ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტი
ზუსტ და საბუნებისმეტყველო მეცნიერებათა ფაკულტეტი

კომპიუტერული მეცნიერება

რუდოლფ ერემიან

ქართული ელექტრონული სიტყვათა ქსელის სერვერული ნაწილის და მონაცემთა ბაზის შექმნის პროცესის აღწერა

ნაშრომი შესრულებულია ინფორმატიკის ბაკალავრის აკადემიური ხარისხის მოსაპოვებლად

ხელმძღვანელი: ლიანა ლორთქიფანიძე

ასისტენტ პროფესორი

თბილისი
2017

სარჩევი

1. ანოტაცია

WordNet-ი ინგლისური სიტყვებისგან შედგენილი დიდი ლექსიკონია. WordNet-ი ფართოდ გამოიყენება სხვადახვა სფეროში, მაგალითად ხელოვნურ ინტელექტში. WordNet-ს საკმაოდ დიდ როლს თამაშობს ხელოვნურ ინტელექტზე დამფუძვნებულ აპლიკაციებში: ჩატბოტებში, სენტიმენტ ანალიზის პროგრამებში, მანქანურ თარგმნაში. ზუსტად ამიტომაც ჩვენ გადაწყვიტეთ ქართული სიტყვათა ქსელის შექმნა. ამ ნაშრომში აღწერილია ქართული სიტყვათა ქსელის კლიენტური ნაწილის ან ე. წ. Back-end-ის და მონაცმეთა ბაზის შექმნის პროცესი, ხოლო ნაშრომის დასკვნით ნაწილში ჩვენ შევეცდებით წარმოავდგინოთ პროგრამის მუშაობის შედეგად მიღებული სიტყვათა ქსელის ვიზუალიზაცია.

WordNet is a large lexical database of English. WordNet applied in different kind of applications: Sentiment Analysis, Machine Translation, Question Answering Systems. Our main goal was development of WordNet for Georgian language, which will be helpful for creating an AI-based tools and applications with processing texts on Georgian language. In this paper, we described the process of development back-end part of application and database architecture. In the results section, we are showing visualized WordNet for Georgian words, generated by our application.

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.

სერვერული ნაწილი შემუშავებულია NodeJS ტექნოლოგიის გამოყენებით, ხოლო სიტყვების შესანახად გამოიყენება მონაცემთა ბაზა MySQL.

აღსანიშნავია, რომ ეს არის პირველი მცდელობა ასეთი ტიპის ქსელის შექნისა ქართული ენისთვის. მიგავჩნია, 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

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.

სიტყვათა ქსელი გამოყენება შესაძლებელია ყველა იმ მიზნისათვის, რასაც დღესდღეობით ემსახურება ბუნებრივი ენის კომპიუტერული მოდელირება, კერძოდ საუბარი გვაქვს მნიშნელოვან ლინგვისტურ ამოცანებზე:

1. მანქანური ლექსიკონის შედგენისას ინფორმაციის ლინგვისტური და ავტომატიზირებული დამუშავება;

2. მანქანაში ტექსტის შეყვანისას დაშვებული შეცდომების აღმოჩენა და შემდეგ მათი გამოსწორება ავტომატიზირებული პროცესების საშუალებით;

3. დოკუმენტებისა და ინფორმაციული მოთხოვნების ავტომატური ინდექსირება;

4. დოკუმენტების ავტომატური კლასიფიკაცია და რეფერირება;

5. ერთენოვან და მრავალენოვან მონაცემთა ბაზებში ინფორმაციის მოძიების პროცესის ორგანიზება;

6. ტექსტების ერთი ენიდან მეორეზე თარგმნის მანქანური სისტემების მოდელირება;

7. ისეთი ლინგვისტური პროცესების შექმნა, რომლებიც გაუადვილებს მომხმარებელს ავტომატიზირებულ ინტელექტუალურ საინფორმაციო სისტემებთან (იგულისხმება ხელოვნური ინტელექტი, რობოტები ანუ ექსპერიმენტული სისტემები) ურთიერთობას ბუნებრივ ან მასთან ძალზედ დაახლოებულ ენაზე;

8. არაფორმალიზებული ტექსტებიდან ფაქტობრივი ინფორმაციის ამოღება.

3.1. შეცდომების აღმოჩენა და გასწორება

საინფორმაციო ცენტრების პრაქტიკულ სამუშაოებში მნიშვნელოვან ადგილს იკავებს მანქანაში ტექსტის შეყვანისას დაშვებული შეცდომების აღმოჩენისა და შემდეგ მათი გამოსწორების პრობლემა. ეს ამოცანა შესაძლოა შემდეგნაირად ჩამოვაყალიბოთ - უნდა ჩატარდეს საკონტორლო ტექსტის ორთოგრაფიული, სინტაქსური და სემანტიკური დამუშავება.

ორთოგრაფიული დამუშავება შესაძლებელია მორფოლოგიური ანალიზის გამოყენებით. ამ შემთხვევაში გამოიყენება მანქანური ლექსიკონის ის ეტალონური ვარიანტი, სადაც მოცემულია სიტყვათა ძირები. სიტყვის ორთოგრაფიული სისწორის კონტროლირებისას ტექსტი ექვემდებარება მორფოლოგიურ ანალიზს. თუ ტექსტში შემავალი სიტყვების ძირები ემთხვევა ძირებს ეტალონი- ლექსიკონიდან, მაშინ სიტყვა ჩაითვლება სწორად. თუ ტექსტში შემავალი სიტყვის ძირი არ ემთხვევა ეტალონ-ლექსიკონში დაფიქსირებულ არც ერთ ძირს, მაშინ უკვე თავად ადამიანი-ავტორი განიხილავს ამ სიტყვის მიკროკონტექსტს. ადამიანს ეძლევა საშუალება აღმოაჩინოს და შემდეგ გაასწოროს მოცემული, გახაზული (ხშირად წითელი წყვეტილი ხაზით) სიტყვა და შესაბამისად, პროგრამა შეიყვანს ამ შესწორებას კონკრეტულ ტექსტში.

ტექსტის სინტაქსური კონტროლის ამოცანა შეცდომების აღმოჩენისა და გასწორების პროცესში უფრო მეტად რთულია, ვიდრე მისი ორთოგრაფიული კონტროლი. ტექსტის სინტაქსური დამუშავების სისწორის შეფასების ამოცანა ჯერჯერობით არ არის სრულყოფილად გადაწყვეტილი. ამის მიუხედავად, ტექსტის ნაწილობრივი

სინტაქსური კონტორლის განხორციელება სრულიად შესაძლებელია. ამ პროცესის ჩასატარებლად შემოთავაზებულია ორი მიმართულება:

1. შევადგინოთ შეძლებისდაგვარად სრულყოფილი სინტაქსური სტრუქტურები და შევიტანოთ ისინი ეტალონ-მანქანურ-ლექსიკონში. ამის შემდეგ შესაძლებლობა გვექნება შემუშავებული სინტაქსური სტრუქტურები შევუდაროთ აკრეფილი ტექსტის სინტაქსურ სტრუქტურებს;

2. შემუშავდეს ტექსტის ელემენტების გრამატიკული შეთანხმებების წესების რთული სისტემა.

