ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტი
ზუსტ და საბუნებისმეტყველო მეცნიერებათა ფაკულტეტი

კომპიუტერული მეცნიერება

გრიგოლი ბაბაჯანიანი

ქართული ელექტრონული სიტყვათა ქსელის ვიზუალიზაციის და კლიენტური ნაწილის შექმნის პროცესის აღწერა

ნაშრომი შესრულებულია ინფორმატიკის ბაკალავრის აკადემიური ხარისხის მოსაპოვებლად

ხელმძღვანელი: ლიანა ლორთქიფანიძე

ასისტენტ პროფესორი

თბილისი
2017

სარჩევი

1. ანოტაცია

ამჟამად არსებობს სხვადასხვა ტიპის ლექსიკონები, მაგალითად EuroWordNet, SentiWordNet, WordNet. მათ შორის ყველაზე პოპულარული არის WordNet-ი, რომელიც ინგლისური სიტყვებისგან შედგენილი დიდი ლექსიკონია. WordNet-ს იყენებენ არა მარტო ლინგვისტები, არამედ იგი ფართოდ გამოიყენება ხელოვნურ ინტელექტში, მაგალითად, ჩატბოტებში, სენტიმენტ ანალიზში, Information Retrieval სისტემებში. ჩვენ ვფიქრობთ, რომ ასეთი ლექსიკონი აუცილებელია ქართულ ენაზე დამფუძვნებული მაღალ-ტექნოლოგიური პროდუქტების შექმნისათვის. ამ ნაშრომში აღწერილია ქართული ელექტრონული სიტყვათა ქსელის შექმნის პროცესი. ნაშრომის დასკვნით ნაწილში ჩვენ შევეცდებით წარმოავდგინოთ პროგრამის მუშაობის შედეგად მიღებული სიტყვათა ქსელის ვიზუალიზაცია.

There are a lot of different kind of lexicons in the internet, for example EuroWordNet, SentiWordNet, WordNet and etc. Most popular between them is a WordNet. WordNet is a large lexical database of English, which applied in different kind of applications: Chatbots, Sentiment Analysis, Information Retrieval Systems. We decided to develop WordNet for Georgian language, because we believe, that such a lexicon will be helpful for IT companies which will try to develop AI-based tools and applications with processing texts on Georgian language. On the last part of this paper we show results of working of our system.

2. შესავალი

Wordnet (ელექტრონული თესაურუსი/სემანტიკურ სიტყვათა ქსელი) შემუშავებული იყო პრინსტონის უნივერსიტეტში, ხოლო პირველად მისი გაშვება მოხდა ინგლისური ენისთვის 1995 წელს.

თესაურუსი შედგება 4 ქსელისაგან სხვადასხვა მეტყველების ნაწილისთვის: არსებითი სახელი, ზედსართავი სახელი, ზმნა და ზმნიზედა. საბაზისო ერთეულს წარმოადგენს არა სიტყვა, არამედ სინონიმური რიგი (**სინსეტი**), რომელიც აერთიანებს მსგავსი მნიშნველობების მქონე სიტყვებს.

რადგანაც ამ ქსელს ამდაგვარი სტრუქტურა აქვს, შესაძლებელი ხდება მისი გამოყენება ისეთ სფეროებში, როგორებიცაა: გამოთვლითი ლინგვისტიკა და ბუნებრივი ენის კომპიუტერული მოდელირება - NLP (Natural Language Processing).

ჩვენ შევიმუშავეთ ქართული სიტყვათა ქსელის **web** აპლიკაცია, რომელიც საშუალებას აძლევს მომხმარებელს ადვილად მოძებნოს მისთვის საინტერესო სიტყვის სინონიმთა ჯგუფები, თავისი აღწერებით.

კლიენტურ მხარეზე გამოყენებულია შემდეგი ტექნოლოგიების სტეკი: AngularJS და Vis.js. უკანასკნელის საშუალებით მოვახდინეთ ქსელის ვიზუალიზაცია გრაფის ფორმით.

აღსანიშნავია, რომ ეს არის პირველი მცდელობა ასეთი ტიპის ქსელის შექნისა ქართული ენისთვის. მიგვაჩნია, რომ ამდაგვარი მიდგომა იქნება წინ გადადგუმლი ნაბიჯი ქართული ბუნებრვი ენის მოდელირების საკითხში და ხელს შეუწყობს რიგი პროექტების განხორციელბას ხელოვნურ ინტელექტსა და მანქანური თარგმინის სისტემებში.

3. WordNet/სიტყვათა ქსელი

თანამედროვე ადამიანის ცხოვრებაში მნიშვნელოვანი ადგილი უჭირავს ავტომატიზირებულ ინფორმაციულ ტექნოლოგიებს. მეოცე საუკუნეში დაწყებული კომპიუტერული მოდელირება შეუქცევად პროცესად იქცა და დროთა განმავლობაში უფრო და უფრო იხვეწება და ვითარდება. უნდა აღინიშნოს, რომ ინფორმაციული განვითარება არაერთგავროვნად მიმდინარეობს: თუ გამომთვლელი ტექნიკისა და კომუნიკაციური ტექნოლოგიის განვითარებამ თანამედროვე ეპოქაში უმაღლეს წერტილს მიაღწია, ინფორმაციის თეორიის დამუშავება არც ისე წინ არის წასული და მრავალ ლინგვისტურ თუ სემანტიკურ პრობლემას აწყდება. ამგვარი ტექნოლოგიური წარმატება, პირველ რიგში, ეფუძნება იმ მიღწევებს, რომლებიც აღინიშნება ადამიანის აზროვნების პროცესის შესწავლაში, ხალხთაშორის კომუნიკაციის დამყარებასა და ამ პროცესის მოდელირების შესაძლებლობაში.

როდესაც ჩვენ საუბარი გვაქვს პერსპექტიული საინფორმაციო ტექნოლოგიის შექმნაზე პირველ პლანზე მოდის ტექსტური ინფორმაციის ავტომატური დამუშავება ბუნებრივი ენის ფარგლებში. ეს აიხსნება იმით, რომ ადამიანის აზროვნება მჭიდრო კავშირშია ენასთან.

აზროვნება და ტექსტი განუყოფელ ელემენტებად გვევლინება. მეტიც, ბუნებრივი ენა გახლავთ აზროვნების ინსტრუმენტი. ბუნებრივი ენა არის ადამიანთა ურთიერთობის უნივერსალური საშუალება - იგი არის საშუალება ინფორმაციის შემეცნების, დაგროვების, შენახვის, დამუშავებისა და გადაცემის. დისციპლინას, რომელიც შეისწავლის ბუნებრივი ენის გამოყენებას ავტომატიზირებულ საინფორმაციო ტექნოლოგიებში ეწოდება კომპიუტერული ლინგვისტიკა. იგი თანამედროვე დისციპლინაა და დასაბამს იღებს მეოცე საუკუნის 50-60-ანი წლებიდან. გასული 50 წლის მანძილზე კომპიუტერულ ლინგვისტიკაში მნიშვნელოვან მეცნიერულ მიღწვებს ჰქონდათ ადგილი, ასევე მიღებული იქნა პრაქტიკული შედეგები: შეიქმნა მანქანური თარგმნის სისტემები, რომლებიც ტექსტს თარგმნიდნენ ერთი ენიდან მეორეზე, ავტომატური ანალიზის სისტემები და სხვა.

