**GEOCODING API BASED ON ADDRESS PARSING**

**by**

**Mustafa Arslan**

**Engineering Project Report**

**Yeditepe University**

**Faculty of Engineering and Architecture**

**Department of Computer Engineering**

**2015**

**GEOCODING API BASED ON ADDRESS PARSING**

APPROVED BY:

Assoc. Prof. Dr. Sezer Gören Uğurdağ ……………………  
(Supervisor)

……………………

……………………

DATE OF APPROVAL: .../.../...

**ABSTRACT**

Geocoding is an important field in military, commercial and educational areas and currently used to locate the geolocation information of postal addresses or reverse geocode a geolocation to find the postal address. A Geocoding System gets a postal address input, processes it, and tries to find the exact geolocation of the given address by comparing the address items with the database records. Geolocation Systems generally use top-down approach in order to minimize the search area and increase the performance. They clean the address and convert address to a known format, split address to tokens in order to identify it, and then compare these tokens with the database items. An exact geolocation is found if possible. Most of the Geocoding systems are finding the location in street or avenue level, so a Geocoding system is developed in order to find the exact location of the postal address. This project aims to find the location from door numbers. .Net Framework and Oracle Database is used to build this Geocoding system.

**ÖZET**

Adresi kordinat bilgilerine çevirme işlemi eğitim ve ticaret sektörü için çok önemli bir alandır ve şu an posta adresi bilgilerini coğrafik koordinat bilgisine çevirmek için kullanılmaktadır. Coğrafik koordinat bulma sistemleri bir posta adresini girdi olarak alır, bu adres bilgisini işler ve adresin içindeki bilgileri referans veritabanındaki bilgilerle karşılaştırarak ilgili adresin coğrafik konumunu en küçük hata oranıyla tespit etmeye çalışır. Genel olarak bu sistemler mümkün olan en büyük parçadan başlar ve en detay parçaya kadar inmeye çalışır. Böylelikle performansı arttırır ve arama alanını kısıtlar. Öncelikle girdi olarak verilen posta adresinde istenmeyen öğeleri temizler, adresi belirli bir formata sokar. Daha sonra adresi parçalarına ayırır ve bu parçaları tanımaya çalışır. Son olarak da veritabanında bulunan referans öğelerle bu parçalar karşılaştırılarak coğrafik koordinat tespit edilmeye çalışılır. Coğrafik koordinat bulma sistemleri genelde konumu sokak yada cadde boyutunda tespit ederler. Bundan dolayı posta adresinden coğrafik koordinatı tam olarak tespit eden bu sistem geliştirilmiştir. Bu proje coğrafik koordinatı kapı seviyesinde aramaktadır. Bu sistem .Net Framework ve Oracle Veritabanı kullanılarak yapılmıştır.

**ACKNOWLEDGEMENT**

I would like to express my gratitude to Assoc. Prof. Dr. Sezer Gören Uğurdağ for his guidance, suggestions, insight and encouragement towards this project. To my family, I offer sincere gratitude for their support throughout my undergraduate studies and for this project. I also thank Yusuf Can Semerci for his useful discussions and great friendship.

TABLE OF CONTENTS

ABSTRACT ………………………………………………………………………………….iii

ÖZET …………………………………………………………………………………………iv

ACKNOWLEDGEMENT …………………………………………………………………….v

LIST OF FIGURES ……………………………………………………………………….....vii

LIST OF TABLES .………………………………………………………………………....viii

[1. INTRODUCTION 1](#_Toc439511328)

[2. BACKGROUND 3](#_Toc439511329)

[2.1. Problems of Postal Address Geocoding 4](#_Toc439511330)

[2.2. Technologies 5](#_Toc439511331)

[3. DESIGN 7](#_Toc439511332)

[4. IMPLEMENTATION 16](#_Toc439511333)

[4.1. Fixer 16](#_Toc439511334)