შესაძლოა პირველი გზა უფრო პერსპექტიული იყოს, თუმცა იგი არ გამორიცხავს მეორე გზის გამოყენებას. ანუ, ისინი კი არ გამორიცხავენ ერთმანეთს, არამედ ავსებენ. ტექსტების სინტაქსური სტრუქტურა უნდა აღიწეროს სიტყვის გრამატიკული კლასის ტერმინებით (სიტყვის სრული გრამატიკული ანალიზის საშუალებით).

ტექსტის სემანტიკური კონტროლის ამოცანა - აღმოაჩინოს ტექსტში შეცდომები და შემდეგ გაასწოროს ისინი - მიემართება ხელოვნური ინტელექტის პრობლემატიკას. ამ საკითხის გადაჭრა შესაძლებელია მხოლოდ და მხოლოდ ადამიანის აზროვნების პროცესის მოდელირებით. როგორც ჩანს, ამისათვის აუცილებელია მძლავრი ენციკლოპედიური ბაზის შექმნა და ამ ბაზაში არსებული ინფორმაციის მანიპულირების პროგრამები. ეს ამოცანა უფრო ადვილად იქნება მიღწეული, თუ ავიღებთ რაიმე ვიწრო სპეციალობას, თემატიკას ან სემანტიკურ ველს ცალ–ცალკე და შემდეგ ერთიან ბაზაში შევიყვანთ მონაცემებს.

3.2. ინდექსირება

ტექსტის ავტომატიზირებული ძიების სისტემისათვის ტრადიციული საკითხი გახლავთ დოკუმენტებისა და მოთხოვნების ავტომატური ინდექსირება.

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[1]. მათი შევსება შესაძლებელია თესაურუსის გამოყენებით.

3.3. დოკუმენტების ავტომატური ძიება

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.

3.4. კომუნიკაცია ხელოვნურ ინტელექტთან

კომპიუტერული ლინგვისტიკის მნიშვნელოვანი და სამომავლო ამოცანაა ისეთი ლინგვისტური პროცესორების შექმნა და აგება, რომლებიც საშუალებას მოგვცემენ კომუნიკაცია დავამყაროთ ხელოვნურ ინტელექტთან ბუნებრივ ან მასთან ძალზე დაახლოებულ ენაზე.

რადგანაც თანამედროვე ინტელექტუალურ სისტემებში ინფორმაცია ფორმალიზებული სახით ინახება, ლინგვისტურმა პროცესორებმა, რომლებიც თამაშობენ შუამავლის როლს ადამიანსა და მანქანას შორის, უნდა გადაჭრან შემდეგი საკითხები:

1. ბუნებრივ ენაზე არსებული ტექსტები უნდა ჩაიწეროს ფორმალიზებულ ენაზე და, პირუკუ;

2. ფორმალიზებულ ენაზე არსებული ტექსტები უნდა რეალიზდეს ბუნებრივი ენის საშუალებით.

პირველი ამოცანის გადაჭრა ხდება **გადასაცემი** ტექსტის ან შეტყობინების მორფოლოგიური, სინტაქსური და პრაგმატიკულ-კოგნიტიური ანალიზის საშუალებით; მეორე ამოცანის გადაჭრა კი ხდება **გადმოსაცემი** ტექსტის ან შეტყობინების მორფოლოგიური, სინტაქსური და პრაგმატიკულ-კოგნიტიური ანალიზის საშუალებით.

ინფორმაციული მოთხოვნებისა და შეტყობინებების კონცეპტუალური ანალიზი მდგომარეობს, უპირველეს ყოვლისა, მათი შინაარსის გამოვლენაში, შემდეგ კი ეს შინაარსი უნდა ჩაიწეროს ფორმალიზებული ენის საშუალებით. აღნიშნული პროცედურა ტარდება შეტყობინების მორფოლოგიურ-სინტაქსური ანალიზის შემდეგ. კონცეპტუალური სინთეზისას ფორმალიზებული ენა იღებს ვერბალურ სახეს. ამის შემდეგ შეტყობინებას უნდა მიენიჭოს აუცილებელი მორფოლოგიური და სინტაქსური გაფორმება.

3.5. მანქანური თარგმნა

გადავიდეთ მანქანური თარგმნის სისტემებზე. ამ სისტემების შექმნისას აუცილებელია ვიხელმძღვანელოთ უკვე არსებული და თაობების მიერ დაგროვილი ლექსიკონებით, სპეციალური გამოცემებით, რომლებიც დიდი ხნის მანძილზე იქმნებოდა თარგმნის თეორიასა და პრაქტიკაში.

ტრადიციულ ორენოვან და მრავალენოვან ლექსიკონებში სათარგმნი ექვივალენტები ძირითადად ენიჭება სიტყვებს, სიტყვათშეთანხმებებს კი ნაკლებად. სიტყვათშეთანხმებებისათვის სათარგმნი ექვივალენტის მინიჭება უფრო სახასიათოა სპეციალური ტერმინოლოგიური ლექსიკონებისათვის. ასე რომ, გასაკვირი არ არის ის ფაქტი, რომ ტექსტის თარგმნისას მთარგმნელი ხშირად გარკვეულ პრობლემებს აწყდება მაშინ, როდესაც მას უწევს მუშაობა მრავალმნიშვნელობის მქონე სიტყვებით შედგენილ ტექსტებთან.

მოვიყვანოთ, რამდენიმე მაგალითი ინგლისური ენიდან,

1. The **bat** looks like a mouse with a wings - ღამურა ჰგავს თაგვს, რომელსაც ფრთები აქვს;

2. I **like** your style - მე მომწონს შენი სტილი;

3. Dry **wood** burns easily - მშრალი შეშა ადვილად იწვება;

4. The **play** was great - პიესა შესანიშნავი იყო.

იმის მიუხედავად, რომ ზემოთ წარმოდგენილი ინგლისურენოვანი ფრაზები არ ატარებენ იდიომატურ შინაარსს, მათში შემავალ გამუქებულ სიტყვებს მრავალი მნიშვნელობა აქვს, რაც თავის მხრივ პრობლემას ქმნის მანქანური თარგმნისას. იმისათვის რომ ადეკვატური თარგმანი მიღწეული იქნას აუცილებელია ზედმიწევნითი ყურადღება მიექცეს ამგვარ სიტყვებს, და მოხდეს მათი მნიშვნელობების სრული განმარტება.