სიტყვათა ქსელი (Word Net) არის ლექსიკურ მონაცემთა ბაზა. მისი ერთ-ერთი ყველაზე უფრო ფართო მახასიათებელია სიტყვების გაერთიანება სინონიმურ ჯგუფებად, რითიც ის ქმნის განმარტებებსა და გამოყენების მაგალითებს, იმახსოვრებს რა უამრავი ტიპის დამოკიდებულებას სინონიმურ ბმულებს შორის. სიტყვათა ქსელს ხშირად აღწერენ როგორც ლექსიკონისა და თესაურუსის კომბინაციას. იგი აგებულია ტექსტის ანალიზისა და ხელოვნური ინტელექტის მანიპულაციების საშუალებით და მოხმარებლამდე მიდის ვებ გვერდის სახით. ამგავრი

სიტყვათა ქსელის შექმნის მიზანი გახლდათ ისეთი ლექსიკური მონაცემთა ბაზის აგება რომელიც მჭიდრო კავშირში იქნებოდა ადამიანის სემანტიკურ მეხსიერებათან, რომელიც თავის მხრივ, როგორც მეცნიერული დისციპლინა, ჩამოყალიბდა მეოცე საუკუნეს 60-იან წლებში და დიდი პოპულარობით სარგებლობდა როგორც მანქანური სისტემების სპეციალიობებში, ასევე ლინგვისტიკასა და ლოგიკაში. 1960 წელს, როდესაც მოხდა ხელოვნური ინტელექტის, პირველი რობოტის, პრაქტიკული რეალიზაცია, კომპიუტერულ ლინგვისტიკას დაეკისრა უდიდესი და უმნიშვნელოვანესი როლი - დაემყარებინა კომუნიკაცია რობოტსა და ადამიანს შორის, ამისათვის სამუშაოები გაგრძელდა და ძირითად პრობლემურ მხარეებად გამოიკვეთა სემანტიკური და პრაგმატიკული ასპექტები.

ფსიქოლოგიურმა ექსპერიმენტმა აჩვენა, რომ მოლაპარაკე ცდისპირები თავიანთ კონცეპტუალურ ცოდნას აორგანიზებდნენ ეკონომიური, იერარქიული მეთოდის გამოყენებით. აღმოჩნდა რომ მოლაპარაკეს სჭირდებოდა გარკვეული დრო იმისათვის, რომ თავისი ცოდნა მეხსიერებაში დაემოდელირებინა, იმის მიხედვით თუ რა კონცეპტი იყო შერჩეული. მაგალითად: სიტყვას „*კანარი*“ ცდისპირები ადვილად უკავშირებდნენ „*სიმღერას*“, რადგაცნ თვითონ სიტყვა ატარებს *მომღერალი ფრინველის* მნიშვნელობას, ხოლო ამავე სიტყვას უფრო მეტ დროში აკავშირდებდნენ სიტყვასთან „*ფრენა*“, რადგან ცდისპირი ჯერ უნდა მისულიყო სიტყვასთან „*ფრინველი*“ და შემდეგ „*ფრენასთან*“ (ინგ: canary, bird).

სიტყვათა ქსელს საკმაოდ ფართო გამოყენება აქვს ინფორმაციულ სისტემებში, ასე მაგალითად: სიტყვა-იდენტობის დისბინაცია, ინფორმაციის მოძიება, ტექსტის ავტომატური კლასიფიკაცია, ტექსტის ავტომატური შეჯამება, მანქანური თარგამანი და კროსვორდული ამოცანების გენერაცია. დღესდღეობით ერთ-ერთ აქტუალურ პრობლემას წარმოადგენს სწორედ დოკუმენტებისათვის ტოპიკების მინიჭება, ანუ ინდექსირება, რისთვისაც სიტყვათა ქსელი აქტუალურად გამოიყენება.

თავდაპირველად ინდექსირების პროცესი აღიქმებოდა, როგორც დოკუმენტებისა და მოთხოვნებისათვის საკლასიფიკაციო ინდექსის მინიჭების პროცესი, რაც საშუალებას გვაძლევდა გაგვერკვია ამა თუ იმ ტექსტის თემატური შინაარსი. შემდგომში ამ მიდგომამ ტრანსფორმაცია განიცადა და ტერნიმინი „*ინდექსირება*“ მიემართა იმ პროცესს, რომელიც საშუალებას გვაძლევს ბუნებრივ ენაზე მოცემული დოკუმენტებისა და მოთხოვნების ჩაწერას ფორმალიზებულ ენაზე. ამ პროცედურის შედეგად მიღებულ გამოსახულებას შეგვიძლია ვუწოდოთ *საძიებო გამოსახულება.*

დოკუმენტის საძიებო გამოსახულებაში დაიწყეს საკვანძო სიტყვებისა და სიტყვათშეთანხმებების ფიქსირება. ისინი ასახავენ დოკუმენტების თემატურ შინაარსს. მოთხოვნების საძიებო გამოსახულებები აისახება ლოგიკური კონსტრუქციების ფორმით. აქ საკვანძო სიტყვები და სიტყვათშეთანხმებები ერთმანეთს დაუკავშირდნენ ლოგიკური და სინტაქსური ოპერატორების მეშვეობით.

დოკუმენტების ავტომატური ინდექსირების პროცესი მოსახერხებელია რეფერატული ტექსტებისათვის, რადგან რეფერატებში მთავარი შინაარსი წინა პლანზეა წამოწეული. ინდექსირება შესაძლებელია ჩატარდეს თესაურუსის გამოყენებითაც და მის გარეშეც.

პირველ ეტაპზე, დოკუმენტის სათაურში და მის რეფერატში იძებნება საკვანძო სიტყვები და სიტყვათშეთანხმებები, რომლებიც წარმოდგენილი არიან ეტალონ მანქანურ ლექსიკონში.

მეორე ეტაპზე, საკვანძო სიტყვები და სიტყვათშეთანხმებები გამოიყოფა ტექსტიდან და შედის სისტემაში იმის მიუხედავად, მოიძებნება თუ არა ისინი ეტალონ-მანქანურ-ლექსიკონში.

ასევე, რეალიზებულია მესამე ვარიანტიც, სადაც სამუშაო სისტემაში შედის როგორც ტერმინები მანქანური თესაურუსიდან, ასევე ტერმინები, რომლებიც მოცემულია დოკუმენტის სათაურში და მის პირველ წინადადებაში.

ექსპერიმენტების შედეგებმა დაადგინეს, რომ ინდექსირებისათვის გამოყენებული მექანიკური, ავტომატური სისტემები ბევრად კარგ შედეგს იძლევა და ეკონომიურია დროის თავლსაზრისით, ვიდრე ხელით ჩატარებული ინდექსირების პროცედურა. ეს შედეგი შესაძლებელია აიხსნას იმით, რომ ავტომატურ სისტემებში შეტანილია დოკუმენტების შინაარსის სხვადასხვა პარამეტრები.

მოთხოვნების ინდექსირებისას ჩვენ წინაშე წამოიჭრება იგივე პრობლემა, რომელიც თავს იჩენს დოკუმენტების ინდექსირებისას. აქაც აუცილებელია გამოიყოს საკვანძო სიტყვები და სიტყვათშეთანხმებები. საკვანძო სიტყვებსა და სიტყვათშეთანხმებებს და კონტექსტუალურ ოპერატორებს შორის კავშირის დამყარება შესაძლებელია ხელით ან ავტომატიზირებული პროცედურების გამოყენებით. მოთხოვნის ინდექსირებისას მნიშვნელოვანი როლი ეკისრება, ასევე, საკვანძო სიტყვებისა და სიტყვათშეთანხმებების სინონიმებსა და ჰიპონიმებს. მათი შევსება შესაძლებელია თესაურუსის გამოყენებით.

დოკუმენტური ინფორმაციის ავტომატური ძიების პრობლემა ჩვენ უკვე ნაწილობრივ განვიხილეთ, როდესაც საუბარი გვქონდა ინდექსირებაზე. ამ შემთხვევაში პერსპექტიულია

დოკუმენტების ძიება მათი სრული ტექსტის მიხედვითაც. ამ მიზნის მისაღწევად გამოყენებულმა ყველანაირმა ჩანაცვლებამ შესაძლოა მიგვიყვანოს ძიების პროცესში ინფორმაციის დაკარგვამდე. დანაკარგები თავს იჩენენ მაშინაც, როდესაც ჩანაცვლების ფორმით გამოიყენება მათი ბიბლიოგრაფიული აღწერილობა.

ინფორმაციის ძიების მნიშვნელოვანი მახასიათებლებია ტექსტის სისრულე და სიზუსტე. ძიების დადებითი შედეგები განპირობებულია მაქსიმალურად სწორად აღქმული პარადიგმატული კავშირებით, რომლებიც მყარდება ენისა და მეტყველების ერთეულებს შორის, ხოლო სიზუსტე ემყარება სინტაქტიკურ კავშირებს.