[4.2. Tokenizer 18](#_Toc439511335)

[4.3. Geocoder 20](#_Toc439511336)

[4.4. Database 24](#_Toc439511337)

[5. EVALUATION 29](#_Toc439511338)

[6. CONCLUSION 31](#_Toc439511339)

[REFERENCES 32](#_Toc439511340)

LIST OF FIGURES

Figure 3.1 Flow of GeocoderAPI ………………………………….......……..….….…………...8

Figure 3.2 Flow of Fixer Process……………………………………….......….….….….……....9

Figure 3.3 Flow of Tokenizer Process …….………………......……….......….….……...….....10

Figure 3.4 Flow of Geocoder Process ……………………………………….......…........……..11

Figure 3.5 The use case diagram.….…………………………………….......….…......….….....12

Figure 3.6 The class diagram ..................................................................................................... 13

Figure 3.7 The state diagram ..................................................................................................... 14

Figure 3.8 The sequence diagram .............................................................................................. 15

Figure 4.1 Fixer clears the spaces ……………………..………………………….........…...… 16

Figure 4.2 Fixer clears the spaces and converts address to uppercase …...……..…......…........ 17

Figure 4.3 Fixer restructures the address …………………………………...…...…….......….. 17

Figure 4.4 Fixer clears the parenthesis ...................................................................................... 18

Figure 4.5 List of the keywords …..………………………………………….….......….....…. 18

Figure 4.6 Tokenizer splits the address into pieces ……………...….……….……..….......… 19

Figure 4.7 AddressLevel object ................................................................................................. 20

Figure 4.8 Geocoder search method for city …....……..…………………..…..…….…........... 21

Figure 4.9 Geocoder search method for town …………………..…….…........…..…..........… 22

Figure 4.10 Procedure collects quarter, road and town ............................................................. 23

Figure 4.11 Door numbering in Turkey ..................................................................................... 24

Figure 4.12 ER Diagram of the reference database ................................................................... 25

Figure 4.13 Connection between Oracle Database and Entity Framework ............................... 27

Figure 4.14 IL object created by Entity Framework .................................................................. 27

Figure 4.15 LINQ query for town search ................................................................................... 28

LIST OF TABLES

Table 5.1 Geocoding Detection Rate ……………………………….......…….….…………….29

Table 5.2 True Geocoding Detection Rate …………………………….......….…..…………...30

Table 5.3 Geocoding Search Performance …………………………….....….…......……..…...30

# INTRODUCTION

In human history, mapmaking has been a crucial part for eight thousand years. From cave paintings to modern maps, people have used maps in order to help them to navigate their way, define and explain the environment that they have been lived. After that, they gave names to every element to identify them.

In modern age, when postal systems introduced, it became necessary to give names to every single element. Because of that, postal address system is found. Postal addresses are textual representation of a location on the Earth.

With the inventions of the computer systems, people wanted to explain these maps in different ways because of the military, logistic, agricultural, educational and some other reasons because postal addresses are not sufficient enough to express every single location on Earth. This leads to invention of geolocation systems.

Geolocation systems represent a point on Earth’s surface with numeric representation. It has been used for many areas such as military, transportation, health analysis, crime analysis etc. This leads the need of creating a relation between postal addresses and geolocation. Geocoding systems are developed because of this reason.

This project aimed to achieve a Geocoding System that can be used in many areas such as commercial or educational purposes and finds the exact geolocation of the given postal address if possible. Most of the geocoding systems accuracy level is limited to streets or avenues, because their aim is not to find an exact location.

In order to find the exact location, these systems need a database that has all the address data. The data contains country data, city data, town data, avenue data, street data, building names data, door numbers data, and their relations between them. This project tries to find the door number and its coordinates to find the exact location.

