CMPE 492 Senior Project II

Plan My Day

Final Report

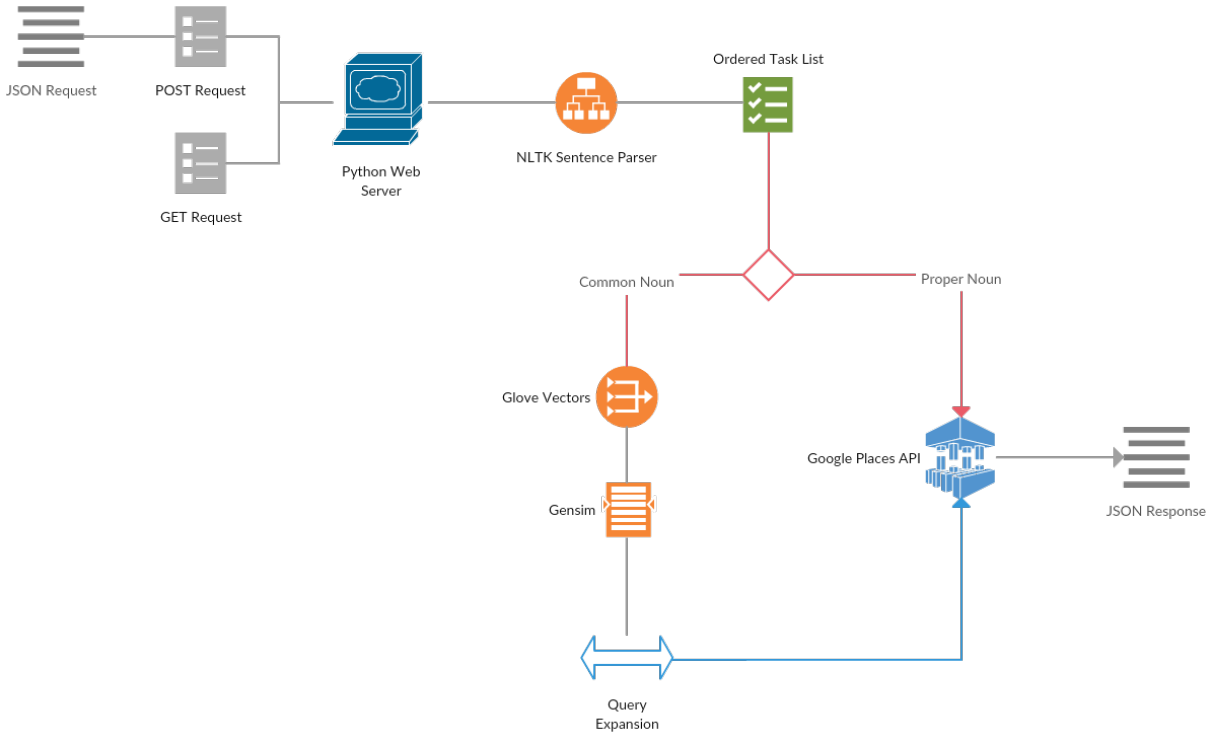
Project Group Members: Nehir Yiğit Karahan & Gökalp Fırat

Supervisor: TAYFUN KÜÇÜKYILMAZ

May 15, 2019

Contents

1. Final Architecture and Design



The figure above represents the final architecture and design of our system. The system starts with "POST" or "GET" request sent to our Python web server. If the request type is "POST" it needs to be content type of JSON and includes a JSON body to send. The purpose and the result of both types are identical the difference is "GET" requests had a character limit which may restrict applications. So the request type depends on developer usage. After the request arrived at our web server the text input will be processed at our NLTK sentence parsing algorithm. Our sentence parser returns an ordered task list depending on text's grammar. This list will be iterated on the server depending on the task's grammar; it will be either sent for a query expansion or directly sent to Google Places API. If the task needs a query expansion it will use glove vectors to find similarity from place type list which is obtained from the Places API. This query expansion leads to get best results for the task. Finally, after all requests are done the server will return the JSON response.

2. Test Results

In this part, we will be discussing the test result in detail. Questions of which tests are passed and failed, the reasons for the failure will be answered.

First of all, we want to remind our test cases.

ID	Test Case	Start Date	End Date
1	Register/Login	05.02.2019	05.02.2019
2	UI Functionality	02.02.2019	04.02.2019
3	Usage of NLTK	01.04.2019	01.04.2019
4	Voice Recognition API	15.04.2019	16.04.2019
5	Sending Request to Python Server From the Android Client and Managing the Response	25.04.2019	-
6	Google Map API With the Response(Parsed Locations)	-	25.04.2019
7	Gensim/Glove Train	01.05.2019	05.05.2019
8	Testing the scenarios all in one breath / Running the whole system	05.05.2019	-
9	Secondary Tests (Section 4)	-	-

Figure 1: Test Plan Report (Sec. 5 - Test Schedule)

TEST LEVEL	Project Team
Unit Testing	P
Integration Testing	P
Regression Testing	P
Subsystem Testing	P
Security Testing	S
Performance Testing	S
User Acceptance Testing	S

Figure 2: Test Plan Report (Sec. 4 - Test Strategy)

As another reminder, secondary tests were including security testing, performance testing, user acceptance testing. The project has not in the state of complete. Since we are still implementing some parts, security tests and user acceptance tests are not done yet.