მონაცემთა სრულტექსტიან ბაზაში ინფორმაციის ძიება უნდა ეფუძნებოდეს დიალოგებს, რომლებიც „იმართება“ საინფორმაციო-საძიებო სისტემებსა და მათ მომხმარებლებს შორის. ამგვარ სისტემებში ცალ-ცალკე განიხილება ტექსტის ფრაგმენტები (აბზაცები, პარაგრაფები), მათგან ამოირჩევა ისეთები, რომლებიც შეესაბამებიან მოთხოვნებს. საბოლოოდ, ძიების დასასრულს, ჩვენ შეიძლება მივიღოთ დოკუმენტის მთლიანი ტექსტი ან მისი ფრაგმენტული ნაწილი, როგორც ძიების რეზულტატი.

ინფორმაციის ავტომატური ძიებისას გვიხდება ენობრივი ბარიერის გადალახვა, რაც წამოიჭრება ინფორმაციულ-საძიებო სისტემასა და მის მომხმარებელს შორის; ასევე მაშინ, როდესაც განსხვავებული ფორმის ტექსტები ერთი და იმავე შინარსს ატარებენ. აღნიშნული ბარიერი კიდევ უფრო მნიშვნელოვანი ხდება, თუ ჩვენ ძიებას ვაწარმოებთ მრავალენოვან მონაცემთა ბაზაში.

პრობლემის კარდინალური გადაწყვეტის გზა ამგვარ შემთხვევაში შესაძლებელია იყოს დოკუმენტური ტექსტების მანქანური თარგმნა ერთი ენიდან მეორე ენაზე. ეს პროცედურა უნდა ჩატარდეს ან საძიებო სისტემაში დოკუმენტების ჩატვირთვამდე ან თავად ინფორმაციის ძიების პროცესისას.

მეორე შემთხვევაში, მომხმარებლის მოთხოვნა უნდა ითარგმნოს დოკუმენტების მასივის იმ ენაზე, სადაც მიმდინარეობს ძიება, ხოლო ძიების რეზულტატები კი უნდა ითარგმნოს მოთხოვნის ენაზე. ასეთი ტიპის საძიებო სისტემები უკვე დანერგილია ინტერნეტში და ხელმისაწვდომია მომხმარებლისათვის. ამგვარდაა აგებული Cyrillic Browser-ის სისტემა, რომელიც საშუალებას გვაძლევს ინფორმაციის ძიება მოვახდინოთ რომელიმე არაინგლისურ ენოვან ტექსტებში მაშინ, როდესაც მოთხოვნა ინგლისურ ენაზეა და მივიღოთ შედეგი იმ ენაზე, რომელსაც იყენებს მომხმარებელი

სიტყვათა ქსელის ყველაზე ფართო გამოყენება მოიცავს სიტყვებს შორის მსგავსების ამოცნობასა და დადგენას. ზემოთ აღნიშნული პრობლების გადასაწყვეტად ამჟამად უკვე საკამოდ ბევრი ალგორითმია შემოთავაზებული, ალგორითმები ადგენენ დისტანციას სიტყვებს შორის, თუ რამდენადაა ერთი სიტყვა დაშორებული მეორესაგან კონცეპტუალურად. უმეტესობა ამდაგავრი ალგორითმებისა აგებულია Perl-ისა და Python-ისა პლათფორმებზე (WordNet:Similarity]; NLTK), ასევე, სიტყვათაშორისი დამოკიდებულების დასადგენად გამოიყენება Java-ს პლათფორმა - ADW.

4. Front-End დეველოპმენტი

Front-End განვითარება (development) გახლავთ საიტის საჯარო (public) ნაწილის პლათფორმა, რომელსაც ეკონტაქტება მომხმარებელი. ეს ნაწილი, როგორც წესი, სრულდება კლიენტურ მხარეზე (ბრაუზერში).

Front-End-ის ასპექტებს მიეკუთვნება: საიტის მაკეტის და თარგის შექმნა, ასევე სხვადასხვა სკრიპტების მიბმა, რომლებიც უზრუნველყოფენ ვიზუალიზაციას და web-ანიმაციას.

ამ ნაწილის ძირითადი ინსტრუმენტებია: HTML, CSS, Javascript.

Javascript-ი Front-End-ის მთავარი ენაა, რომელზეც იწერება სამომხმარებლო ინტერფეისის პროგრამული კოდი. არსებობს ბევრი დამატებითი ინსტრუმენტი, რომლიც რუტინული სამუშაოს ნაწილის ავტომატიზაციას ახდენს: Sass/SCSS, jQuery, LESS, AngularJS, Bootstrap, Prototype, Ember.js, Backbone, React.js, Grunt და Gulp.

4.1. MVC

Model View Controller (პირველად აღწერილ იქნა 1978 წელს) - პროგრამის მონაცემების, სამომხმარებლო ინტერფეისის და მმართავი ლოგიკის განცალკევების სქემაა (Design Pattern), რომელშიც გვაქვს 3 კომპონენტი: **მოდელი**, **წარმოდგენა** და **კონტროლერი.** თითოეული კომპონენტის მოდიფიცირება შესაძლებელია სხვა კომპონენტებისგან დამოუკიდებლად. განვიხილოთ თითოეული მათგანი:

- **მოდელი** (Model) წარმოადგენს მონაცემებს და მეთოდებს მათთან სამუშაოდ: მოთხოვნებს (request) მონაცემთა ბაზაში, კორექტულობის შემოწმებას. მოდელი არ არის დამოკიდებული არც წარმოდგენაზე - არ იცის როგორ ხდება მონაცემთა ვიზუალიზაცია, არც კონტროლერზე გააჩნია მომხმარებელთან ურთიერთქმედების წერტილები.

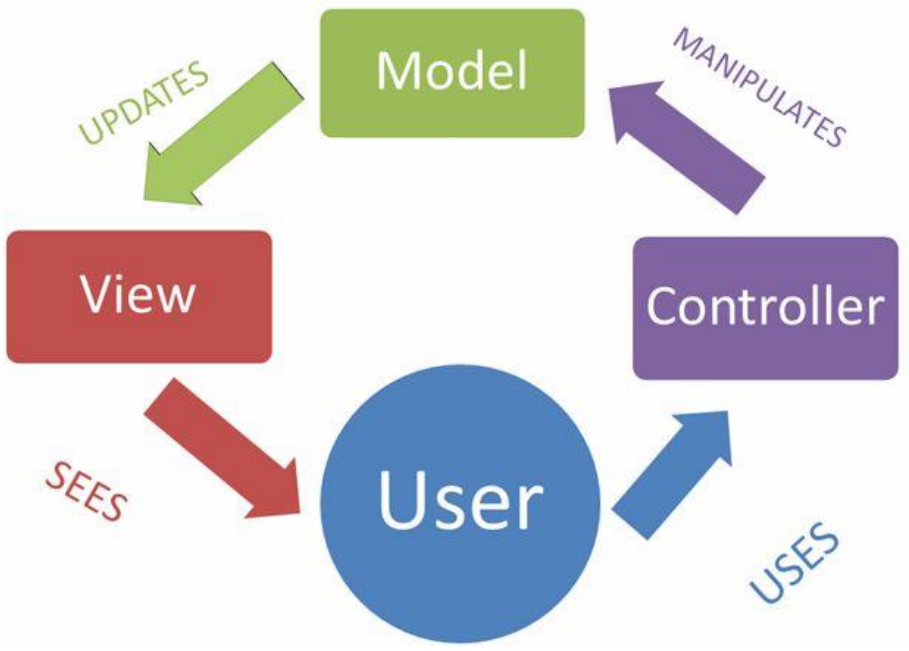
  ასევე, ერთ მოდელს შესაძლოა ჰქონდეს რამდენიმე წარმოდგენა;

- **წარმოდგენა** (View) პასუხისმგებელია მოდელიდან საჭირო ინფორმაციის მიღებაზე (და ამ ინფორმაციის გადაგზავნაზე მომხმარებელთან). წარმოდგენამ შეიძლება იმოქმედოს მოდელის მდგომარეობაზე;
- **კონტროლერი** (Controller) უზრუნველყოფს მომხმარებელსა და სისტემას შორის „კავშირს“. იყენებს მოდელს და წარმოდგენას საჭირო მოქმედების შესასრულებლად.