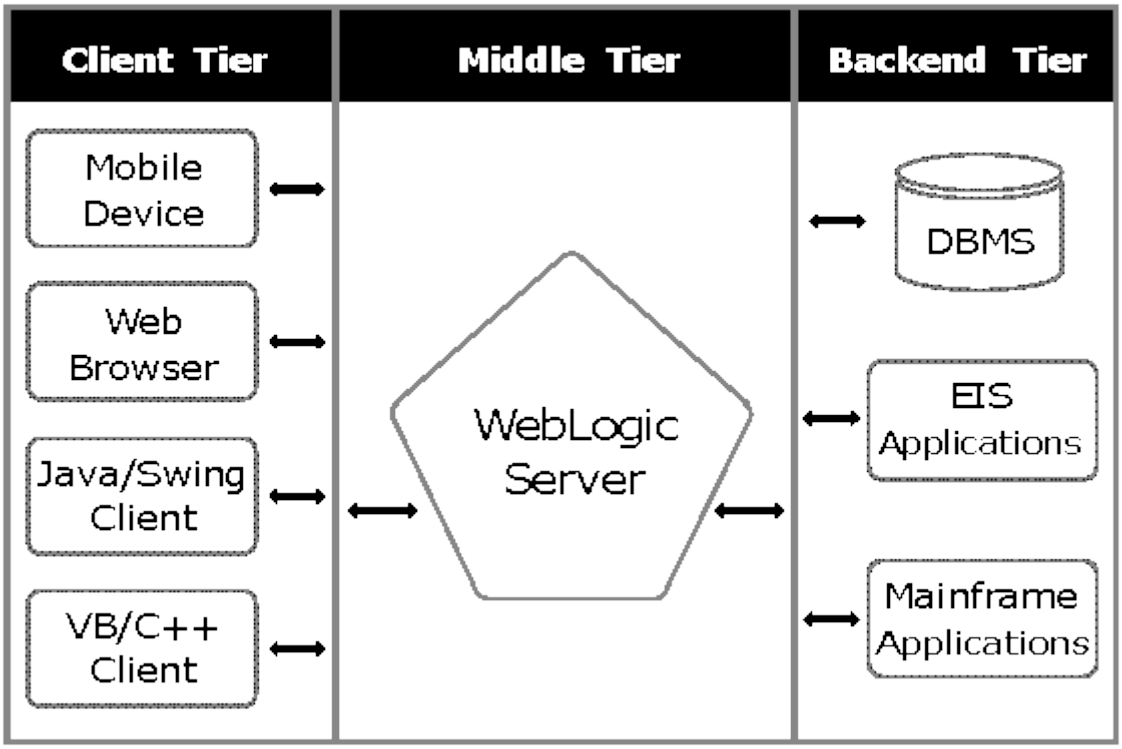
შემდგომ თავში ჩვენ განვიხილავთ მანქანური თარგმნის მთავარ სისტემებს, რომლებიც მომხმარებელს ეხმარებიან სამუშაოს ჩატარებასა თუ ენის შესწავლაში.

4. Back-End დეველოპმენტი

დინამიური ვებ აპლიკაციები შედგება რამდენიმე შრისგან: **სტრუქტურა**, **დიზაინი**, **კონტენტი** და **ფუნქციონალობა**. ტექნოლოგიას, რომლის მეშვეობითაც ვებ გვერდი ფუნქციონირებს და რომელსაც ვებ გვერდის მომხმარებელი ვერ ხედავს ეწოდება Back End. Back End შედგება 3 ძირითადი ნაწილისგან:

- სერვერი;
- მონაცემთა ბაზა;
- სერვერული ნაწილის პროგრამული უზრუნველყოფა.

ზემოაღნიშნული სამი ძირითადი ნაწილი ერთმანეთთან არიან დაკავშირებული და ერთად მუშაობენ. სხვანაირად, Back End-ს ასევე შეგვიძლია ვუწოდოთ ვებ აპლიკაციის ტვინი. Back End-ის ეკოსისტემაზე ზრუნავს მონაცემთა ბაზის მენეჯერი და სერვერული ნაწილის დეველოპერი.



Back End-ი წარმოადგენს მონაცემთა ბაზისა და სერვერული ნაწილისათვის დაწერილი პროგრამული უზრუნველყოფის კომბინაციას, რომლის გაშვება შესაძლებელია web სერვერებზე, cloud ტექნოლოგიაზე დამფუძნებულ სერვერებსა და ასევე, ამ ორის ტიპის სერვერის ჰიბრიდზე. სერვერული ნაწილის პროგრამული უზრუნველყოფა შესაძლებელია პირდაპირი ინტერაქციის გზით მონაცემთა ბაზასთან API-ს მეშვეობით, რაც აძლევს პროგრამას საშუალებას შეიტანოს ცვლილებები ბაზაში, ასე მაგალითად: ინფორმაციის წაშლა, დამატება, განახლება და ამოღება. ხოლო Back End-იდან მიღებული მონაცემები Front-End-ის ნაწილში იღებენ ვიზუალურ ფორმას, მაგალითად: მომხმარებლის შესახებ ინფორმაციის ფორმა, ონლაინ მაღაზიის პროდუქტების სია, ფინანსური მონაცემების ცხრილში ასახვა ან გრაფზე დახატვა და სხვა.

ქვემოთ მოცემული ჩამონათვალი წარმოადგენს Back End ტექნოლოგიის პასუხისმგებლის პუნქტებს:

- მონაცემთა ბაზის შექმნა, ინტეგრაცია და მენეჯმენტი—ე.წ., MySQL, SQLite, PostgreSQL, და MongoDB. SQLite შედარებით პატარა და სწრაფია, რომელიც ხშირად გამოიყენება როგორც ალტერნატივა შედარებით უფრო დიდი MySQL-თვის;
- Back-end ფრეიმვორკების გამოყენებით სერვერული ნაწილის პროგრამული უზრუნველყოფის შემნისათვის, როგორიცა Express.js;

- Cloud ტექნოლოგიებზე ინტეგრაცია - მაგალითად Amazon Web Services ან IBM Azure;
- API ინტერგრაცია;
- უსაფრთხოებს კონფიგურირება და ხაკერული შეტევების თავიდან აცილება;
- CMS სისტემების შექმნა, დეპლოიმენტი და შენარჩუნება;
- ანგარიშსწორება - ანალიტიკის და სტატისტიკის გენერირება სერვერის დატვირთვის, მომხმარებლების რაოდენობის, მომხმარებლების ლოკაციის შესახებ.

Back End დეველოპერები, სერვერული ნაწილისთვის პროგრამების დასაწერად იყენებენ პროგრამირების სხვადასხვა ენებსა და ფრეიმვორკებს. ენის და ფრეიმვორკის არჩევა შეიძლება იყოს დამოკიდებული ვებ გვერდის სპეციფიკის მოთხოვნიდან გამომდინარე. ყველაზე ხშირად სერვერული ნაწილის პროგრამების შექმნისთვის გამოიყენება შედეგი ენები და ტექნოლოგიები: Ruby, Java, C#, Python, PHP, Erlang, Node.js. ჩვენ პროექტში გამოყენებულია შემდეგი ენები და ტექნოლოგიები:

- Python - ერთ-ერთი უძველესი პროგრამის ენა, რომელიც როგორც სკრიპტულ, ასევე ობიექტზე ორიენტირებული პროგრამირების ენაა. პითონი აკეთებს აქცენტს იმაზე, რომ საკმაოდ მოკლე კოდის დაწერით, შესაძლებელი იყოს პროგრამების რეალიზაცია და გამოყენება. აღსანიშნავია, რომ პითონზე კოდის დაწერა ადვილია, ის ძალიან სწრაფია და გამოიყენება ისეთ ცნობილ პროექტებში, როგორებიცაა Youtube, Google და სხვა.
- Node.js - არის ახალი ტექნოლოგია, რომელიც დამფუძვნებულია JavaScript-ის ენაზე, რომელიც Express.js ფრეიმვორკთან ერთად გამოიყენება სერვერული ნაწილის პროგრამული უზრუნველყოფის შექმნისათვის.