Geocoding systems are accurate if the input postal address contains the true form of the address. The input has to have full names of all the address parts. There shouldn’t be any misspelled word or an incomplete address part. However, it is really hard to get the address in this form. People usually make mistakes while writing their addresses. This decreases the success of the geocoding systems. Also town names, avenues names, streets names or buildings names can be changed. Incomplete addresses are also another problem and the most common one.

Incomplete addresses are the most common problems that geocoding systems encounter. In order to fix this problem, this system tries to cross the related input fields and also the related data fields. After that, it searches the input in these data fields.

In this report, at Chapter 2 the background of Geocoding systems, problems of postal addresses and technologies that used to build this system is described. Chapter 3 has the design part of this system and gives information about the Geocoding processes. Implementation part is given in Chapter 4. This part has the detail explanations of the processes and the database. At Chapter 5, test results are given. Finally, Chapter 6, future work is presented.

# BACKGROUND

Geocoding is a concept of determination of geographic representation of a location which can be used to find the location on the map. It finds the representation of a location by a set of letters, numbers or symbols which is called coordinates. Coordinates have three parts; latitude, longitude and elevation.

Latitude represents the north-south position of a point on the surface of the Earth and Longitude shows the east-west position of the point. Elevation shows the points height above or below of a reference point. Latitude and longitude are commonly represented by numeric characters.

There are many ways for searching the geolocation of a point. GPS systems are popular for searching geolocation because every smartphone has a GPS device inside it. Some systems use the modems which is a device that enable computer to transmit data over, search for location. Using postal addresses is another method of geolocation search.

Postal address is a textual representation of a location. Postal address consist of several sub parts, these are country, city, town, village, quarter, avenue, street, building number, door number or places of interests (POI). City is a large place that humans constructed and settled. Town is smaller version of city. Village is small human settlement, it is smaller than town. Quarter is a human settlement, too but it is smaller than village. Avenue and street is a road. Building and door number are the numbers to identify the buildings and doors. Places of Interests are the places that well known by everyone that lives in that area. Shopping malls, government buildings, schools, universities are examples of POI’s.

In some cases, people need to find the postal address by using the geolocation data that they have. This process is called reverse geocoding. Reverse geocoding is a critical process because location based services such as emergency services use it to find the readable address.

There are some well-known geocoding and reverse geocoding services such as Google geocoding service [1], Yandex geocoding service [2] and Bing Geocoding Service [3].

Google geocoding service uses Google Maps [4] data to geocode a postal address or reverse geocode a geolocation data. Google collects and updates the address data by using the satellites and scanning streets with dedicated teams. Google geocoding service also offers users to create routes from one location to another.

Yandex geocoding service uses Yandex Maps [5] data to geocode a postal address or reverse geocode a geolocation data. Yandex geocoding services are commonly used in navigation systems because Yandex determines the traffic jams from the users that uses Yandex navigation systems and offers alternative routes. This service is free-to use.

Bing geocoding services use Bing Maps [6] data. Microsoft [7] uses satellites and field teams to collect these data. Bing geocoding services also provide what Google and Yandex geocoding systems provide, too.

## Problems of Postal Address Geocoding

Searching geolocation from a postal address has some problems. One of the problems is, in daily life, address parts such as street, avenue, building number etc. can be changed by the government because of reorganization of the cities or towns. In this scenario, geocoding system databases have to be updated; however old data has to be stored as a legacy address because humans may not adopt the new address structure instantly.

Second problem is the defects that postal addresses have such as misspelled words. Geocoding systems may not identify these words and this decreases the success rate of the geocoding systems.

Incomplete addresses are the other problem of the geocoding systems. Address parts are tightly bounded to each other. An incomplete address breaks this relation between the address parts and decreases the success rate of the geocoding systems, too.

Humans have a habit to write special characters and notes in the addresses. These are unnecessary items because they don’t have any information about the location or geocoding systems can’t identify them.