2.1 Performance Test

For the performance testing part, we measured the execution time, which starts when the request is sent and ends when the response is ready to send, like 6-7 seconds for 1 request at a time. In our test, we used a sentence which includes total of 7 location and object:

```
"I will buy some keshar cheese or milk from migros or ikea and I will eat hamburger or tea and I will buy meat"
```

During this process, there are serious steps which we discussed in the previous sections and executions times for each is given below;

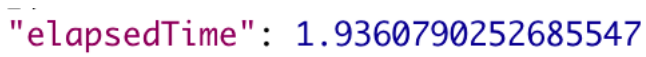
- Usage of nltk

- Usage gensim/glove part
- Preparing a json as a response

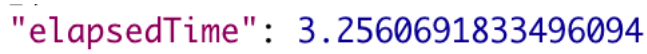
For multiple requests at a time, we may increase the performance by having a stronger server.

At the end of the JSON response of our web service we added an elapsedTime key to calculate our performance of search and query expansion processes performance. These values are calculated without usage of NLTK. The results of cases are shown in their below figures.

- First case: Searching 2 proper nouns that will do direct search. It is the fastest case because there is no need for a translate process or query expansion process. Just searching the terms.

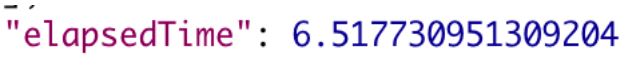


- Second case: Searching 2 common nouns. We can saw a difference near 2 sec between the first case. That can be related with the translate process and query expansion (genism & glove)



- Third case: Searching 2 proper nouns but now with or clauses. In our service if a task separated with or, the closest place between them will be resulted in the response. This process includes calculation between 2 latitude and 2 longitude values with using Haversine approach. The results are much longer as we expected

- Fourth case: Searching 2 common nouns with an or statement. Now the process not only includes distance calculation an all of the common noun processes included too such as query expansion and resulted the worst performance scenario.



2.2 Primary Tests

Primary tests includes register/login, UI functionality, usage of nltk, voice recognition API, google map API for marking the locations, gensin/glove.

- Register/Login
  - ➢ We tried to log in with any google account(Gmail) and the test is passed.
- Usage of NLTK
  - ➢ We used lots of input test case sentences in our python server and observe the process.
  - ➢ We checked if every sentences are parsed into expected meaningful parts and it is working well. The test is passed.
- Voice Recognition API
  - ➢ We tried Google's API on an Android device by just speaking and we observed that it sometimes fails and we cannot take the risk so for us, it failed.
  - ➢ To fix this, we decided to let our users edit the text after they speak.
- Google Map API

- We used this API on our server-side and we used longitude and latitude data set for a certain location as an input for the API and it found and marked that place. The test is passed.

- Gensim/Glove Train
  - We used this for query expansion method(Sec. NO), and we gave “hamburger” food as an input and we observed the expected outcome which is “hamburger food” in return. So, this test is passed.
- Testing the whole running system
  - Since the implementation is not completed yet, we delayed this part. So the test result is unknown yet.

2.2.1 Google Place API Tests

- The service’s main search engine for place lookup is Google’s Places service. The company provides an API to let developers made search on their place database. Because of Google is the biggest search engine in the world, most of the places that either new or not are registering their business to Google’s database. We made tests with our parsed words and proper nouns are giving true results nearly 100%. Such as searching an shopping mall or a school name.


```
{
   "html_attributions" : [],
   "results" : [
      {
         "geometry" : {
            "location" : {
               "lat" : 39.88656590000001,
               "lng" : 32.8554594
            },
            "viewport" : {
               "northeast" : {
                  "lat" : 39.88791572989273,
                  "lng" : 32.85680922989273
               },
               "southwest" : {
                  "lat" : 39.88521607010729,
                  "lng" : 32.85410957010728
               }
            }
         },
         "icon" : "https://maps.gstatic.com/mapfiles/place_api/icons/geocode-71.png",
         "id" : "73cbd8327ddc8df7d36e73f33f34b126f0ef86a5",
         "name" : "Atakule",
         "place_id" : "ChIJkc6nToBP0xQR9bhx3q1SF0I",
         "reference" : "ChIJkc6nToBP0xQR9bhx3q1SF0I",
         "scope" : "GOOGLE",
         "types" : [ "route" ],
         "vicinity" : "Çankaya"
      }
   ],
   "status" : "OK"
}
```


- You can see in the above figure that “Atakule” keyword found as one result and it’s the true one. There are some decisions we need to take before sending requests to place API.
- First one is **nearbysearch,** by using this parameter we get near locations within a radius or ranked by distance. This decision must be given from developer too. We chose ranked by distance parameter because our service is a planning service and it need to achieve nearest locations possible.