ამ კონცეფციის გამოყენების ძირითადი მიზანია ბიზნეს-ლოგიკისა (მოდელი) და წარმოდგენის განცალკევება. ასეთი განცალკევების გამო იზრდება კოდის ხელმეორედ გამოყენების (reusability) შესაძლებლობა. MVC-ს გამოყენება სასარგებლოა იმ შემთხვევებში, როდესაც მომხმარებელს სურს ერთი და იმავე მონაცემების ნახვა ერთდროულად რამდენიმე კონტექსტში.

ჩვენ შევეცადეთ წარმოგვედგინა სქემები, რომლებიც საშუალებას მოგვცემს უფრო ნათალად წარმოვიდგინოთ სამუშაო პროცესი:

სქემატური წარმოდგენა:



4.2. AngularJS

AngularJS – Javascript ფრეიმვორკი (open source - ღია კოდი), რომელიც იყენებს MVC ნიმუშს (pattern). განკუთვნილია SPA[1] შექმნისთვის.

ფრეიმვორკი მხარს უჭერს ისეთ ფუნქციონალს, როგორიცაა: Ajax, DOM-ით მანიპულირება, ანიმაცია, თარგები (templates), მარშრუტიზაცია (routing) და ა.შ. გამოიყენება საკმასოდ დიდი რაოდენობ ვებ აპლიკაციაში და არის Javascript-ის ერთ-ერთი ყველაზე პოპულარული ფრეიმვორკი.

AngularJS არ არის დამოკიდებული HTML- ზე, მეტიც - იგი მას აუმჯობესებს, სძენს ახალ თვისებებს (დირექტივების საშუალებით), რადგანაც საქმე გვაქვს მონაცემების დაფიქსირების ორ შრესთან:

მოდელსა (**Model**) და წარმოდგენასთან (**View**).

ამ შრეებს შორის კავშირი ორმხრივია (Two way data-binding) - მოდელში განხორციელებული ცვლილებები გავლენას ახდენს წარმოდგენაზე (ინფორმაცია, რომელიც გამოდის მომხმარებლის ეკრანზე) და პირიქით.

კონტროლერი (**Controller**) წარმოდგენასა და მოდელს შორის შუამავალია - იგი იღებს მოთხოვნას მომხმარებლისგან, აანალიზებს მას და შემდგომი დამუშებისთვის გადასცემს სისტემის შესაბამის რგოლს. კონტროლერის შექმნა შეგვიძლია შემდეგნაირად:

```
angular.module('wordnet', [])
        .controller('HomeController', function () {
                // კოდი
        });
```

კონტროლერს გადაეცემა ორი არგუმენტი: მისი მაიდენტიფიცირებელი სახელი და შესასრულებელი ფუნქცია, რომელიც წარმოადგენს კონტროლერის ტანს.

[1] SPA (Single Page Application) - ვებ აპლიკაცია ან საიტი, რომლის ფუნქციონირებისათვის საჭირო კოდი (HTML, CSS და JavaScript) სერვერიდან ან ერთბაშად გადმოიტვირთება კლიენტის მხარეს, ან ჩაიტვირთება დინამიურად, როგორც წესი, მომხმარებლის ქმედებებიდან გამომდინარე. ხშირად აღნიშნული პროცესი სრულდება ფონურ რეჟიმში, ვებგვერდის გადატვირთვის გარეშე.

დირექტივები საშუალებას გვაძლევენ, შევქმნათ სამომხმარებლო HTML ტეგები და ატრიბუტები. აღვწეროთ რამდენიმე მათგანი, რომლებიც უკვე არსებობენ ფრეიმვორკში (built-in directives):

- ng-app - აპლიკაციის ინიციალიზაცია, მაგალითად:

  <html **ng-app**=”wordnet”> <!-- ვებ გვერდის ტანი --> </html>

- **ng-model** - უზრუნველყოფს მონაცემთა ორმხრივ კავშირს. თუ შეიცვლება ელემენტის შიგთავსი, ანგულარი შეცვლის მოდელის მნიშვნელობას. ანალოგიურად, თუ შეიცვლება მოდელის მნიშვნელობა, ანგულარი განაახლებს ტექსტს ელემენტის შიგნით.
- **ng-class** - დინამიურად აბამს ერთ ან რამდენიმე CSS კლასს HTML ელემენტზე.
- **ng-controller** - განსაზღვრავს კონტროლერს. მაგალითად:
  <div **ng-controller**=”HomeController"> <!-- … -->
- **ng-repeat** - იმეორებს გარკვეულ HTML თეგებს, ყველა ელემენტისთვის კოლექციიდან (კოლექცია შეიძლება წარმოაგდენდეს მასივს). მაგალითად:

  <ul **ng-repeat**=”x in records”>
  {{ x }}

- **ng-show** და **ng-hide** - აჩენს ან მალავს ელემენტს (ან ელემენტთა ჯგუფს), ლოგიკური გამოსახულებიდან გამომდინარე.

4.2.1. მარშრუტიზაცია

მარშრუტიზაცია გახლავთ აპლიკაციის გვერდებს შორის გადასვლა.

AngularJS აპლიკაციის როუტინგის უზრუნველყოფა შეგვიძლია 2 გზით: ngRoute ან ui-router -ის გამოყენებით, რომელიც ავირჩიეთ ჩვენი პროექტისათვის.

ngRoute - მოდული, შექმნილი ანგულარის ჯგუფის მიერ. წარმოადგენს ამ ფრეიმვორკის ადრინდელ ნაწილს.

ui-router - ფრეიმვორკი, შექმნილი AngularJS პროექტის „გარეთ“. შექმნის მიზანს წარმოადგენდა როუტინგ შესაძლებლობების გაუმჯობესება და გაზრდა.

ძირითადი ცნება, რომლითაც ოპერირებს ui-router არის State, ხოლო ngRoute შემთხვევაში - Route.

State	**Route**
ოპერირებს კონკრეტული ადგილით აპლიკაციაში	ოპერირებს URL მისამართით
იერარქიული დამოკიდებულება - ნებისმიერ state-ს შეიძლება ჰყავდეს „შვილები“	ერთდონიანი დამოკიდებულება
თითოეულ state-ს აქვს სახელი, რომლითაც შეიძლება მასზე მიმართვა	Route-ის სახელი არის მისი URL-ი
ნავიგაცია სახელით ან URL-ით	ნავიგაცია მხოლოდ URL-ით
მრავალი წარმოდგენა ui-view	მხოლოდ ერთი წარმოდგენა ng-view

ჩვენს პროექტში დაკონფიგურირებული routing-ი გამოიყურება შემდეგნაირად:

```
angular
    .module('wordnet.routes', ['ui.router'])
    .config(config);

function config ($stateProvider, $urlRouterProvider) {

    $stateProvider
    .state('search', {
        url: '/search',
        templateUrl: 'app/views/homeView.html',
        controller: 'HomeController'
    })
    .state('search.word', {
        url: '/:word',
        templateUrl: 'app/views/wordView.html',
        controller: 'WordController'
    })
    .state('about', {
        url: '/about',
        templateUrl: 'app/views/aboutView.html',
        controller: 'AboutController'
    });

    // ყველა სხვა url შემთხვევაში გადადი /search -ზე
    $urlRouterProvider.otherwise('/search');
}
```

4.2.2. $http სერვისი

ზოგადად, სერვისი ანგულარში წარმოადგენს ფუნქციას, რომელიც ხელმისაწვდომია თქვენი აპლიკაციისათვის.

$http სერვისი ურთიერთქმედებს დაშორებულ (remote) სერვერებთან. ეს სერვისი აგზავნის მოთხოვნას (request) სერვერზე და აბრუნებს პასუხს (response).