## Technologies

Geocoding systems need a storage unit to store the location information. Generally a relational database management system (RDMS) is chosen for this purpose such as Oracle Database [8], Microsoft SQL Server [9], MySQL [10], IBM DB2 [11] etc. RDMS is system software that stores data and allows programmers to work on the data.

Oracle Database is a RDMS developed by the Oracle Company. It requires a server which is a computer that has high abilities rather than normal computers. Oracle Database uses this server to store and retrieve data.

In order to process the data provided from the database, geocoding systems uses programing languages like C#, Java, C++ etc. A programing language is a special language that programmers use to develop applications or give instructions for computers to execute.

C# is developed by the Microsoft. It is a high level and object oriented programing language that allows programmers to develop robust and secure applications. C# can be used as a programing language to create client applications, web applications or web service applications etc. Visual studio is generally used for developing C# applications as an Integrated Development Environment (IDE). C# works with .NET Framework.

IDE is basically a text editor that allows programmers to create and then run these applications. IDE’s are generally consist of three main parts; a debugger, text editor and build automation tools. Visual Studio, NetBeans and Eclipse are the commonly used IDE’s. Visual Studio is developed by Microsoft.

.NET Framework is a technology that supports building and running applications. It consists of class libraries that C# programing language needs to use such as general objects and general methods that programming community agrees on it.

Object Relational Mapping (ORM) is a programing technique that converts database objects to programing language objects and let programmers to query these object without writing SQL queries. ORM also guaranties the type safety of the written code. Entity Framework and Hibernate are the well-known ORM tools that programmers use. Entity Framework is an open source ORM tool that developed by the Microsoft.

Language Integrated Query (LINQ) library is a sub part of .NET Framework. It can be used for querying the databases or in memory objects. LINQ is a library that prevents type mismatch problems and allows developers to write queries in .NET languages like C# or VB.NET. LINQ can also be used by querying any type of datasets like XMLs, databases or in memory objects like arrays, lists or the key-value storages.

Web Services or background services are software systems and their main focus is information exchange via the internet, local network or application layers. They use predefined structures that accepted by the programming community.

# DESIGN

In this project, a system is developed that finds the exact location of the postal addresses. The system uses top-down approach to minimize the error rate, increase the performance, and determines the location.

The postal address will be gotten as an input from the user to search for a location but the system is limited because of the sample database. The user has to search addresses that exist in Turkey.

In order to identify the location found by the process, it has decided to categorize the levels of the addresses as follows; city, town, quarter, avenue, street, POI and door.

Most geocoding algorithms work like a navigation system. Because of that they are searching addresses for the first six levels, so except the POI information the locations that these algorithms found are not accurate enough. They return the coordinates and information for the center of the address levels that provided above.

Aim of this project is to determine the exact location of the postal address which is the textual representation of a location if possible. In order to achieve that, system is divided into three main processes: Fixer, Tokenizer and Geocoder. These processes are shown in Figure 3.1.

An input postal address is given to GeocoderAPI at first. This input postal address is passed to Fixer process. Fixer process restructures the postal address and eliminates the special characters and returns a restructured address string. Restructured address string is given to Tokenizer process. Tokenizer splits address into tokens and tries to identify these tokens by using reference keywords. It returns a container and this container is given to Geocoding process. Geocoding process uses items inside the container and search them in the address database. If a geolocation is found, Geocoder process returns it as an output.