- Second one is **output**. There are 2 types of outputs API can result. These are json and xml. Most of the applications use JSON and its better because XML is a bit deprecated nowadays and have some performance concerning. JSON gives result cleaner to read and better to process.
- Third parameter is **key.** Every developer must provide their Google API key to use Google services. For huge amount of usages, Google will process some transactions from your credit card.
- Forth parameter is **location.** We need to have and provide client's current location as longitude and latitude values. This is to get nearest locations
- Last but not least parameter is **keyword.** This is where we send place name to search. A term to be matched against all content that Google has indexed for this place, including but not limited to name, type, and address, as well as customer reviews and other third-party content. We used type in our query expansions which will be explained in a detailed way later on. Also, there is an opennow parameter which only results currently open places by comparing their work times with timezone. This parameter wasn't so effectively for us because most of the places or not providing their work times in the Google database.

  https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=39.881596,32.855591&rankby=distance&keyword=Atakule&key=AIzaSyCDsQTZEJZAjEcjm3fP9s5Rz6ANXRB4G2o

Sample request URL for the result above with parameters

2.2.2 Query Expansion Tests

In the 2.2.1 test we checked results for proper nouns which are not challenging so much. We had one more case which is common nouns. First our test will be done similarly as for proper nouns. We tried direct search by putting the word in the keyword parameter and there are a lot of false results retuned to us. We tried keyword of cheese as Turkish because we are located at Turkey. We saw that searching only the cheese term will not give us any supermarket as it need to do. However sometimes some words without query expansion can give good results such as hamburger or pizza. These words are mostly used at place titles so their results will be better. But for the cheese case we can absolutely see the needing of an engineering solution and this solution is **query expansion**

```
{
   "html_attributions" : [],
   "results" : [
      {
         "geometry" : {
            "location" : {
               "lat" : 39.8735292,
               "lng" : 32.8620509
            },
            "viewport" : {
               "northeast" : {
                  "lat" : 39.87486062989272,
                  "lng" : 32.86345157989273
               },
               "southwest" : {
                  "lat" : 39.87216097010728,
                  "lng" : 32.86075192010728
               }
            }
         },
         "icon" : "https://maps.gstatic.com/mapfiles/place_api/icons/shopping-71.png",
         "id" : "a89635405186a0927f643212ab90d9dc8a30c5da",
         "name" : "Ege Yörem Zeytin Peynir Evi",
         "opening_hours" : {
            "open_now" : false
         },
         "place_id" : "ChIJ3Qa8fXpF0xQRjZOJzlKyteo",
         "plus_code" : {
            "compound_code" : "VVF6+CR Ankara, Turkey",
            "global_code" : "8GFJVVF6+CR"
         },
         "rating" : 0,
         "reference" : "ChIJ3Qa8fXpF0xQRjZOJzlKyteo",
         "scope" : "GOOGLE",
         "types" : [ "store", "point_of_interest", "food", "establishment" ],
         "user_ratings_total" : 0,
         "vicinity" : "Yıldızevler Mahallesi, Turan Güneş Blv. No:36, Çankaya"
      },
      {
         "geometry" : {
            "location" : {
               "lat" : 39.890023,
               "lng" : 32.846698
            },
            "viewport" : {
               "northeast" : {
                  "lat" : 39.89136087989272,
```


- The main challenge for query expansion is obtaining those additional tokens and phrases. We used Glove & Gensim tools to obtain most similar place type and merge the place type and word resulted as “cheese supermarket” We can criticize language tests under this test category too. Our service’s text language to process is English because the natural language processing libraries are developed for English. The problematic issue is searching these English words at different countries than UK or USA. However, we managed to solve this problem by using a translator tool which will be described later on. Back to our case the keyword will iterate over place types and most similar one which will be “supermarket” and made a query expansion to create our new keyword “cheese supermarket” which will result a better result can be seen below.

```
{
   "html_attributions" : [],
   "next_page_token" : "CrQCMAEAALPb3MkcDs63_0L6xmZnvJwo7FqXPALIgdpaZdyL7oAz9nGK-
7IaDjjlcHNioo0bHllSSuckStiVq_JRzw7APaM4G9_UC_aLjMqM6LragTPfnOprv6d3u2JsugtzXrYLTmdNPUhhz
2Sb7ODYCxVIAa4fthXUBRtkxtSVJMn-EwrvDbv8SQmv7ImFzHnDvkMBFt9NPGz46n78CDuqYwEelSyA5waYLc3Qr
Pt7wVNCgQFRvHJfvw8SEJTCXw_NSwxnKyTSWRqA_s4aFJdVKQiwCmlfC-i7N91wTlRLM6kE",
   "results" : [
      {
         "geometry" : {
            "location" : {
               "lat" : 39.8837068,
               "lng" : 32.8546949
            },
            "viewport" : {
               "northeast" : {
                  "lat" : 39.88499952989272,
                  "lng" : 32.85606322989272
               },
               "southwest" : {
                  "lat" : 39.88229987010727,
                  "lng" : 32.85336357010728
               }
            }
         },
         "icon" : "https://maps.gstatic.com/mapfiles/place_api/icons/shopping-71.png",
         "id" : "2aa2a734c78498d9cc821f5367fbe2e1a47b51a0",
         "name" : "Niyazi Şarküteri",
         "opening_hours" : {
            "open_now" : false
         },
         "photos" : [
            {
               "height" : 4160,
               "html_attributions" : [
                  "\u003ca href=\"https://maps.google.com/maps/contrib/10169529167468612
               ],
               "photo_reference" : "CmRaAAAAJCNTWUBQuer1Iga9eHc8I4RgilxH7EXZ5EGIdE5HA3JC
yyu4uaEhANBTxDJEEzbyEUSonRiQzxGhT1TQwwBw09fX0Ng8l9U1MGfxbAvA",
               "width" : 3120
            }
         ],
         "place_id" : "ChIJf90j639F0xQRrEb216OqqHw",
         "plus_code" : {
            "compound_code" : "VVM3+FV Çankaya, Ankara, Turkey",
            "global_code" : "8GFJVVM3+FV"
         },
         "rating" : 4.7,
```


2.2.3 Distance Calculation Tests

- In previous test case the process of or statement is told already. It will split the task with or delimiter and do searches based on this split, calculate distances and give the true result. We need to be sure that it will split the words correctly, calculate distance correctly and choose the shortest one.
- Our test values will be Bilkent, Atakule and market. The text is given as “I will visit Bilkent or Atakule or market”. This text must give the shortest one between them. First let’s check all the places one by one.