4.1. NodeJS ტექნოლოგია

JavaScript - ყველაზე პოპულარი ტექნოლოგია რომელიც კლიენტის ნაწილის შექმნისთვის გამოიყენება—აქვს ფართო ქსელი front-end დეველოპმენტთან. აღსანიშნავია, რომ 2009 წელს Node.js პლათფორმის გაშვების შემდეგ JavaScript-ი გახდა ერთ-ერთი მთავარი მოთამაშე, ასევე back-end ტექნოლოგიებშიც - მოხდა მიგრაცია ასინქრონული გამოთვლებების სერვერულ ნაწილშიც, რამაც, თავის მხრივ, შეიტანა back-end დეველოპმენტში თავისი უპირატესობანი.

4.2. NodeJS ტექნოლოგიის შესახებ

Node.js is a server-side environment, რომელიც საშუალებას აძლევს Node დეველოპერებს შექმნან სერვერები და ინტერნეტ აპლიკაციები JavaScript-ის ენის გამოყენებით. ხოლოს ეს, თავის მხრივ, ნიშნავს, რომ ვებ გვერდი როგორ front-end, ასევე back-end ნაწილები შეიძლება შექმნილი იყოს JavaScript-ის გამოყენებით.

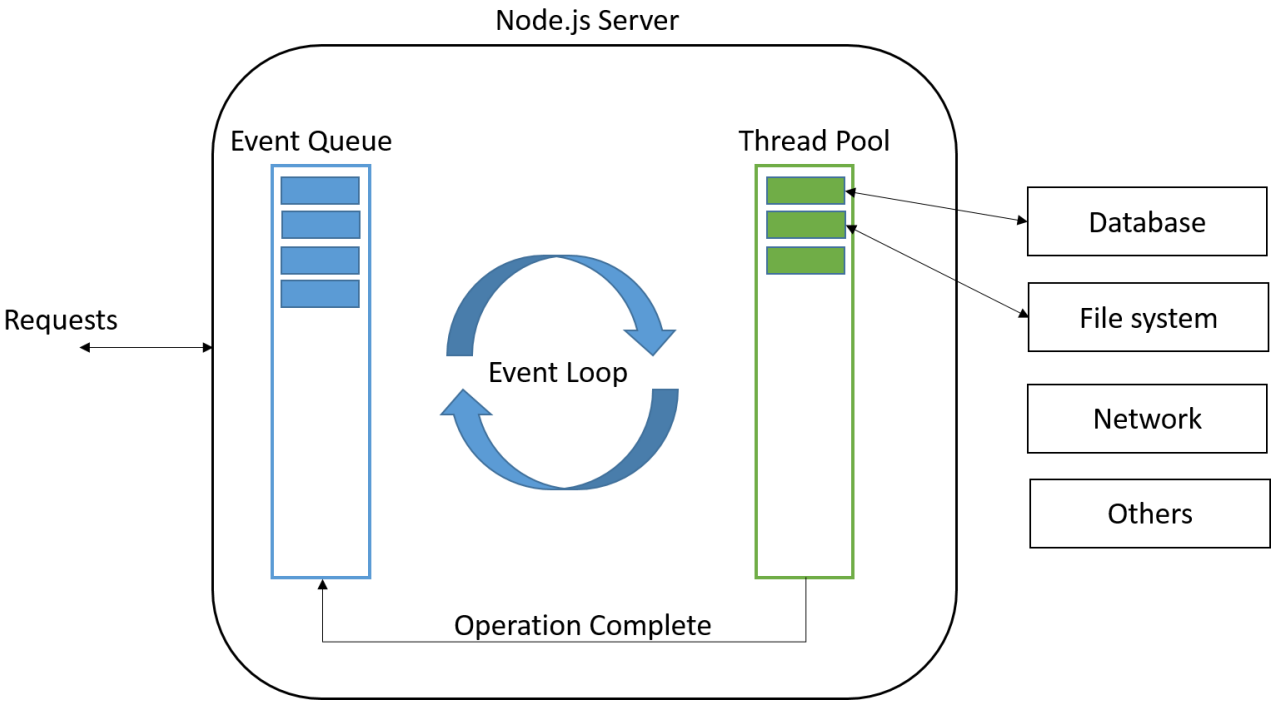
ტექნიკური თვალსაზრისით, NodeJS არის პლათფორმა და არა ფრეიმვორკი, რომელიც გამოიყენება დიდი რაოდენობის სხვა ფრეიმვორკებთან ერთად. ასევე NodeJS არის **runtime environment**, დეველოპმენტის პლათფორმის კომპონენტი, რომელიც დეველოპერებს აძლევს საშუალებას მოახდინონ პროგრამული უზრუნველყოფის ტესტირება იმ დროს, როდესაც ის გაშვებულია. Node.js უშვებს პროგრამულ კონსტრუქციას, რომელსაც ეწოდება “event loop“. იგი ელოდება მომხმარებლის მოთხოვნას და მოთხოვნის მიღებისას აგზავნის მას სერვერზე ან მონაცემთა ბაზაზე. Node.js-ზე საუბრისას აუცილებელია, რომ შენიშვნული იყოს შემდეგი მნიშნელოვანი ინფორმაცია:

- NodeJS არის კროს პლათფორმული და Open Source-ი;

- NodeJS არის ასინქრონული, **არა ბლოკირებადი, და ე.წ. event-driven I/O სისტემა(სურ.1).** სხვა სიტყვებით რომ ვთქვათ, ეს ნიშნავს, რომ აპლიკაცია არ ელოდება მიმდენარე ოპერაციის შესრულებას და გადადის სხვა ოპერაციაზე. ის აკეთებს callback-ის რიგს event loop-ში იმ მიმდევრობით, რომლითაც ეს მოთხოვნები მიღებულია, და ასრულებს ახვა პროცესებს სანამ ხდება მიმდინარე პროცესების შესრულება.

- NodeJS **შექმნილ Chrome’s JavaScript ვირტუალურ მანქანაზე, V8.** V8 არის Google-ის JavaScript

ძრავი და პასუხისმგებელი Node-ის კოდის გაშვებისათვის.



სურ. 1 **event-driven I/O სისტემა**

JavaScript პროგრამისტებს შეუძლიათ გამოიყენონ Node.js real-time web აპლიკაციებისთვის, ქსელური პროგრამებისთვის, მაგალითად სერვერებისათვს. სწრაფი სერვერული პროგრამების შექმნა ხდება event-driven მოდელში, რომელშიც ხორციელდება პარალელიზმი დედლოკის რისკის გარეშე. Node.js გახდა პოპულარი იმ ფაქტორის გამო, რომ მისი გამოყენება ადვილია და იდეალურად გამოსადეგია უმეტესობა ვებ აპლიკაციებისთვის.