პასუხის ობიექტს ექნება შემდეგი თვისებები:

- **data** - პასუხის ტანი (რომელშიც, წარმატების შემთხვევაში, გვექნება დაბრუნებული მონაცემები);
- **status** – HTTP პასუხის სტატუს კოდი (200 – OK, 304 – Not modified, 404 – Not found, …);
- **headers** – დამატებითი ინფორმაცია დაბრუნებული სერვერიდან;
- **config** - კონფიგურაციის ობიექტი;
- **statusText** – HTTP პასუხის სტატუს ტექსტი.

ჩვენ შემთხვევაში, კონკრეტული სიტყვისთვის ინფორმაციის დასაბრუნებლად სერვერიდან გამოიყენება შემდეგი სერვისი:

```
var config = {
    endpoint: 'http://127.0.0.1:80/',
    path: 'api/'
};

angular.module('wordnet.core')
.service('WordService' , ['$http', function($http) {

       this.getWordInfo = function (word) {

           return $http.get(config.endpoint +
     config.path + word);

       };

}]);
```

4.3. მონაცემთა ვიზუალიზაცია

ვიზუალიზაცია გახლავთ სხვადასხვა ინფორმაციის წარმოდგენა სურათების, გრაფიკების, ცხრილების, ნახაზების საშუალებით. ითვლება, რომ ადამიანი ინფორმაციის 90% აღიქვამს ვიზუალურად (მხედველობის საშუალებით).

ვიზუალიზაციის დროს გამოიყენება ისეთი დარგებისა და დისციპლინების მეთოდური ინსტრუმენტები, როგორიცაა ინფორმატიკა, სტატისტიკა და ა.შ.

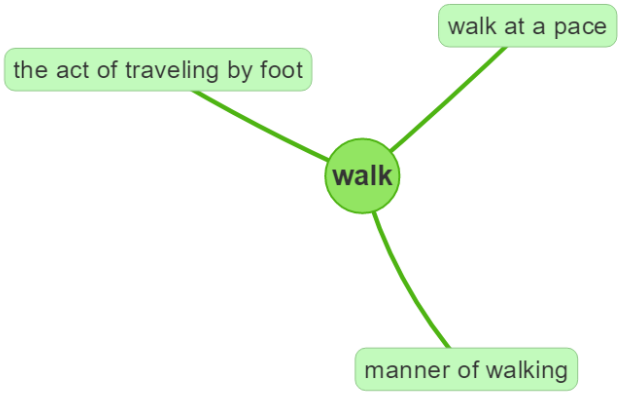
WordNet მონაცემთა ბაზის ვიზუალიზაცია შესაძლებელია გრაფის საშუალებით, თითოეული სიტყვა წარმოადგენს გრაფის კვანძს (წვეროს), წიბოები კი მეტყველების ნაწილის ინფორმაციის მატარებლები არიან.

არსებობს ბევრი Javascript ბიბლიოთეკა, რომელიც საშუალებას გვაძლევს ვიზუალიზაცია გავუკეთოთ ჩვენ ინფორმაციას. მათ შორის: Chartlist.js, Timesheet.js, Vis.js, Plottable.js, D3.js.

ჩვენ არჩევანი გავაჩერეთ Vis.js ბიბლიოთეკაზე, რადგან მას შეუძლია დიდი რაოდენობის დინამიურ მონაცემებთან მუშაობა, მანიპულირება, ურთიერთქმედება. ასევე, სვხებთა შედარებით იგი მარტივია გამოყენებისა და მოხმარებისათვის.

ეს ბიბლიოთეკა შედგება შემდეგი კომპონენტებისაგან: Timeline, Network, Graph2D, Graph3D. Network კომპონენტის საშუალებით შეგვიძლია გრაფების ვიზუალიზაცია, ამიტომ ბიბლიოთეკის ზუსტად ამ ნაწილს ვიყენებთ პროექტში. ეს კომპონენტი საშუალებას იძლევა სურბილისამებრ ვცვალოთ ფონტები, ზომები, ფერები.

მაგალითად:



ცენტრში გვაქვს სიტყვა, რომელიც დაკავშირებულია თავის მნიშვნელობებთან. წიბოს მწვანე ფერი კი ნიშნავს, რომ *walk* არის არსებითი სახელი.

ფერთა კოდირებას და მაგალითებს უფრო დაწვრილებით განვიხილავთ შედეგებში.

განხილული მაგალითის სკრიპტს აქვს შემდეგი სახე:

```
// წვეროთა მასივის შექმნა
var nodes = new vis.DataSet([
             {id: 1, label: 'walk', shape: 'circle', color: '#7BE141',
font: {size: 16}},
             {id: 2, label: 'manner of walking', shape: 'box',
color:'#C2FABC'},
             {id: 3, label: 'the act of traveling by foot', shape:
'box', color:'#C2FABC'},
             {id: 4, label: 'walk at a pace', shape: 'box',
color:'#C2FABC'}

]);

// წიბოთა მასივის შექმნა
var edges = new vis.DataSet([
  {from: 1, to: 2},
  {from: 1, to: 3},
  {from: 1, to: 4}
]);

// network-ის შექმნა
var container = document.getElementById('mynetwork');

// ვინახავთ მონაცემებს vis ფორმატში
var data = {
   nodes: nodes,
   edges: edges
};

var options = {};

// ინიციალიზაცია
var network = new vis.Network(container, data, options);
```

4.4. Gulp

Gulp - ეს არის ინსტუმენტი, რომელიც საშუალებას გვაძლევს ავტომატიზაცია გავუკეთოთ ზოგიერთ ამოცანას, მაგალითად: CSS და Javascript ფაილების შეკრება და მინიფიკაცია,

ტესტების გაშვება, ბრაუზერის გადატვირთვა და ა.შ. ამიტომ ცხადია, რომ ეს ინსტრუმენტი აჩქარებს და და ოპტიმიზაციას უკეთებს ვებ აპლიკაციის შექმნის პროცესს.

Gulp აწყობილია Node.js -ზე, გამშვები ფაილი იწერება Javascript-ის პლათფორმაზე. ინსტრუმენტს გააჩნია პლაგინების დიდი რაოდენობა, რომელთა მოძებნა შეიძლება ოფიციალურ ვებ-გვერდზე.

სანამ გადავალთ Gulp-ის განხილვაზე, სასურველია გავერკვეთ ნაკადებში (Streams) და მილებში (Pipe).

ნაკადი შეგვიძლია წარმოვიდგინოთ როგორც მონაცემთა ნაკრები, რომლის გადაცემა ხდება ფრაგმენტებით. ამიტომ მონაცემების დამუშავება შეგვიძლია მათი დაგროვებისთანავე (და არა მაშინ, როცა ისინი გადაეცემა მთლიანად).

არსებობს ნაკადების რამდენიმე ტიპი:

- **Readable** - კითხვის ნაკადი, გამოიყენება წაკითხვის ოპერაციისთვის;
- **Writable** - ჩაწერის ნაკადი, გამოიყენება ჩაწერის ოპერაციისთვის;
- **Duplex** - გამოიყენება როგორც წაკითხვის, ასევე ჩაწერისთვის;
- **Transform** - დუპლექსური ნაკადების კერძო შემთხვევა, გამომავალი ინფორმაცია დამოკიდებულია იმაზე, თუ რა მიეწოდება შესასვლელში.

მილი - მექანიზმი (მეთოდი), რომელიც აგზავნის მონაცემებს კითხვის ნაკადიდან (შეიძლება იყოს დუპლექსურიც) მითითებულ ადგილზე. ამით ეს მექანიზმი აერთიანებს ნაკადებს.