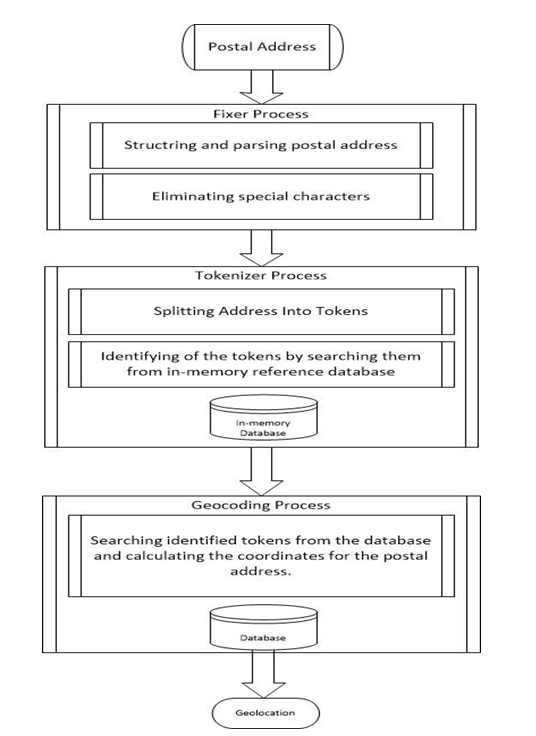
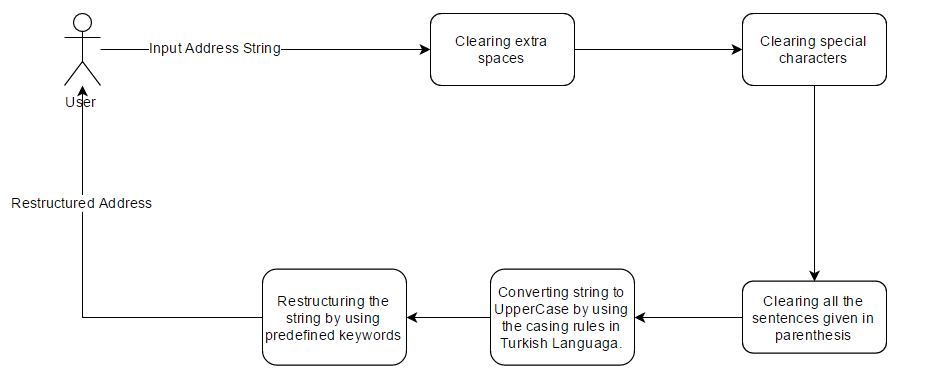


Figure 3.1 – Flow of GeocoderAPI

In Fixer process, Fixer tries to clear and re-structure the input address. In order to do that, it clears all extra spaces and special characters. Then all the sentences that given in parenthesis are clear because these items aren’t a part of the address structure and Geocoder doesn’t needed them for geocoding. Then address is converted to uppercase letters by using the casing rules in Turkish language. After that, Fixer needs to restructure the address in order to fix the defects that input address have and make address fields recognizable by the geocoder algorithm. In order to do that it uses predetermined keywords such as *“NO:”, “MAHALLESİ”, “CADDESİ”, “SOKAK”, “KÖYÜ”, “BULVARI”, “APT.”, “D:”* etc. Detailed explanation of Fixer process is given in Figure 3.2.

Figure 3.2 – Flow of Fixer process

In Tokenizer process, Tokenizer splits the address into tokens and stores it to an array. After that, it searches predefined keywords inside those tokens. It scans the array and tries to determine the address fields that input address has. It processes every item in the array one by one and tries to determine if the item is predefined keywords or not. If the item is not a predefined keyword, Tokenizer skips it and look for the next item. If it is a keyword, then Tokenizer looks for the previous items. It scans the array backwards until it hits another keyword or the boundaries of the array. Then copies these items into a containerand continues with the next item on the array. Containerholds the output items of this process. It stores the detailed version of the address. At the end, the unidentified items are returned as an output to process them later. Tokenizer process is shown in Figure 3.3.