4.3. NodeJS-ის გამოყენების უპირატესობები

როცა დანარჩენ ვებ ტექნოლოგიებს აქვთ თავისი საშუალებები დინამიური, ასინქრონული ვებ გვერდების შექმნისთვის, Node.js შედარებით ადვილია გამოყენებისთვის, რაც ზრდის პროგრამისტის პროდუქტიობას. ერთი შეხედვით, Node-ი მსგავსია სხვა PHP, Java და Python-ზე, შექმნილი სერვერული ნაწილის პლათფორმებისა, მაგრამ აუცილებელია აღვნიშნოთ, რომ Node-ს აქვს ერთი სერიოუზული უპირატესობა: ის ადვილად წყვეტს სისწრაფესთან დაკავშირებულ პრობლემებს. დღეს, საკმაოდ ბევრი კომპანია იყენებს Node.JS თავისი აპლიკაციების სერვერულ

ნაწილში, მაგალითად: PayPal, LinkedIn, Groupon, Walmart, Yahoo!, Intuit და Voxer. აქვე, აუცილებელია აღვნიშნოთ და ხაზი გავუსვათ იმ ძირითად უპირატესობებს, რომელიც გააჩნია Node-ს:

1. **კროს-პლათფორმულობა.** JavaScript მუშაობს თითქმის ყველა ოპერაციულ სისტემაში, რადგანაც ის თავიდანვე ბრაუზერებისთვის იყო შექმნილი. ამჟამად შესაძლებელია მისი გაშვება OS X, Windows, Linux, FreeBSD, და სხვა დანარჩენებზეც;

2. **V8 ვირტუალური მანქანა.** ძრავი, რომელიც საშუალებას გვაძლევას Node.js დაწერილ სერვერების გაშვებისა, ე. წ. V8 ვირტუალური მანქანა. ის ფანტასტიკურად სწრაფია და ასრულებს JavaScript კოდის კომპილაციას მანქანურ კოდში —ოპერაციული სისტემისთვის გასაგებ ენაზე, რომელზეც გაშვებულია. V8 ვირტუალური მანქანა C++ პროგრამირების ენაზეა დაწერილი, ეს არის ერთ-ერთი მიზეზი იმისა, თუ რატომ არის V8 ასეთი სწრაფი;

3. **JSON მუშაობს Node.js, როგორც კი მუშაობდა JavaScript.** JSON ძალიან ადვილი, სწრაფი ფორმატია ტექსტური ინფორმაციის გადანაცვლებისათვის. JSON და JavaScript ძალიან დიდი ხანია გამოიყენება ერთად, ხოლო უკვე Node.js-შიც. JSON, ასევე, იძლევა დაკავშირების საშუალებას დოკუმენტზე-ორიენტირებულ NoSQL მონაცემთა ბაზებთან , მაგალითად MongoDB (MEAN სტეკის ერთ-ერთ კომპონენტი);

4. **როგორც AJAX, Node.js-ც ფუნციონირებს მხოლოდ “callback”-ის შემთხვევაში.** ეს აბსოლუტურად განსხვავებული მიდგომა, რომელიც უკვე სერვერულ ნაწილზე გამოიყენება. პარალელური კავშირებზე საუბრისას, ზუსტად “callback”-ს ვგულისხმობთ, უფრო ზუსტად callbacks შორის კავშირს. ასეთი შემთხვევაზე დამფუძნებული მოდელი JavaScript-ისგან არის მიღებული.

4.4. სიტყვათა ქსელის აპლიკაციის სერვერული ნაწილის (Backend) აღწერა

ჩვენი ნაშრომის ამ ნაწილში წარმოდგენილია სიტყვათა ქსელის სერვერული ნაწილის არქიტექტურის, ფაილების და კოდის აღწერა. ქვემოთ ნაჩვენებია აპლიკაციის არქიტექტურა ფაილების დონეზე:

1. server.js
2. package.json
3. server
    1.1. config
        1.1.1. app.mode.js
        1.1.2. mysql.conf.js
    1.2. db
        1.1.1. mysql.js
    1.3. routes
        1.1.1. api.js

server.js - ამ ფაილში იმპორტირებულია აპლიკაციის ყველა მოდული და აღწერილია საჭირო პარამეტრები (პორტი, მისამართი და ა.შ.) სერვერის გასაშვებად. დეტალურად ამ ფაილს განვიხილავთ ნაშრომის შემდეგ ნაწილში;

package.json - ფაილი შეიცავს ყველა საჭირო მოდულების სახელწოდებებს და ასევე, სრულ ინფორმაციას აპლიკაციის შესახებ, მაგალითად პროგრამის მიმდინარე ვერსია;

app.mode.js - ფაილში მითითებულია პროგრამის გაშვების სტატუსი: “Production Mode” ან “Development Mode”;

mysql.conf.js - ფაილი შეიცავს მონაცემთა ბაზასთან კავშირისთვის საჭირო პარამეტრებს: ჰოსტი ან იგივე მისამართი, მონაცემთა ბაზის მომხმარებლის სახელი და პაროლი, და ასევე მონაცემთა ბაზის სახელწოდება.

4.5. server.js კოდის აღწერა

რადგანაც server.js წარმოადგენს სერვერული ნაწილის ყველაზე მნიშნელოვან ფაილს, რომელიც შეიცავს ყველა საჭირო მოდულსა და სერვერის გაშვებისთვის საჭირო პარამეტრებს, ჩვენ ჩავთვალეთ, რომ აუცილებელია ნაშრომში არსებული კოდის და ლოგიკის დეტალური აღწერა. ქვემოთ ნაჩვენებია კოდი JavaScript პროგრამირების ენაზე server.js ფაილიდან:

1) საჭირო მოდულების იმპორტირება:

```
const express = require('express');
const path = require('path');
const bodyParser  = require('body-parser');
const morgan  = require('morgan');
const mode = require('./server/config/app.mode');
const api = require('./server/routes/api');
const app = express();
```

2) Morgan-ის გამოყენება შემოსული მოთხოვნევის და დაგენერირებულ პასუხების ლოგირებისათვის:

```
app.use(morgan('dev'));
```

3) სტატიკური ფაილების ფოლდერის განსაზღვრა:

```
var staticFolder = mode.staticFolder;
```

4) აპლიკაციაში გამოყენებული სტატიკური ფოლდერი:

```
app.use(express.static(path.join(__dirname, staticFolder)));
```

5) api მოდულის გამოყენება ‘/api’ - მისამართისთვის:

```
app.use('/api', api);
```

6) თუ შემოცულია სხვა მარშრუტი, აპლიკაცია აბრუნებს index ფაილს:

```
app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, staticFolder+'/index.html'));
});
```

7) სერვერის მისამართის და პორტის განსაზღვრა:

```
const port = '80';
const hostname = ‘127.0.0.1’
```

8) სერვერის გაშვების ფუნქცია:

```
app.listen(port, hostname, () => {
  console.log(`API running on localhost:${port}`);
});
```

5. სიტყვათა ქსელის მონაცემთა ბაზა

მონაცემთა ბაზა არის ერთ-ერთი მნიშნელოვანი ნაწილი არა მარტო ჩვენი, არამედ ყველა მსგავსი პროექტისათვის. არსებობს მონაცემთა ბაზების ძირითადი ორი ტიპი: რელაციური და არარელაციური. ერთმანეთისგან განსხვავდებიან იმით, რომ რელაციურ მონაცემთა ბაზებში ინფორმაცია შეინახება სტრუქტურულად, ცხრილებში, როცა არარელაციურე ბაზებში მონაცამები შენახულია არასტრუქტურირებულად. რელაციურ ბაზებს შორის საკმაოდ პოპულარულია შემდეგი მონაცემების შენახვის და მართვის პროგრამები:

1. MySQL Workbench;
2. Oracle Database;
3. Microsoft SQL Server;
4. PostgreSQL.

ხოლო არარელაციური მონაცემთა ბაზების შემთხვევაში, ყველაზე ხშირად იყენებენ შემდეგ პროგრამებს:

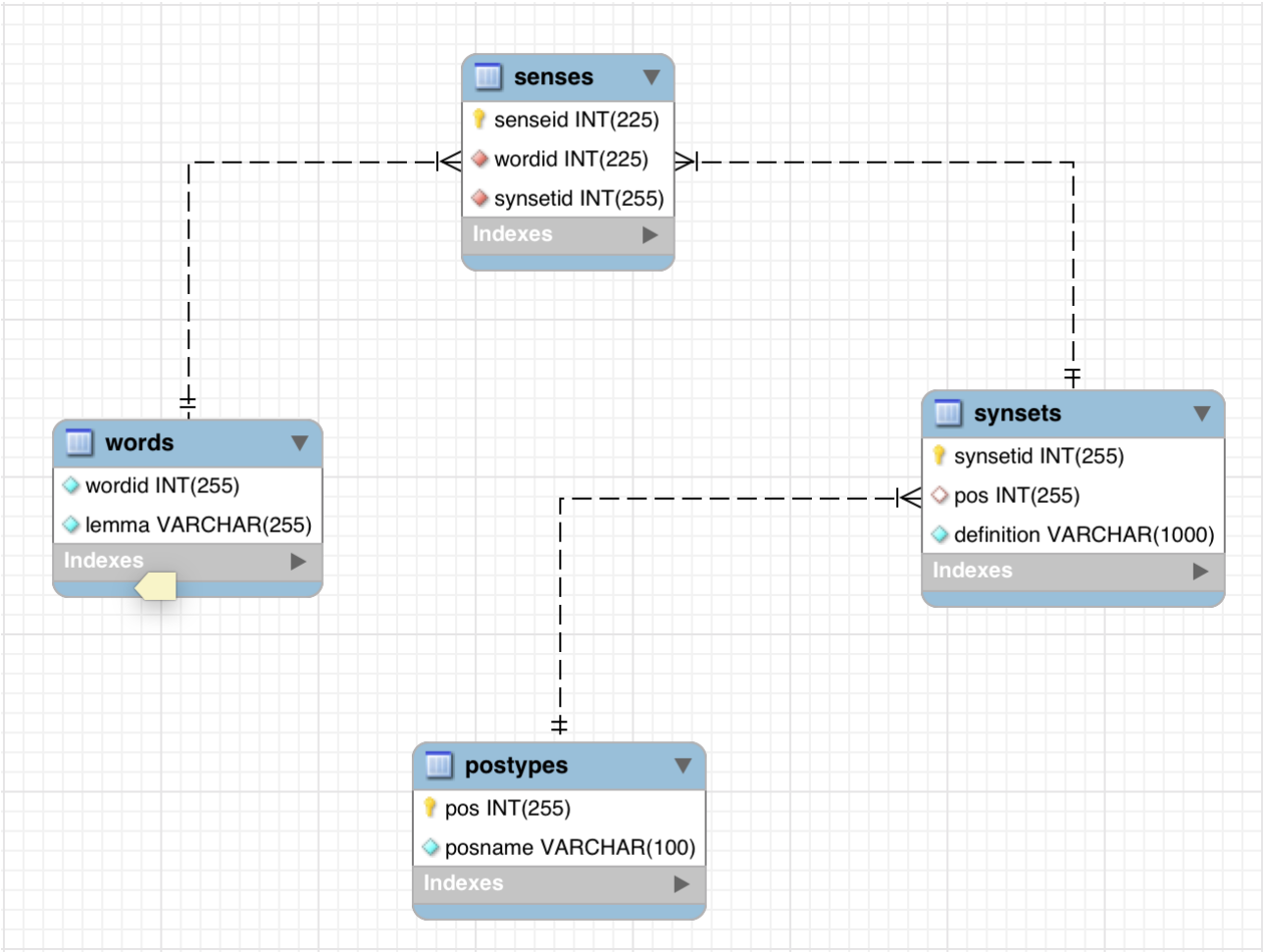
1. MongoDB;
2. CouchDB;
3. Redis;
4. DynamoDB
5. IBM Domino

ნაშრომის შემდეგ ნაწილებში ვილაპარაკებთ სიტყვათა ქსელის მონაცემთა ბაზის შესახებ.

5.1. სიტყვათა ქსელის მონაცემთა ბაზის აღწერა

ჩვენი პროექტისთვის ჩვენ ავირჩიეთ MySQL მონაცმთა მართვის სისტემა. MySQL არის რელაციური მონაცემთა ბაზის მართვის სისტემა, რომელიც ამჟამად ერთ-ერთი ყველაზე პოპულარია ბაზარზე. ის უფასოა და საკმაოდ მდიდარი ფუნქციებისგან შედგება.

ჩვენი მონაცემთა ბაზა შედგება 4 ცხრილისგან, რომლებიც დაკავშირებულნი არიან ერთმანეთთან. ქვემოთ ნაჩვენები მონაცემთა ბაზის დიაგრამა, დაგანერირებული MySQL Wokbench პროგრამაში:



დიაგრამაზე ნაჩვენებია 4 ცხრილი:

1. Senses;
2. Words;
3. Synsets;

4. Postypes.

words ცხრილში შეტანილია და დანომრილია ყველა სიტყვის სიწყისი ფორმა;

synsets ცხრილი შეიცავს სინსეტების განსაზღვრებებს, რომლებსაც მინიჭებულია უნიკალური ნომერი და ასევე POS თაგი;

postypes ცხრილში შეტანილია POS თაგების სია;

senses ცხრილი არის მოკავშირე ცხრილი words და synsets ცხრილებს შორის.

ცხრილებს შორის არსებობს სხვადასხვა ტიპის კავშირები, ესენია:

1. 1:1 - ერთი ერთთან;
2. 1:N - ერთი მრავალთან;
3. N:N - მრავალი მრავალთან.

ჩვენი მონაცემთა ბაზის შემთხვევაში, ცხრილებს შორის გვაქვს შემდეგი ტიპის კავშირები:

words - senses კავშირის ტიპია 1:N;