განვიხილოთ მაგალითი (რომელიც ახდენს js ფაილს მინიფიკაციას და ინახავს 'build' საქაღალდეში):

```
var gulp = require('gulp'),
    uglify = require('gulp-uglify');

gulp.task('minify', function () {
    gulp.src('js/app.js')
        .pipe(uglify())
        .pipe(gulp.dest('build'));
});
```

პირველ ორ ხაზზე ხდება პაკეტების ჩართვა. შემდეგ ვქმნით ამოცანას (task), რომელიც ორპარამეტრიანია, პირველი - ამოცანის დასახელება ('minify'), მეორე კი - ფუნქცია, რომელშიც აღიწერება ამოცანის ქმედებები. ფუნქცია gulp.src() მიიღებს სტრიქონს, რომელიც

შეესაბამება ფაილს ან ფაილების ჯგუფს და შექმნის ამ ფაილების ობიექტების ნაკადს. შემდეგ ეს ობიექტები გადაეცემა ფუნქციას uglify(), რომელიც დააბრუნებს მინიფიცირებული ფაილის ახალ ობიექტებს. საბოლოოდ, ფუნქცია gulp.dest() შეინახავს შეცვლილ ობიექტებს.

ამოცანა შეიძლება შექმნილი იყოს სხვა ამოცანების შესასრულებლად, მაგალითად:

```
gulp.task('build', ['css', 'js']);
```

ამოცანა 'build' არსულებს 'css' და 'js' ამოცანებს. უნდა აღინიშნოს, რომ ამ ფორმით ჩაწერილი ამოცანების მიმდევრობა არ ნიშნავს იმას, რომ 'css' ამოცანა აუცილებლად შესრულდება 'js' ამოცანამდე ან პირიქით.

4.4.1. Gulpfile.js შიგთავსი

ჩვენს gulpfile-ს აქვს შემდეგი სახე:

```
var gulp        = require('gulp');
var concat      = require('gulp-concat');
var rename      = require('gulp-rename');
var concatCss   = require('gulp-concat-css');
var uglify      = require('gulp-uglify');
var rename      = require('gulp-rename');
var cleanCSS    = require('gulp-clean-css');
var clean       = require('gulp-clean');

// Files paths
var jsFiles = 'public/app/js/',
    jsLibs = 'public/assets/libs/',
    dest = 'dist/app/';

// Concat + uglify app scripts
gulp.task('scripts', function() {
    return gulp.src([
          jsFiles + 'app.module.js',
          jsFiles + 'app.core.js',
          jsFiles + 'app.routes.js',
          jsFiles + 'homeController.js',
          jsFiles + 'wordService.js',
          jsFiles + 'wordController.js',
          jsFiles + 'aboutController.js'
        ])
       .pipe(concat('bundle.js'))
```

```
        .pipe(uglify({mangle: false}))
        .pipe(gulp.dest(dest+"js"));
});

// Concat + uglify libs scripts
gulp.task('libs', function() {
   return gulp.src([
          jsLibs + 'angular.min.js',
          jsLibs + 'angular-ui-router.min.js',
          jsLibs + 'jquery-3.1.1.min.js',
          jsLibs + 'bootstrap.min.js',
          jsLibs + 'vis.js',
          jsLibs + 'angular-vis.js',
          jsLibs + 'html2canvas.min.js',
                jsLibs + 'bootstrap-typeahead.min.js'
       ])
       .pipe(concat('libs.js'))
       .pipe(uglify({mangle: false}))
       .pipe(gulp.dest(dest+"js"));
});

// Concat + uglify css
gulp.task('css', function () {
 return gulp.src([
     'public/assets/css/bootstrap.min.css',
     'public/assets/css/vis.css',
     'public/assets/css/styles.css'
    ])
   .pipe(concatCss("bundle.css"))
   .pipe(cleanCSS({}))
   .pipe(gulp.dest(dest+'css'));
});

// Copy rest files
gulp.task('copy-fonts', function() {
   return gulp.src('public/assets/fonts/*')
            .pipe(gulp.dest(dest+'fonts'));
});

gulp.task('copy-img', function() {
   return gulp.src('public/assets/img/**/*')
            .pipe(gulp.dest(dest+'img'));
});

gulp.task('copy-html', function() {
   return gulp.src('public/app/views/*')
            .pipe(gulp.dest(dest+'views'));
});
```

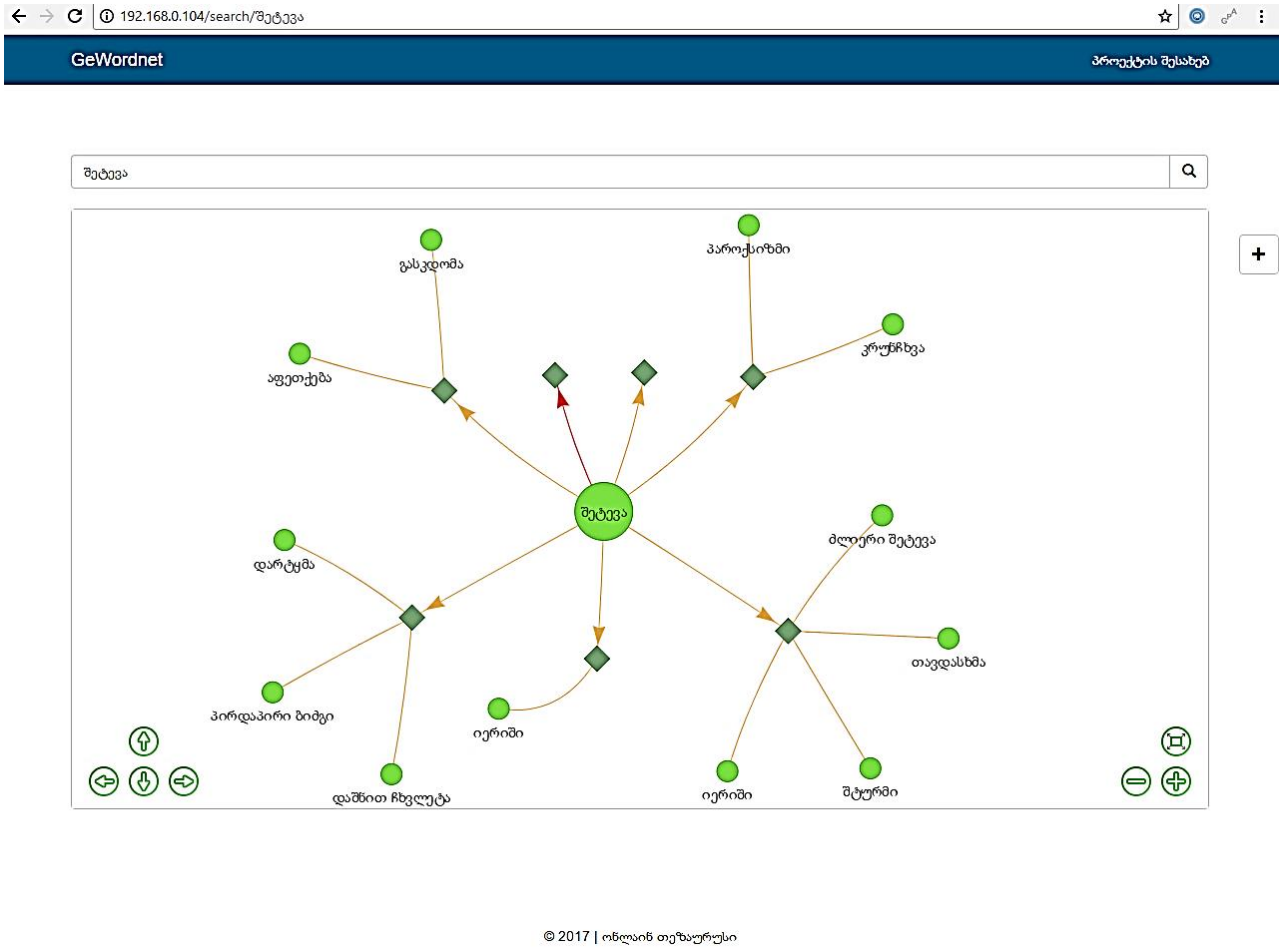
```
gulp.task('copy-index', function() {
   return gulp.src('public/index1.html')
            .pipe(rename('index.html'))
            .pipe(gulp.dest('dist'));
});

// Clean destination directory
gulp.task('clean', function () {
   return gulp.src('dist', {})
      .pipe(clean());
});

// Default task
gulp.task('default', [
          'scripts',
          'libs',
          'css',
          'copy-html',
          'copy-fonts',
          'copy-img',
          'copy-index'
]);
```