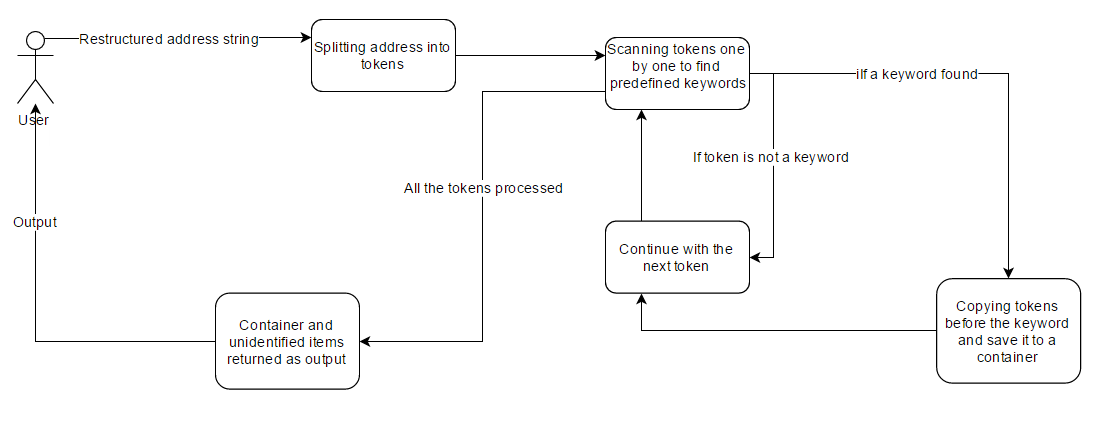


Figure 3.3 – Flow of Tokenizer process

In Geocoder process, Geocoder tries to find the exact location of the address if possible. The output of the tokenizer is the input of the geocoder. Geocoder creates a database connection in order to reach to address database. Geocoder tries to process the unidentified items first to determine if items have a city, town or POI information inside. After this step, geocoder starts with the top most element that tokenizer found and tries to minimize the search area. City information is the only mandatory field for Geocoder. Apart from city information, other fields can be empty. If city information is not provided by the input address or tokenizer can’t identify it, Geocoder will terminate its process and returns empty coordinates but returns the values that Tokenizer finds before. If city is found by the Geocoder, it stores the data into a container, set the detection level to city and continues with the next element. Then Geocoder tries to determine the town. If it founds town, it stores it into the container and set the detection level to town. After that Geocoder continues with the other items. System tries to find POI information at first because its location is already known and Geocoder doesn’t need to do extra searches to find location. If the POI information exists but Geocoder can’t find it in the reference database, it looks for the quarter item. If it exists, system searches POI with quarter information. If POI is found, detection level has been set to POI and the related information is stored into container. Then process is terminated and the container is returned as an output. If it is not found, Geocoder continues with the quarter information. If system finds the quarter, stores it into container, set the detection level to quarter and continues with the other items. The process continues like this to minimize the error rate and increase the success rate until the door number. The door number provided from the input address or identified by tokenizer may not be in our database. In this case, Geocoder looks for street or avenue. If one of these levels identified by tokenizer and found by geocoder, Geocoder gets all the door numbers that these levels have and tries to find the closest door number and stores it into the container, sets detection level to door and returns the container and exits the process. Geocoder process flow is shown in Figure 3.4.