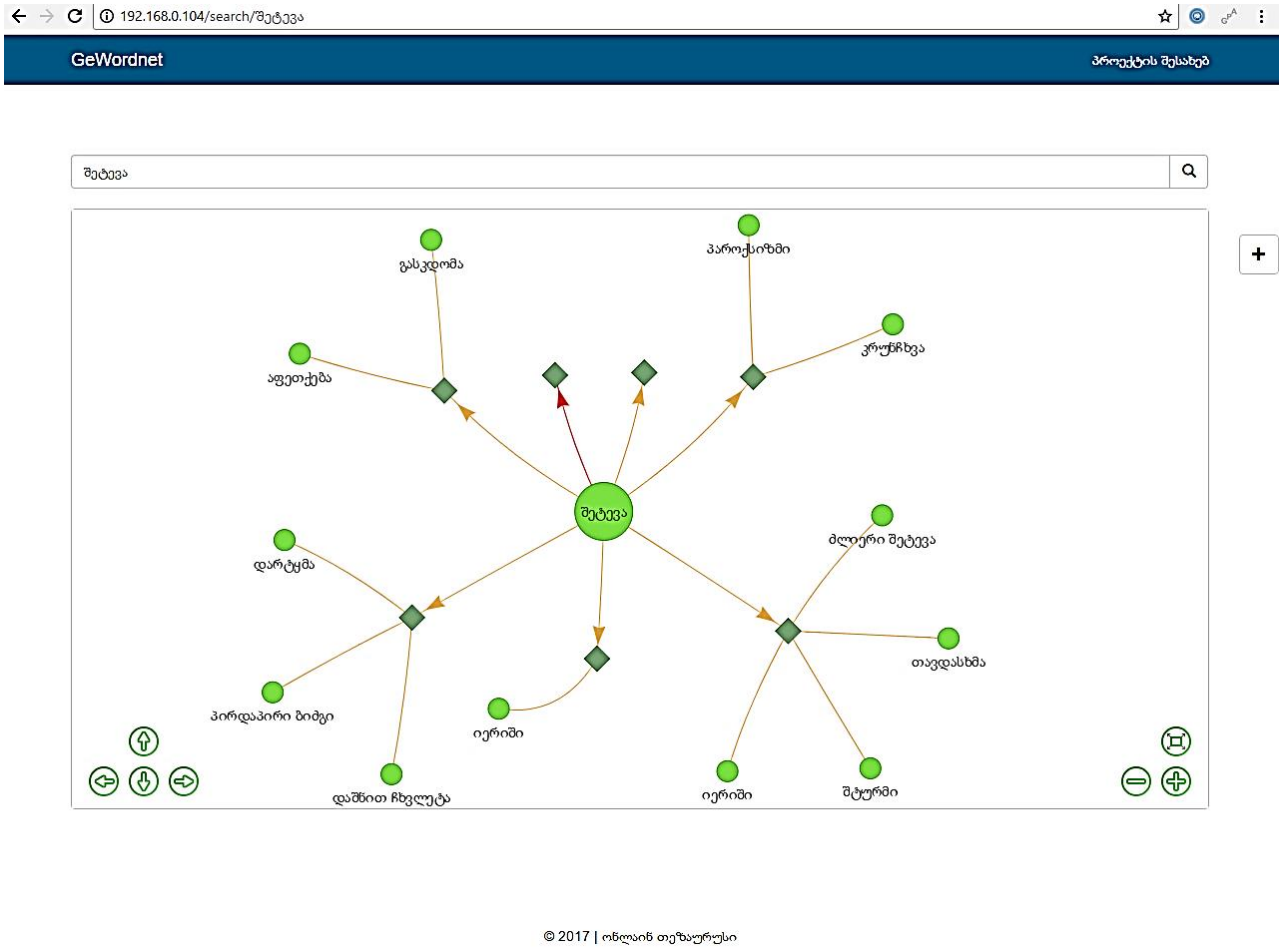
synsets - senses კავშირის ტიპია 1:N.

რადგანაც senses ცხრილი არის მოკავშირე, words და synsets ცხრილებს შორის, ამიტომაც კავშირი ამ ცხრილებს შორის იქნება N:N ან იგივე მრავალი მრავალთან.

postypes - synsets კავშირის ტიპია 1:N

6. შედეგები

სამომხმარებლო ინტერფეისი გამოიყურება შემდეგნაირად:

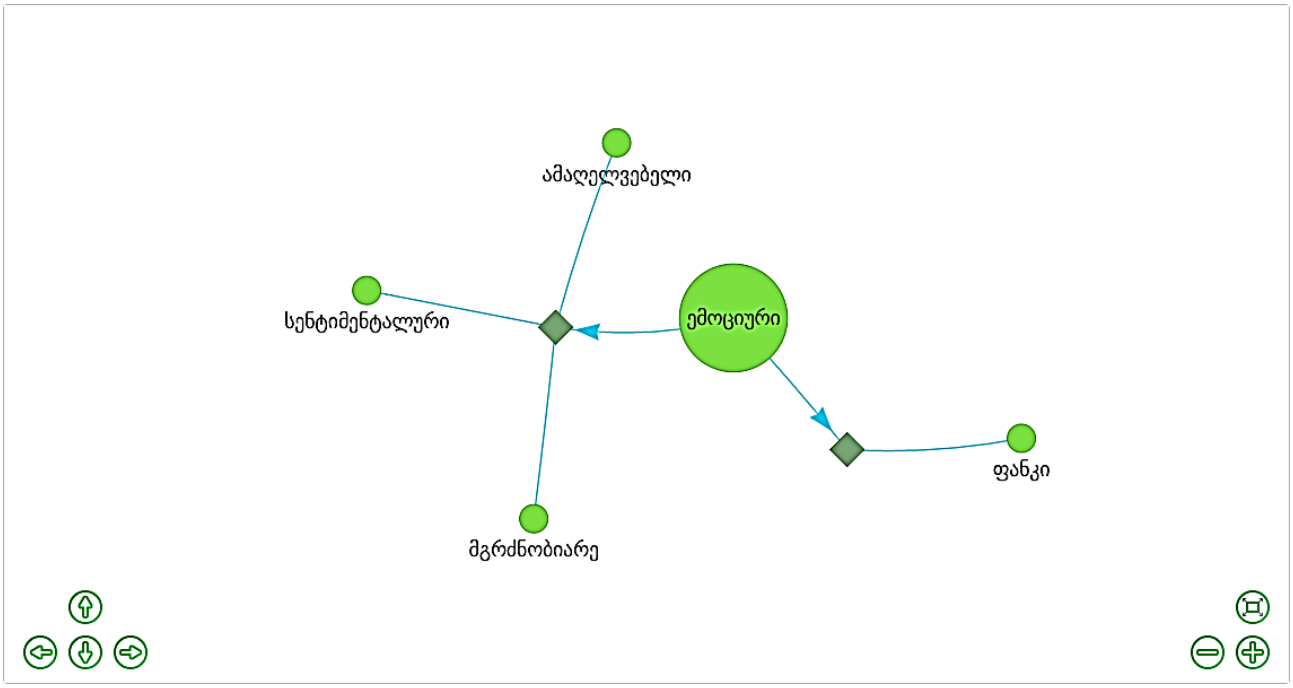


როგორც უკვე ზემოთ აღვნიშნეთ ვორდნეტის ვიზუალიზაცია მოვახდინეთ გრაფის საშუალებით. ფუნქციონალური ღილაკებით (‘+’ , ‘-‘ და ა.შ.) შეგვიძლია გრაფის გადიდება/დაპატარავება, ასევე მისი ნორმალიზაცია.