*** The distances are related to our test client’s location

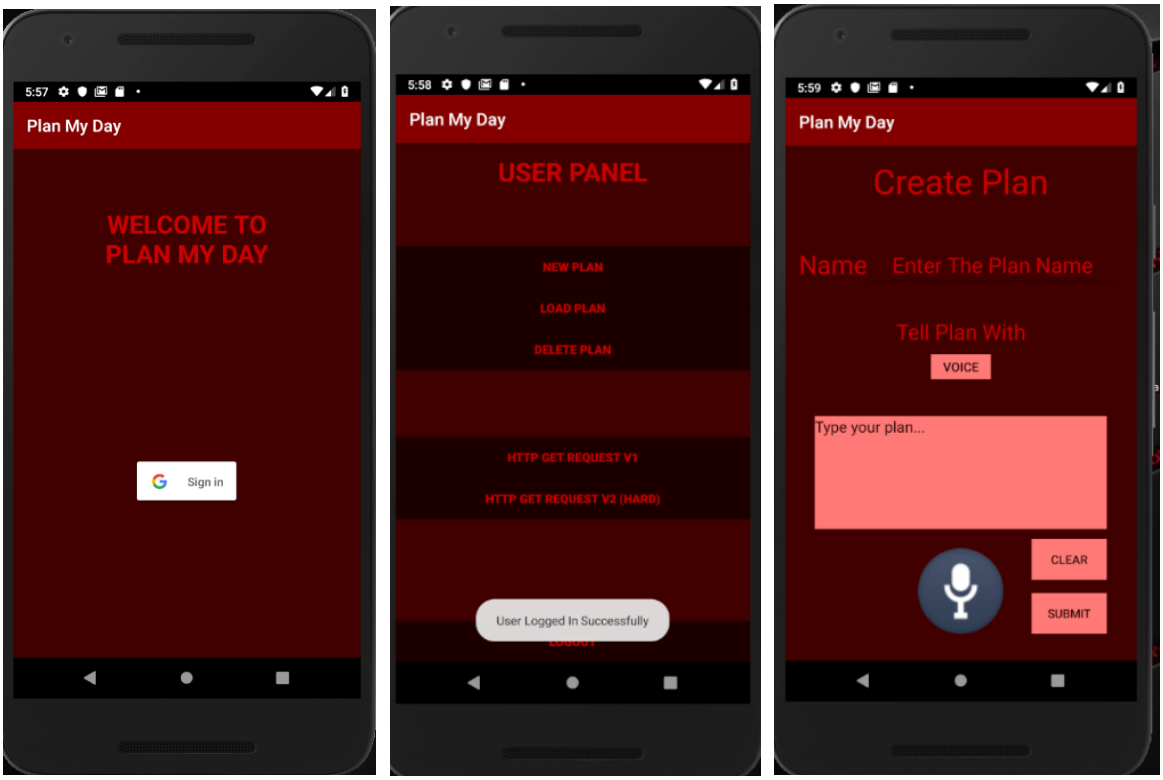
```
    "name": "Bilkent Üniversitesi Doğu Kampüs Turizm ve Otelcilik (THM)",
    "location": {
        "lat": 39.8746195,
        "lng": 32.7617833
    },
    "query": "Bilkent",
    "distance": "10.57 km"

    "name": "Atakule",
    "location": {
        "lat": 39.88656590000001,
        "lng": 32.8554594
    },
    "query": "Atakule",
    "distance": "4.42 km"

  "name": "Kismet Food Market",
  "location": {
      "lat": 39.925177,
      "lng": 32.864902
  },
  "query": "market interest",
  "distance": "0.12 km"
```


- These are the results of these keywords one by one. So that, the result of that or statement task should be give “Kısmet Food Market” which is the nearest. The result is given below, and the test is passed.


```
    "name": "Kismet Food Market",
    "location": {
        "lat": 39.925177,
        "lng": 32.864902
    },
    "query": "Bilkent or Atakule or market",
    "selected": "market interest",
    "distance": "0.12 km"
```


2.2.4 Android Test

- Since we need to do tests on devices. We need to create an application an made runs at it to prove that our service is application friendly that can support many devices. For this purpose, we create our Android application’s visual interface and some server request tests. However, it is not completed yet. So, we can’t provide our test results in this report yet, we can only show our progress for the visual interface of our Android tests.



2.3 Potential Enhancements For The Future

- We can buy a stronger server in order to support more request at a time and reduce the bottleneck to improve the overall performance.
- With time, we can add new futures accordingly to the user feedback.
- We can run tests to find the security vulnerabilities.
- We can work on Android application's UI part.
- After completed the tasks above, we can put our application and API to the market.