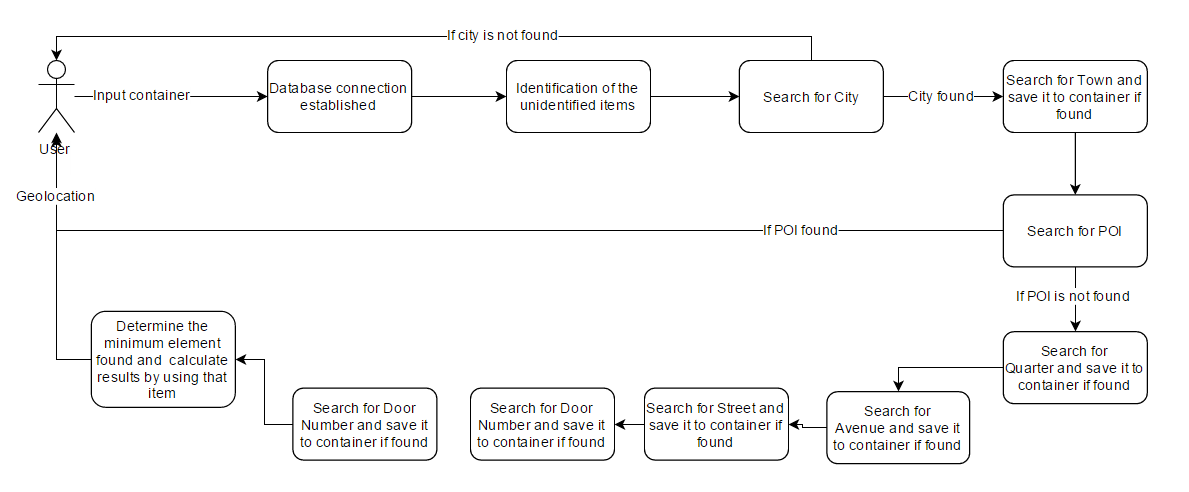


Figure 3.4 – Flow of Geocoder process

The Fixer, Tokenizer and Geocoder processes can be used individually or together in any combination. All processes start with an input given from user. User can start with Fixer and gets the restructured address as an output. User can directly uses Tokenizer to get the tokenized address as an output or can use the Geocoder to get the geolocation as an output. In GeocoderConfiguration part, a database connection is created by Entity Framework. Entity Framework uses the given configurations to create this database connection. ResultConverter removes system variables from the output objects of Geocoder and Tokenizer and returns the relevant data to users. Detailed explanation is given in Figure 3.5.