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:

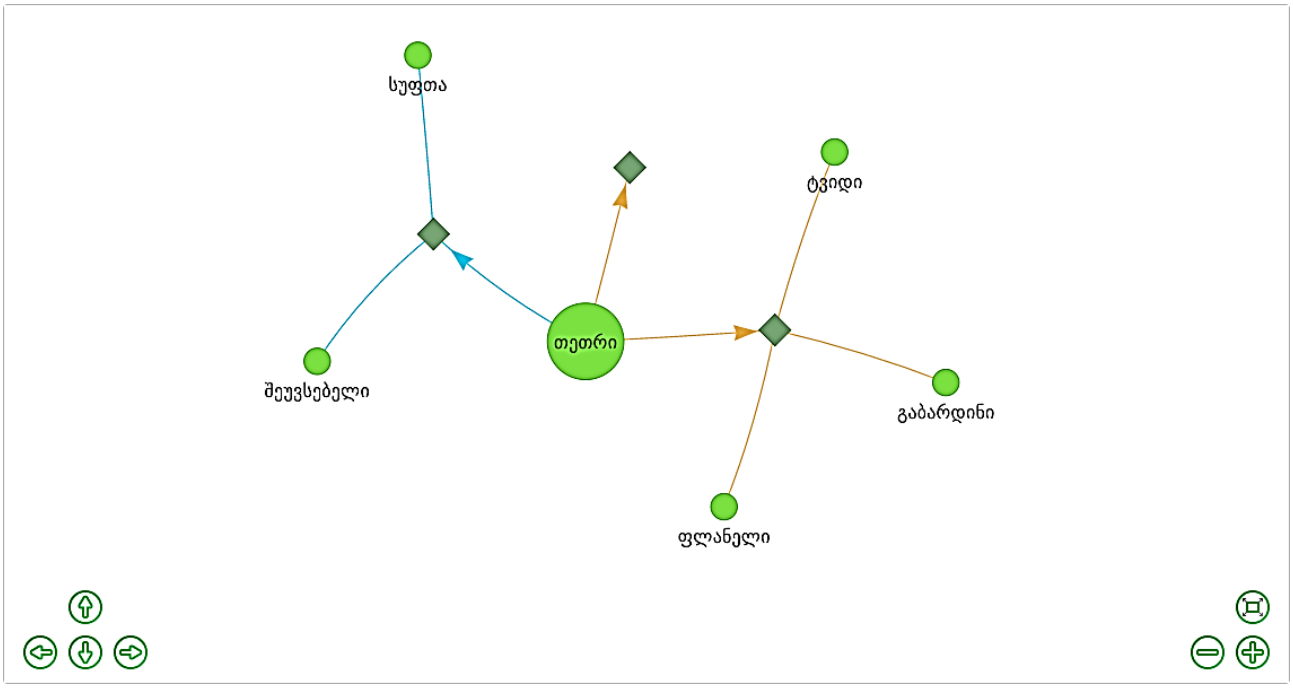
მაგალითი 2: ნაჩვენებია ძებნის შედეგი სიტყვისთვის *ემოციური*. ცისფერი წიბოები შეესაბამებიან სატელიტს. ასევე, ჩანს რომ „*ემოციური*“ დაკავშირებულია ორ სინონიმურ რიგთან, პირველი შეიცავს სიტყვებს: ამაღლვებელი, სენტიმენტალური, მგრძნობიარე, ხოლო მეორე კი შედგება ერთი სიტყვისაგან: ფანკი.



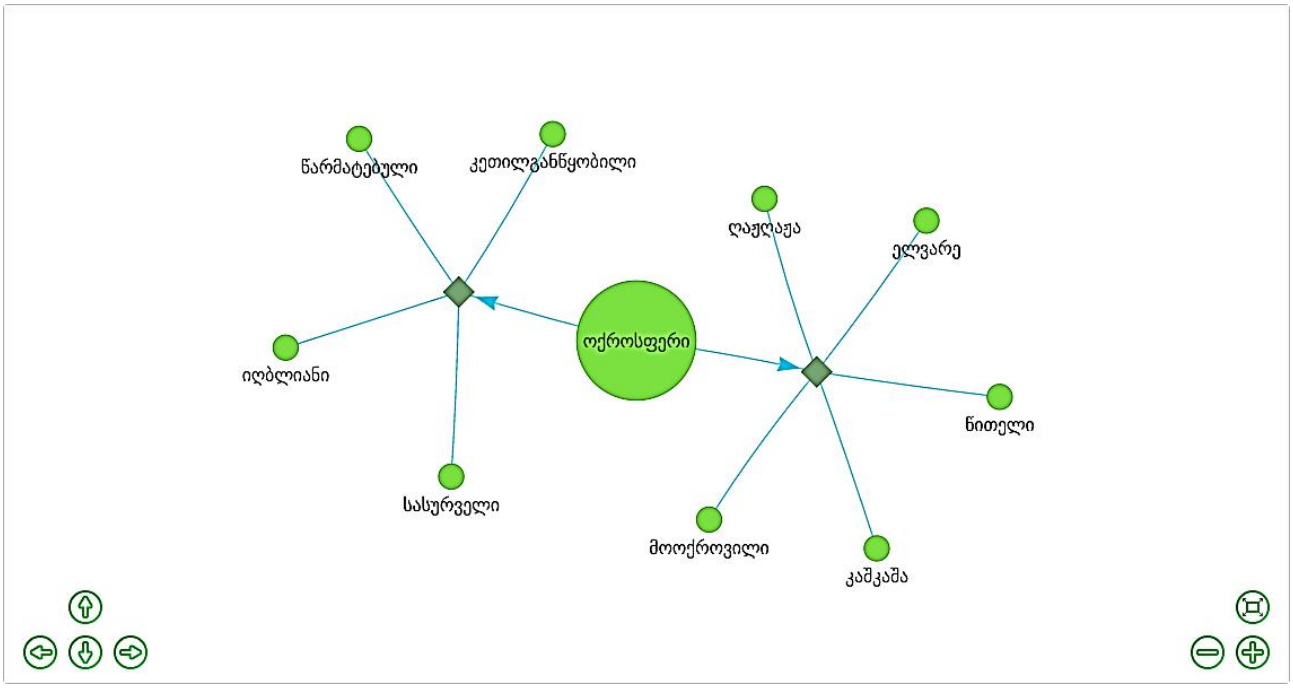
ძებნის შედეგები სხვადასხვა სიტყვებისთვის:

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

მაგალითი 3: სიტყვა „თეთრი“.



მაგალითი 4: სიტყვა „ოქროსფერი“.



7. 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 Back-end და მონაცემთა ბაზის აღწერაზე.

დეტალურად ვისაუბრეთ სერვერული ნაწილის შესახებ, რომელიც NodeJS პლათფორმაზეა აგებული. მონაცემთა ბაზის ნაწილში მოკლედ მოვიხსენიეთ არსებული მონაცემთა ბაზების ტიპები და შემდეგ ვეცადეთ გვეჩვენებინა პროექტის მონაცემთა ბაზის დიაგრამა, რომელიც ასახავდა კავშირებს ცხრილებს შორის. 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.

8. გამოყენებული ლიტერატურა

- https://en.wikipedia.org/wiki/WordNet
- https://wordnet.princeton.edu
- https://en.wikipedia.org/wiki/MySQL
- https://www.fi.muni.cz/gwc2004/proc/105.pdf
- http://www.profguide.ru/professions/front_end_developer.html
- http://hinex.ru/front-end-i-back-end-razrabotka
- http://getinstance.info/articles/tools/introduction-to-gulp/
- http://gulpjs.com/
- https://nodejs.org/en/docs/
- http://www.journaldev.com/7462/node-js-architecture-single-threaded-event-loop