3. Engineering Solutions

Our main process on the server can be mentioned as;

Nltk parses the sentence into meaningful parts and understands and detects the locations and objects(anything the user wants to have) and ordering them accordingly to the user's input sentence's structure and returns a list, then the gensim/glove part gets the part of this data set as an input and makes query expansion for that part and finds the relative places for the objects and picks those locations' coordinates and send those to the Google Places API. And the other part which the gensim didnt take will be sent to the Google places API directly.

After those steps, the server prepares a Json which includes these coordinates and their relative data (name of the place, name of the object, distance between the user) and returns this Json by the response to the client.

Our service's architectural design is REST. The concept of RESTful services is the notion of resources. The clients send request to these URIs using the methods defined by the HTTP protocol. We have been using GET and POST methods in our application. The characteristics of a REST system are defined by six design rules:

1) Client-Server: Separation between server that offers the service and client that consumes it.
2) Stateless: Every request from a client must contain all the requirements of the server to carry out request.
3) Cacheable: The server must indicate to the client if requests can be cached or not.
4) Layered System: Communication between a client and a server should be standardized in such a way that allows intermediaries to respond to requests instead of the end server, without the client having to do anything different.
5) Uniform Interface: The method of communication between a client and a server must be uniform.
6) Code on demand: Servers can provide executable code or scripts for clients to execute in their context. This constraint is the only one that is optional.

The impact of our engineering solution in global context can be analyzed as usability. Any devices with a network connection can send HTTP requests; therefore, our service can be used by a lot of devices. To talk about societal context that the service can be used by blind people with a combination of voice recognition systems. That is mostly depending on voice recognition systems because this technology is still at the beginning of its development. The blind people can use it to plan their day by just telling the text to our server and the rest of it will be handled by server. There may be some benefits for environments too such as decreased usage of maps or lists for plans that will decrease usage of paper proportionally.

3.1 NLTK

Firstly, NLTK is a leading platform for building Python programs to work with human language data. In our project, we are getting the whole plan of the user in a text format and trying to understand users' purpose in order to prepare the best and efficient path for them.

First of all, we need to understand how nltk works. It uses two default methods below for tokenizing and tagging the elements of the sentence.

```
tagged = nltk.pos_tag(nltk.word_tokenize(text))
```

In this example, the text is the input. Using nltk library it tags the words in the sentence. For example, if our text is:

```
"I will buy milk and bread"
```

Then the tagged version of this sentence is:

```
[       ('I', 'PRP'), ('will', 'MD'), ('buy', 'VB'), ('milk', 'NN'),
        ('and', 'CC'), ('bread', 'NN')                                ]
```

Alphabetical list of part-of-speech tags used in the Penn Treebank Project:

Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

This is default behavior of nltk and these tags have meanings, for example, NN means that "milk" is a noun. VB means that "buy" is a verb. There is a list of these tags and their meanings provided in the nltk's website.

Our aim was to understand the user's plan by using this library. For our project, we do not need anything but common nouns, proper nouns or direct company/place names. Assuming that our text is:

"I will buy hamburger"

For this sentence, we do not consider these questions;

- Who will buy
- Who will do what
- When do s/he do this

Only question is, "what will be bougth". So, taking the noun will be enough for this case. But what if our text is:

"I will buy cheddar cheese"

For this case, just getting nouns simply will not support the case. Because, both "cheddar" and "cheese" will be tagged as nouns and they will be treated as different objects. So, instead of getting "cheddar cheese" we would pick "cheddar" and "cheese" seperatedly which causes the wrong results after all.

The solution for this is using chunking. Chunking is a process of extracting phrases from the text instead of just simple tokens which may not represent the actual meaning of the text. So, we wrote a grammar which is regular expression and give this to the chunkparser method, then we were able to combine and pick the "cheddar cheese" together. In our grammar, we are specifaying the NP (noun phrase) label as any number of arithmetic nouns.

Considering another example, the input text is:

"I will eat hamburger at burgerking or mcdonalds"

The case that we should support here is the choosing burgerking or mcdonalds by the distance, we should pick the closest one for the user. And we should understand that the "hamburger" is not important for this case because user already spesifaied the places where s/he wants to go. For this we decided to search the sentence and tried to find the word which has the "IN" tag(at, in, on, from words have this tag). First of all we are checking if there is such a word in the sentence and if there is, we are getting the rest of the sentence by starting from the next word. So, we will have:

"burgerking or mcdonalds"

And another issue that should be resolved here is the deciding that both words will be chosen or not. If there was an "and" instead of "or" we should've choose the "burgerking" and "mcdonalds" as splitted but otherwise we should choose the closest one. Sınce the returned list of nltk process will be examined in the next subsytem, we should be able to controll this in that part. So, the solution method that we created was, updating our regular expression to create a label for this case. Now, we are labeling the words if they meet the requirement of this rule;

NN followed by OR followed by NN

Since the "or" and "and" words are tagged with the same tag which is "CC" there is no way to distinguish if those nouns are seperated by "and" or "or" with this version of grammar. So, we added another label rule to our regular expression in order to group the words meet this rule;

```
OR:{(<NP>?<CC><NP>?)*}
```

So, now we can split the sentence by the delimiter of " and ". Then, we check if there is still "OR" label in our sentence and if there is, we can be sure that the conjunction is "OR" since we already eliminated the "and"s.