5. შედეგები

სამომხმარებლო ინტერფეისი გამოიყურება შემდეგნაირად:

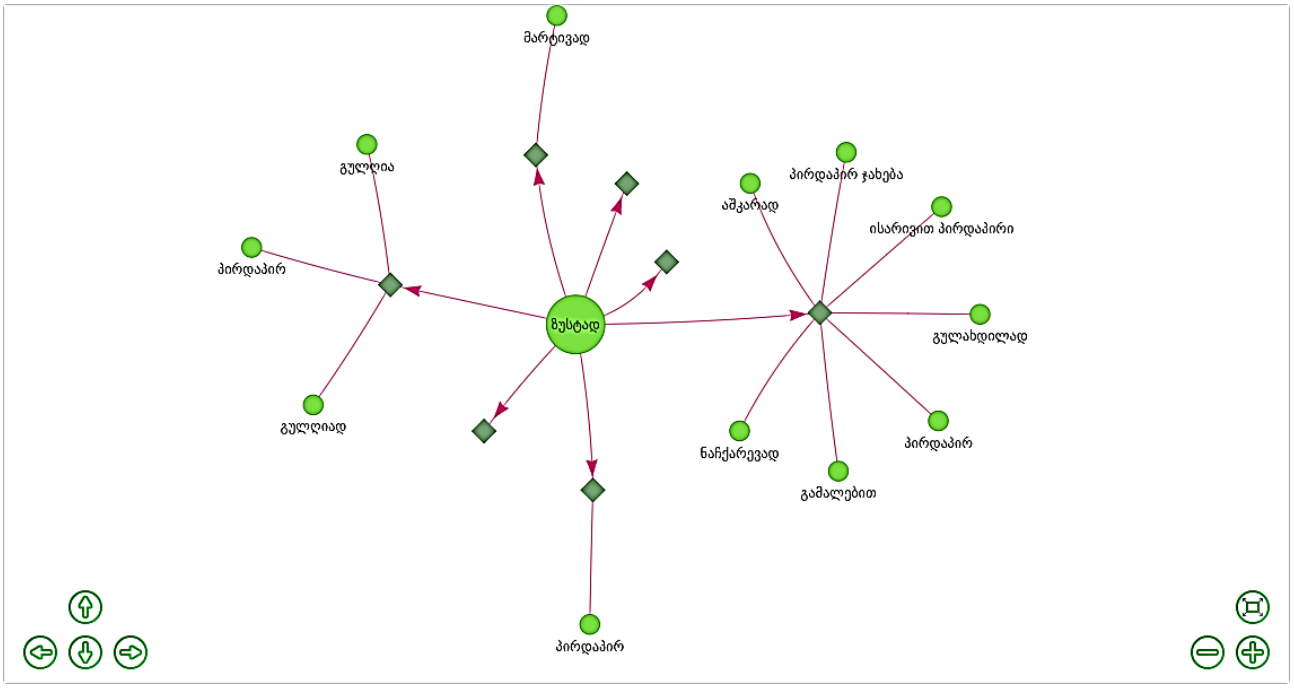


როგორც უკვე ზემოთ აღვნიშნეთ ვორდნეტის ვიზუალიზაცია მოვახდინეთ გრაფის საშუალებით. ფუნქციონალური ღილაკებით (‘+’ , ‘-‘ და ა.შ.) შეგვიძლია გრაფის გადიდება/დაპატარავება, ასევე მისი ნორმალიზაცია.

გრაფზე რომბებით აღნიშნულია მოცემული სიტყვის („შეტევა“) მნიშველობები შესაბამის სინონიმურ რიგში. ეს მნიშნველობები ჩნდება რომბში მაუსის მიტანით. დანარჩენი სიტყვები მიერთებული ამ მნიშვნელობაზე წარმოადგენენ სინონიმური რიგის ელემენტებს. ანუ გაერთიანებულები არიან ერთი მნიშვნელობაში. მეტყველების თითოეულ ნაწილს შეესაბამება წიბოების სხვადასხვა ფერები. როგორც სურათზე ჩანს: სინსეტების (სინონიმური რიგი) უმეტესობა აღნიშნულია ნარინჯისფერით, რომელსაც შეესაბამება არსებითი სახელი.

ფერთა კოდირება შემდეგნაირია:

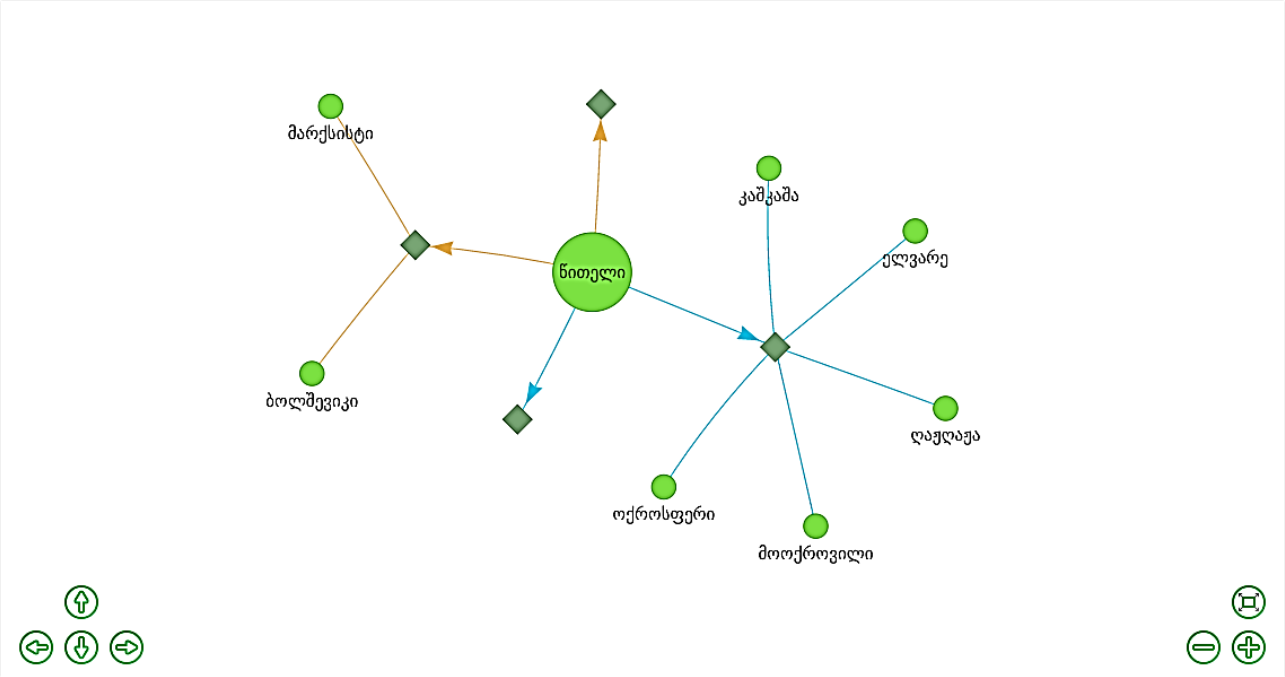
მაგალითი 2: **სიტყვა „ზუსტად“.** ვარდისფერი წიბოები შეესაბამებიან ზნიზედას. ასევე, ჩანს რომ „*ზუსტად*“ დაკავშირებულია შვიდ სინონიმურ რიგთან, სურათზე მარჯვენა რიგი შეიცავს სიტყვებს: აშკარად, გულახდილად, ნაჩქარევად და ა.შ.



