url(#link_path_end_red)';
            } else return '';
        })
        .attr('marker-start', function(edge) {
            if (_.contains(graph[edge.start]['incoming_lines'], edge.name)) {
                return 'url(#link_path_start)';
            } else return '';
        });
```

```
    // remove deleted
    flow_edges.exit().remove();

    // bind lable
    var lables = edge_canvas.selectAll('.text')
        .data(edge_arr, function(d) {
            return d.name;
        });

    //create new labels
    lables.enter()
    .append('text')
        .attr('class', 'text')
        .text(function(d) {
            return d.name;
        })
        .attr("font-family", "sans-serif")
        .attr("font-size", "20px")
        .attr("fill", "balck");

    // update labels
    lables
    .attr('x', function(d) {
        return d.geometry['x'] || (d.geometry['x2'] - 20);
    })
        .attr('y', function(d) {
            return (d.geometry['y'] - 0) || (d.geometry['y2'] + 10);
        })

    // remove lables

    lables.exit().remove();
}
```

Virsotnes izveides funkcija

```
function make_node(graph, x, y) {

    var table_name;

    // gets new name for node
```

```
    table_name = 't' + get_max_element_number(graph, 'node');

    // node template that will go into query_module
    var temp = {
        'name': table_name,
        'query_var_name': table_name,
        'type': 'node',
        'incoming_lines': [],
        'query_param': {
            'type_arr': [],
            'current_type': null,
            'selection': []
        },
        'geometry': {
            'x': x,
            'y': y,
            'width': 1010,
            'height': 500
        },
        // must be one of keys in visualization_defs
        'active_visualization_type': 'TableView',
        'visualization_defs': {
            'TableView': {

            },
            'BarChartView': {
                'properties': {
                    'y_axis_column': '??'
                }
            }
        },
        'columns': [
        ]
    };
    // adds element to query_graph
    graph[table_name] = temp;

    // refreshes all views
    render_graph(graph);
    fill_tables(graph);
}
```

Atkarības plūsmas maiņas funkcija

```
function toggleWay(graph, edge) {

    var start_node = edge['start'];
    var end_node = edge['end'];

    //console.log('swap', edge.name, graph[start_node], graph[end_node]);

    if ((_.contains(graph[start_node]['incoming_lines'], edge.name)) && !(_.contains(graph[end_node]['incoming_lines'], edge.name))) {
        graph[start_node]['incoming_lines'] = _.without(graph[start_node]['incoming_lines'], edge.name);

        graph[end_node]['incoming_lines'].push(edge.name);

    } else if (!(_.contains(graph[start_node]['incoming_lines'], edge.name)) && (_.contains(graph[end_node]['incoming_lines'], edge.name))) {
        graph[start_node]['incoming_lines'].push(edge.name);

    } else if ((_.contains(graph[start_node]['incoming_lines'], edge.name)) && (_.contains(graph[end_node]['incoming_lines'], edge.name))) {
        graph[start_node]['incoming_lines'] = _.without(graph[start_node]['incoming_lines'], edge.name);
        graph[end_node]['incoming_lines'] = _.without(graph[end_node]['incoming_lines'], edge.name);

    } else if (!(_.contains(graph[start_node]['incoming_lines'], edge.name)) && !(_.contains(graph[end_node]['incoming_lines'], edge.name))) {
        graph[start_node]['incoming_lines'].push(edge.name);

    }
    render_graph(graph)
    fill_tables(graph);
}
```

Kverija grafa skata inicializācija

```
function init_query_graph(graph) {
```

```
// makes container for all elements that will be handled
var query_graph_container = d3.select('#query-graph-container')
    .style('position', 'relative');

// makes container for all edges
var edge_container = query_graph_container.append('svg')
    //.style('position', 'absolute')
    .attr('width', 4000)
    .attr('height', 2000)
    .style('top', 30 + 'px')
    .classed('edge_canvas', true)
    .call(make_marker)
    .on('click', function(d) {

        var x = d3.mouse(this)[0];
        var y = d3.mouse(this)[1];
        if (this == d3.event.target) {
            make_node(graph, x, y);
        }

    });

// starts making form for rendering columns
init_column_make_container();


// assign function to connection to database input fields
d3.select('#db_name_input')
    .data(graph.database)
    .on('change', function(d) {

        graph.database = this.value;

        fill_tables(graph);
    })


d3.select('#endpoint_input')
    .on('change', function(d) {
```

```
        graph.endpoint = this.value;

        fill_tables(graph);
    })

d3.select('#reasoning_input')
    .on('change', function(d) {

        graph.reasoning = this.value;

        fill_tables(graph);
    })

d3.select('#user_input')
    .on('change', function(d) {

        graph.credentials.user = this.value;

        fill_tables(graph);
    })

d3.select('#pass_input')
    .on('change', function(d) {

        graph.credentials.pass = this.value;

        fill_tables(graph);
    })


// catches some of the column value changes
$(document.body).on('click', '.column_ul', function(event) {

    console.log(event.srcElement.text)
    var li = d3.select(event.currentTarget);
    console.log('a', li.data(),
        li.datum().update(event.srcElement.text))

    render_columns(graph, li.datum().node);
    render_graph(graph);
    fill_tables(graph);
```

```
    });

    // catches the event that will make an edge
    d3.select('body')
    .on('keydown', function() {
        // event will happen when 'shift' is pressed
        if (d3.event.keyCode == 16) {
            event_for_making_edge1(graph);
            return;
        }
    });


    // calls functions that will show graph in the browser
    render_graph(graph);
    fill_tables(graph);
}
```

Kvalifikācijas darbs „**Interaktīvais datu apskates rīks, kas bāzēts uz ad-hoc koordinētu grafa apskati interneta pārlūkā**” izstrādāts Latvijas Universitātes Datorikas fakultātē.

Ar savu parakstu apliecinu, ka darbs izstrādāts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: ***Arnis Bērziņš*** _____ **___.06.2014**.

Rekomendēju darbu aizstāvēšanai
Darba vadītājs: ***M. Dat., Renārs Liepiņš*** _____ **___.06.2014.**

Recenzents: ***M. Dat., Pēteris Krastiņš***

Darbs iesniegts **___.___.2014**.
Kvalifikācijas darbu pārbaudījumu komisijas sekretārs: ____

Darbs aizstāvēts kvalifikācijas darbu pārbaudījuma komisijas sēdē
___.06.2014. prot. Nr. _____

Komisijas sekretārs(-e): _____