Then we are adding this "OR" label all together in order to keep that "or" word because in the next subsystem we will search the returned elements of the list for "or" word and if there is, we will choose the closest one by the distance.

There is a priority on the labels in our code flow. If there is a word with the tag of "IN" in the sentence, we are taking the rest of the sentence and splitting it by the " and " word and checking if there is a verb in the parsed sentences, if there is, we are giving those sentences to our method recursively. If there is not we are directly adding it to the list. For example:

- "I will eat meat at burgerking and I will eat meat"

There is a word with the tag of "IN" in this sentence which is "at". So we are taking the rest of the sentence as:

- "burgerking and I will eat meat""

Then splitting this part accordingly to the " and "

- "burgerking"
- "I will eat meat"

Checking is there a verb in these two sentences, if there is not add it to the list and if there is send it back to the method. So, "burgerking" will be added directly and the "I will eat meat" part will be sent to the method. And it will add the meat to the list after all. So, the "IN" tag has the highest priority. It also provides us to eliminate unnecessarry words, for example;

"I will eat hamburger and drink cola at burgerking or mcdonalds"

In this case, we won't even look at the "hamburger" and "cola" since there is a word with the tag of "IN". We will find the distance between user-burgerking and user-mcdonalds and pick the closest one.

Special name groups (NNP) has the second priority and the common noun groups (NP) has the third and the lowest priority. We actually defined this priority rule before we implement and work on the "IN" label, so it is not necessarry for now but it was providing better results before we updated our grammer to support "IN" label cases, so we decided to keep this priority as it is. If the sentence is:

"I will eat hamburger at Burgerking or Mcdonalds"

It helped us on just taking the special names(Burgerking and Mcdonalds" and we could eliminate "hamburger" (NN) again but users had to write them in capital and we couldn't check the conjection without using "OR" label method. So, this was a temporary solution. However, this priority rule may still function for the cases/sentences which includes common nouns and special names together and does not include "OR" label.

Assuming that the input sentence is:

```
"I will buy some keshar cheese or milk from migros or ikea and I will eat
hamburger or tea and I will buy meat"
```

As we mentioned, we put some priorities on the tags. In this case, if there is a word with the tag of "IN", which includes these words: "at", "in", "from", "on", it means thats a special location for the

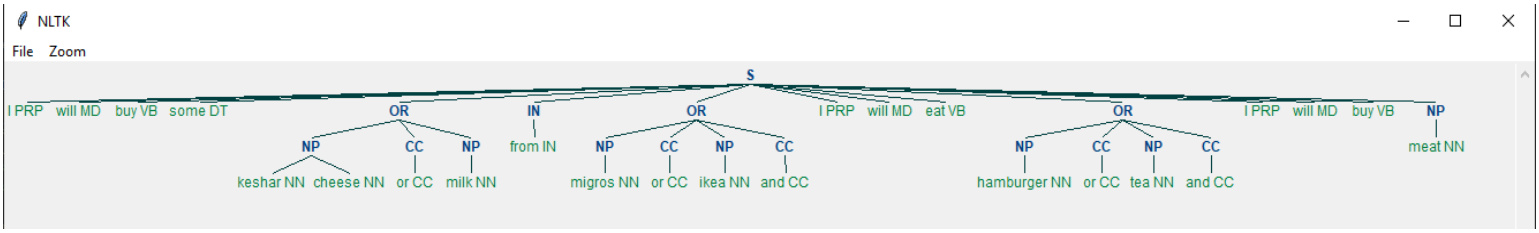
user. In the example above, its migros or ikea, so user will buy some stuff but we do not need to pick those because we know where the user actually wants to go. And getting migros and ikea was another challenging part because figuring that there is “from” in the sentence does not make picking “migros” or “ikea” easy by any means.

To have those two keywords(migros and ikea), we used the chunking method which nltk provides. With this method, we could write a regular expression to combine the words together accordingly to their tags. For example;

```
grammar = """NNP:{<NNP><NN>*<NNP>*}
NP:{<NN>*}
IN:{<IN>}
"""
chunkparser = nltk.RegexpParser(grammar)
mytree = chunkparser.parse(tagged)
```

“mytree” is a tree and it has several subtrees accordingly to the chunks which we described in the grammar. So, NNP, NP and IN are the labels are created by the rules. By checking the existence of these labels in a sentence, we can have some serious ideas about the user’s input.

This is how this tree looks like accordingly to the different grammar which has the rule for the “OR” label too.



So for this case, since the “IN” label has the highest priority we are focusing on the migros or ikea part by skipping the keshar cheese and milk. Then we are getting into the “OR” label and finding “NP” labels by observing their conjunctions. Because if there is “and” between nouns, which means the user will go to the migros and ikea both, but if there is or between them, we need to pick the closest one for the user.

The same logic is applied to the “hamburger or tea” part too. After all, the list of tasks in order is below;

['migros or ikea', 'hamburger or tea', 'meat']

If there is “or” word, in the “OR” label, we are adding it between the “NP” labels (migros or ikea), but if there is “and” word, we are separating our “NP” labels and putting them into the array as different elements. So, if we write “and” instead of “or” then the returned list is;

['migros', 'ikea', 'hamburger', 'tea', 'meat']

As we treat the sentences differently for some cases, we are changing our chunk parser accordingly to these cases.