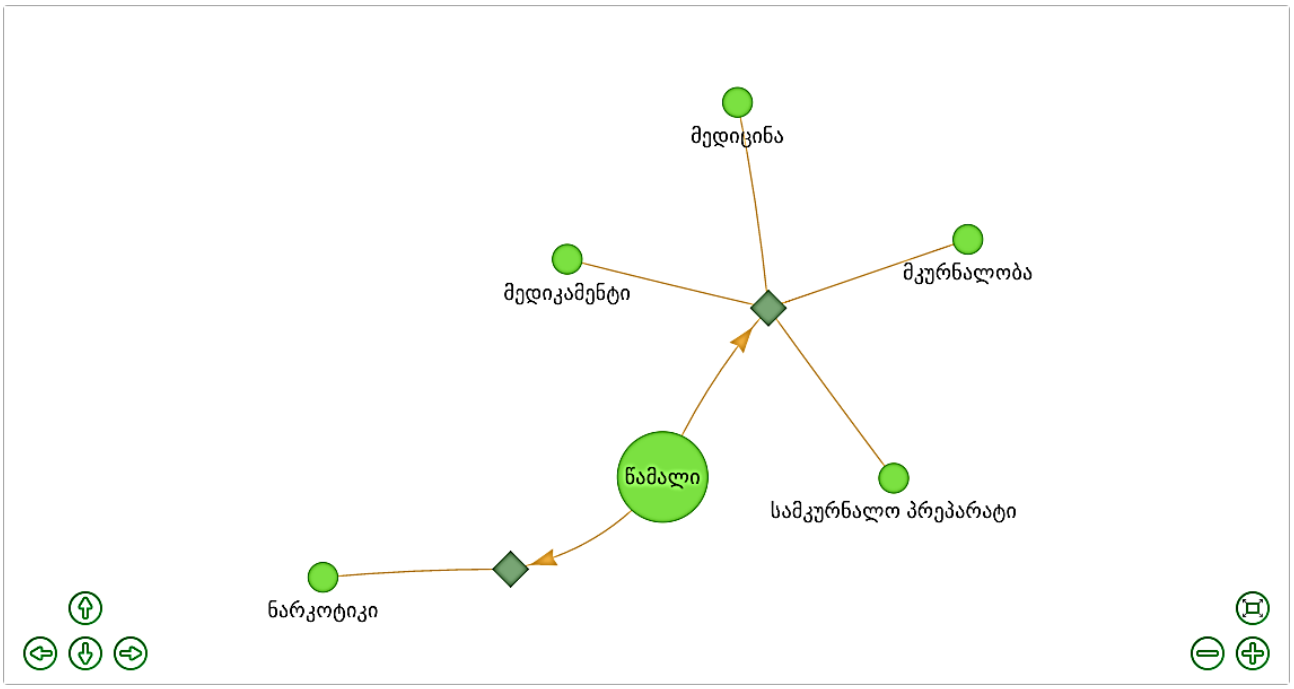
ძებნის შედეგები სხვადასხვა სიტყვებისთვის:

რადგანაც გრაფის ფორმა არ იცვლება სხვა მაგალითებში, ამიტომ დეტალურ აღწერას აღარ განვიხილავთ, რადგან ზემოთ განხილული მაგალითების ანალოგიურია.

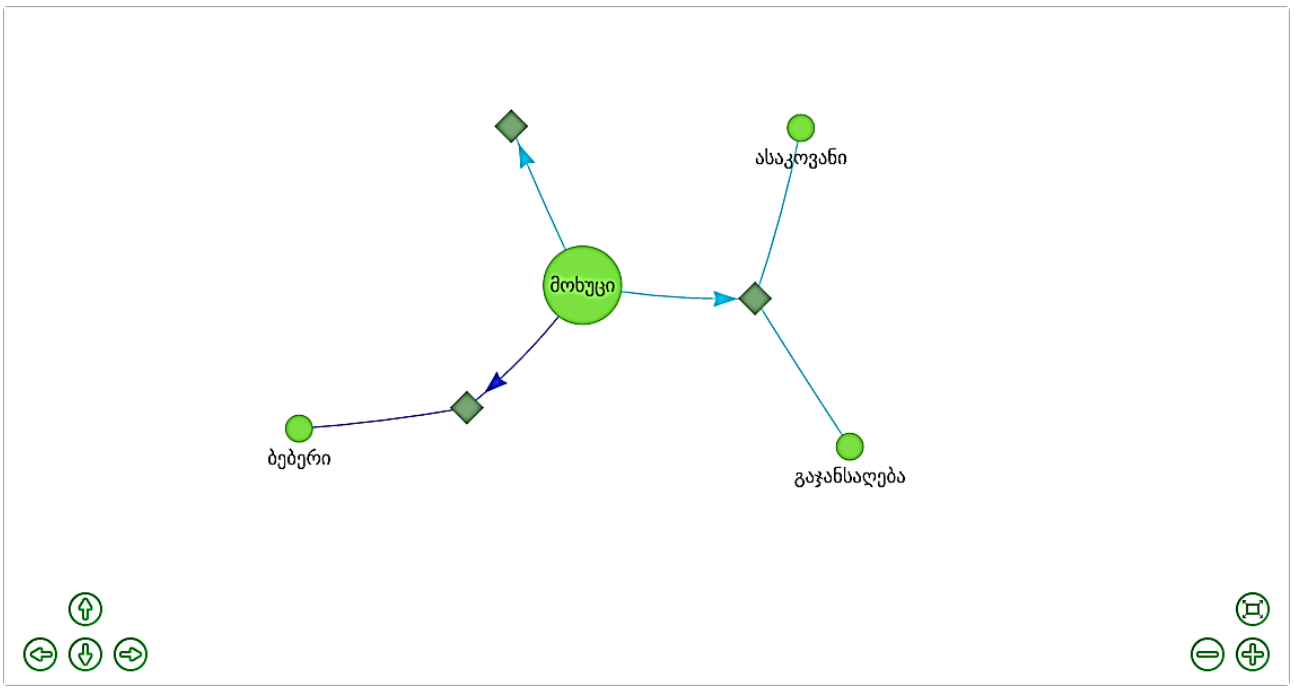
მაგალითი 3: სიტყვა „წითელი“.



მაგალითი 4: სიტყვა „წამალი“.



მაგალითი 5: სიტყვა „მოხუცი“.



6. დასკვნა

როგორც მოგეხსენებათ, ჩვენი ნაშრომის მთავარი თემატიკა გახლდათ ქართული სიტყვათა ქსელის აპლიკაციის შექმნა. ნაშრომის პირველ ნაწილში აღვწერეთ საკმაოდ პოპულარული და ფართოდ გამოყენებადი ინგლისური WordNet-ი, რომელიც საორიენტაციოდ გამოვიყენეთ ჩვენი აპლიკაციის შექმნისას, ხოლო შემდეგ ნაწილებში გადავედით ჩვენი აპლიკაციის Front-end ნაწილის აღწერაზე.

Front-end -ის ნაწილში ნაჩვენებია აპლიკაციის კლიენტური ნაწილი, აღწერილია ყველა ტექნოლოგია, რომელიც გამოვიყენეთ კლიენტური ნაწილის შექმნისას. ნაშრომის ბოლოს გამოვიტანეთ სიტყვათა ქსელის მუშაობისას, ვიზუალიზირებული რამდენიმე სიტყვის გრაფი და ვაჩვენეთ აპლიკაციის მუშაობის შედეგები.

ჩვენ გვჯერა, რომ ადრე თუ გვიან ასეთი ელექტრონული ლექსიკონები იქნება აყვანილი იმ დონემდე, რომ შესაძლებელი გახდებს ქართულ ენაზე არსებული ტექსტების მეტ-ნაკლებად სრულყოფილი, ავტომატური დამუშავება, რაც იქნება წინ გადადგუმლი ნაბიჯი ხელოვნურ ინტელექტსა და ზოგადად, ინფორმაციულ ტექნოლოგიებში.

7. გამოყენებული ლიტერატურა

- https://en.wikipedia.org/wiki/WordNet
- https://wordnet.princeton.edu
- https://www.fi.muni.cz/gwc2004/proc/105.pdf
- http://www.profguide.ru/professions/front_end_developer.html
- https://ru.wikipedia.org/wiki/AngularJS
- https://github.com/angular-ui/ui-router
- http://visjs.org/
- http://getinstance.info/articles/tools/introduction-to-gulp/
- http://gulpjs.com/