These cases can be listed as below:

- If there is a word with the tag of “IN”
- If there is special name which are tagged with the tag of “NNP”
- If there is common nouns which are taged with the tag of “NP”

- If there is a label with the name of “OR”
- If any two of them are together in the sentence

These topics will be discussed in the presentation with all the details.

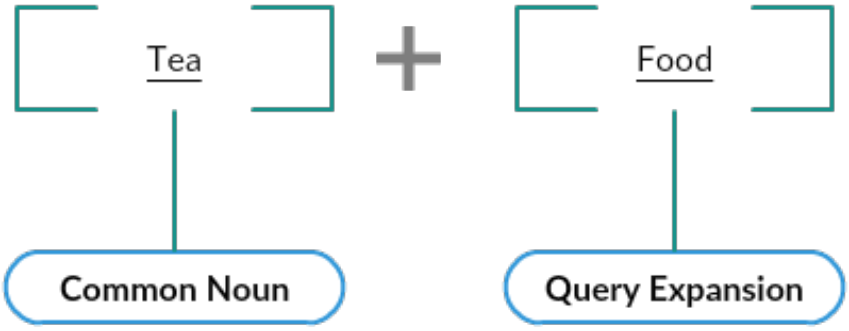
3.2 Query Expansion

In our tests we told why we need query expansion. In query expansion, users give additional input on query words or phrases, possibly suggesting additional query terms. Some search engines (especially on the web) suggest related queries in response to a query; the users then opt to use one of these alternative query suggestions. Query expansion broadens the query by introducing additional tokens or phrases. The search engine automatically rewrites the query to include them. For example, the query vp marketing becomes (vp OR “vice president”) marketing.

Query expansion is one of the most important engineering solutions that we provide in our service. Because as you can see that searching directly the common nouns will not always give the true results that we want. However, with the use of query expansion we are expanding the searching keyword with the place type. Google allows to search a keyword with place types but there is a supported list for it. Some of them are:

accounting	jewelry_store	car_repair	parking
airport	laundry	car_wash	pet_store
amusement_park	lawyer	casino	pharmacy
aquarium	library	cemetery	physiotherapist
art_gallery	liquor_store	church	plumber
atm	local_government_office	city_hall	police
bakery	locksmith	clothing_store	post_office
bank	lodging	convenience_store	real_estate_agency
bar	meal_delivery	courthouse	restaurant
beauty_salon	meal_takeaway	dentist	roofing_contractor
bicycle_store	mosque	department_store	rv_park
book_store	movie_rental	doctor	school
bowling_alley	movie_theater	electrician	shoe_store
bus_station	moving_company	electronics_store	shopping_mall
cafe	museum	embassy	spa
campground	night_club	fire_station	stadium
car_dealer	painter	florist	storage
car_rental	park	funeral_home	store
		furniture_store	subway_station

We can expand our keyword with these place types but how do we know which one to use? The answer is by using Glove. Glove's detailed explanation is given at tools part of our report. Basically, it is vector representation for words. There are some pre-trained vector files such as Twitter 2B tweets at the Glove's main page. The vectors are trained by their usage frequency at Glove so twitter may not give us the best results. Therefore, we have used 2014 Wikipedia's Dump data which includes 6B tokens and 400k vocabulary inside. It is small and best case for our problem. Finally, we plug this vector file to genism to calculate the similarities between two words. At the end, we made an the functionality of genism to compare similarity of common noun with place types and find most similar one. The most similar one is our place type to use at query expansion.



4. Issues with the topic

4.1 NLTK

If the input sentence is:

"I will Visit Ted university or Bilkent university and I will eat meat at burgerking and I will eat meat"

Then the result list has "burgerking, meat" in it. It won't pick the two universities where the user wants to visit. The reason is the priority, if there is a word with the tag of "IN", we are taking the rest of the sentence:

"at burgerking and I will eat meat"

Then parsing it for the "and" word, and sending these parsed parts to the method recursively.

"at burgerking"

"I will eat meat"

This is how it works better for many cases. Because otherwise, we could not support the better case, for example;

"I will buy milk, cheese, water, and french fries from migros"

For this case, if we split the sentence accordingly to the "and" before checking if it has a word with the tag of "IN";

"I will buy milk, cheese, water"

"french fries from migros"

Then the list would be;

['milk', 'cheese', 'migros']

And we could needlessly pick the milk and cheese when just picking the migros was enough.

So, this is not actually a problem or bug. We know the reason for this problem and we know the solution so we are considering this as a trade-off.

Additionally, if the user enters this sentence as;

- "I will Visit Ted university or Bilkent university THEN/AFTER/BEFORE I will eat meat at burgerking and I will eat meat"
- "I will Visit Ted university or Bilkent University. I will eat meat at burgerking and I will eat meat"

There will be no problem.

4.2 Gensim & Glove

- There are some issues we face with while matching similarities with Gensim & Glove.
- These problems mostly caused by Glove because Gensim uses algorithmic calculations.

- The problem is depending on we're using a pre-trained word vector form Wikipedia. Some words may give false similarities such as hamburger must match with food or restaurant in similarity search.
- But this similarity search for now gives supermarket and the query expansion becomes "hamburger supermarket"
- That will affect place results too. When searching for hamburger we can get restaurants to eat hamburger but with this result we will get results to buy a hamburger meat.

- We can solve this issue by using a predefined hash table or a database which includes key value inputs for some words such as matching hamburger with a food.
- Therefore, before checking similarity the service will look up through the database and if it's not found then it will do similarity search.
- This will minimize the false results but it will increase cost because there will be need of engineers to fill the database.

5. New Tools and Technologies

5.1 Glove & Gensim

There are two learning algorithm for word vector representation these are word2vec and Glove. In word2vec, Skipgram models try to capture co-occurrence one window at a time. In Glove it tries to capture the counts of overall statistics how often it appears. Glove basically counts how frequently a word appears in a context. Depending on our needs Glove is the best choice. We need to apply query expansion for common nouns. Query expansion is basically adding another text at the end of the text. Google Place API allows us to do that for given keyword. We can expand our query with the place types allowed by Google. This process is giving us the best results for common noun place search. But first we need to get most similar place type to the noun. For this purpose doing "Cosines Distance Formula" will give us the similarity we need. We used Gensim library to use this algorithms easily. Gensim is designed to process raw, unstructured digital texts. The algorithms in Gensim, such as Word2Vec, FastText, Latent Semantic Analysis (LSI, LSA, see LsiModel), Latent Dirichlet Allocation (LDA, see LdaModel) etc, automatically discover the semantic structure of documents by examining statistical co-occurrence patterns within a corpus of training documents.

5.2 Flask

Flask is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website. We had used this framework for our API's homepage and request/responses.

5.3 Googletrans

Googletrans is a free and unlimited python library that implemented Google Translate API. This uses the Google Translate Ajax API to make calls to such methods as detect and translate. We need this library because even our service text input should be in English the client's location may be not in UK or USA. Therefore common nouns should be translated to the client's language before making requests to Google Place API.

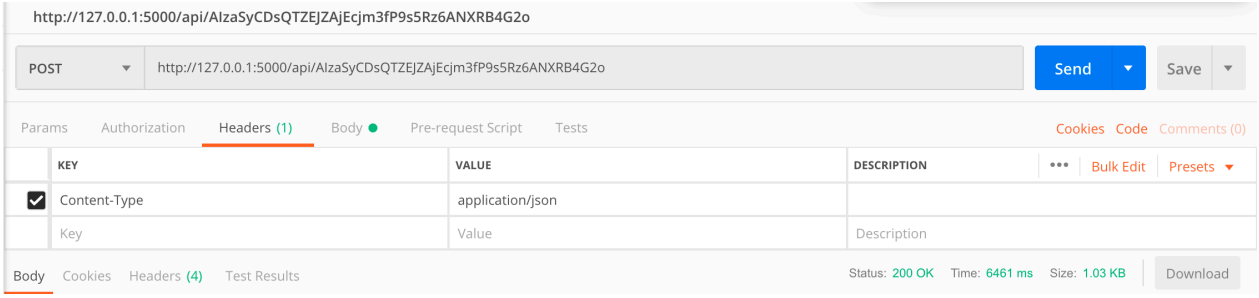
Googletrans library needs source and destination languages to work and these languages should be provided as countries short names such as Turkey TR. To find client's country short name we used another Google service of maps. We have sent client's latitude and longitude to geocode API of maps and get JSON result that gives short code of the country that can be seen at the below.

```
{
   "plus_code" : {
      "compound_code" : "M8X9+8X Sofia, Bulgaria",
      "global_code" : "8GJ5M8X9+8X"
   },
   "results" : [
      {
         "address_components" : [
            {
               "long_name" : "8",
               "short_name" : "8",
               "types" : [ "street_number" ]
            },
            {
               "long_name" : "Lavele Street",
               "short_name" : "Lavele St",
               "types" : [ "route" ]
            },
            {
               "long_name" : "Sofia Center",
               "short_name" : "Sofia Center",
               "types" : [ "neighborhood", "political" ]
            },
            {
               "long_name" : "Sofia",
               "short_name" : "Sofia",
               "types" : [ "locality", "political" ]
            },
            {
               "long_name" : "Sofia City Province",
               "short_name" : "Sofia City Province",
               "types" : [ "administrative_area_level_1", "political" ]
            },
            {
               "long_name" : "Bulgaria",
               "short_name" : "BG",
               "types" : [ "country", "political" ]
            },
            {
               "long_name" : "1000",
               "short_name" : "1000",
               "types" : [ "postal_code" ]
            }
         ],
         "formatted_address" : "Lavele St 8, 1000 Sofia Center, Sofia, Bulgaria",
```


5.4 Postman

Postman is a powerful HTTP client for testing web services. Postman makes it easy to test, develop and document APIs by allowing users to quickly put together both simple and complex HTTP requests.

This application fits best to test our **GET** and **POST** requests by defining parameters or request body. We made our API's **Unit** Tests and **Integration** Tests to test its functionality will work. The client provides time, size & status information which are really important performance values for a web service. These values allow us to examine our service's current performance and where to made optimizations.



6. References

- https://radimrehurek.com/gensim/intro.html
- https://www.quora.com/How-is-GloVe-different-from-word2vec
- https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask
- https://nlp.stanford.edu/IR-book/html/htmledition/query-expansion-1.html
- https://queryunderstanding.com/query-expansion-2d68d47cf9c8