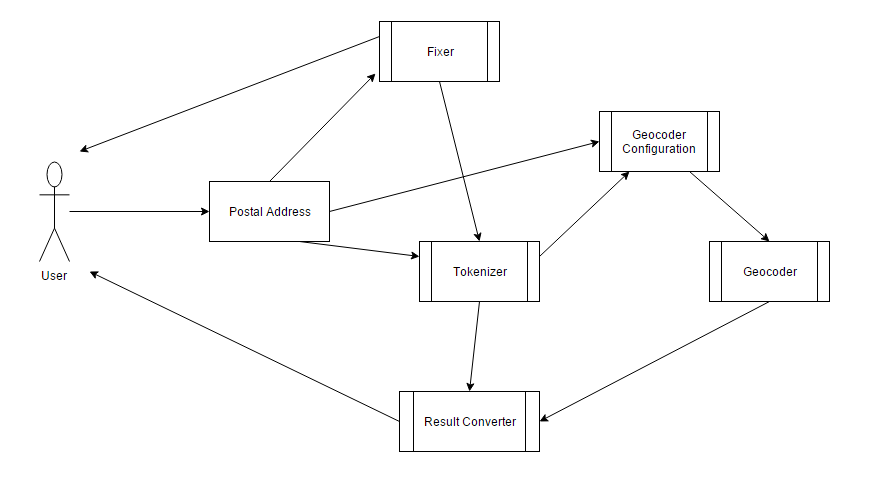
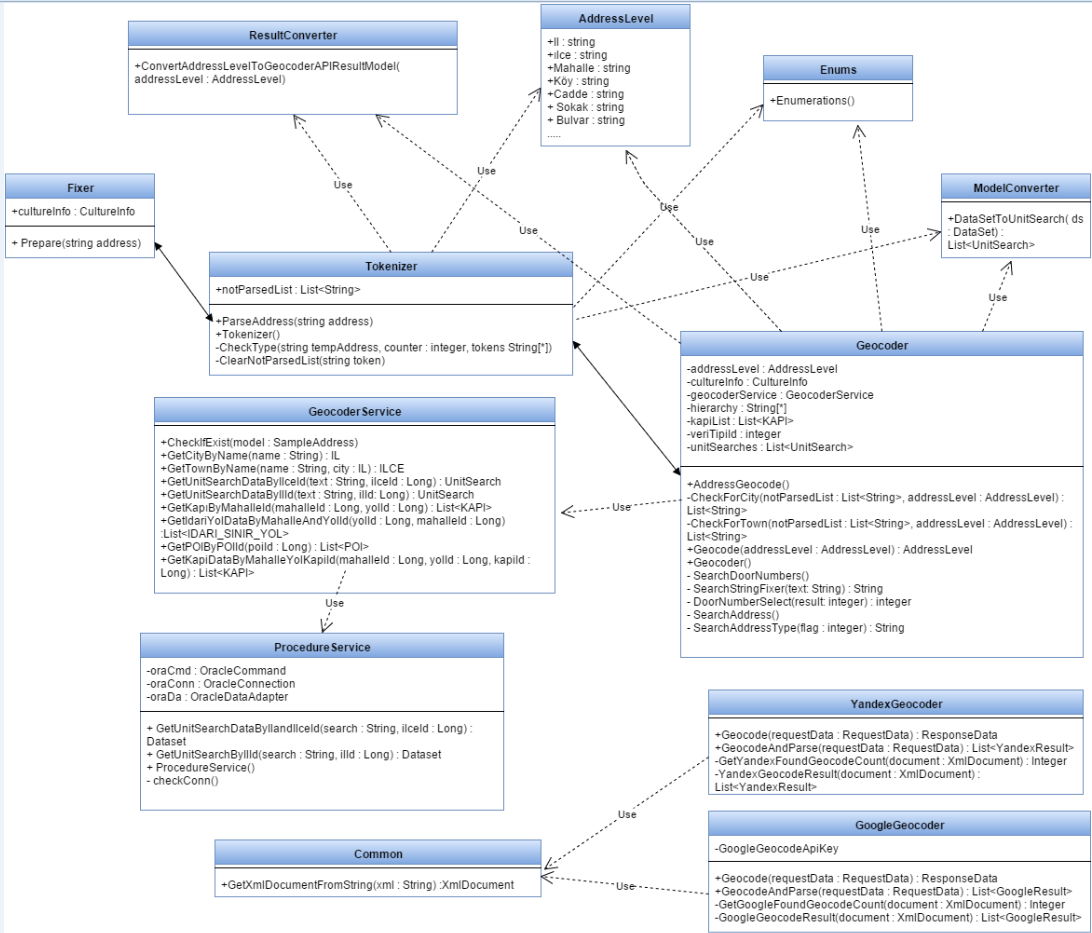


Figure 3.5 – The use case diagram

In Figure 3.6, Class diagram is given. Fixer, Tokenizer and Geocoder classes are the public classes that used to geocode a postal address. Fixer has a method called Prepare to clean and restructure the input address. The method ParseAddress is implemented in Tokenizer class in order to split the address into meaningful pieces. Geocoder has a Geocode method to search geolocation from the output of ParseAddress method.

GeocoderService class is implemented to make queries to the database from the application level. Geocoder class creates an instance of GeocoderService and uses its methods to search geolocation. Entity Framework and LINQ are used in GeocoderService class.

ProcedureService class holds the necessary Oracle procedure queries and implemented under the GeocoderService class. This class has attributes such as OracleConnection, OracleCommand, and OracleDataAdaptor to get the connection information and to query the database. ProcedureService does not use Entity Framework, it queries the database directly. This is because these procedures are querying a view, not a table.

Figure 3.6 – The class diagram

Enums class is a static class that holds the static values and predefined types like address types such as city, town or POI.

ModelConverter class is used to make conversions between objects. Needed object conversions are written in this class. This class is also a static class to make conversions faster.

In order to compare the result with other geocoding services, GoogleGeocoder and YandexGeocoder classes are implemented. These classes have methods to retrieve the result from Google and Yandex geocoding systems and to process it. Common class is used by these classes to make requests to their geocoding systems.

The state diagram of the geocoding process is given in Figure 3.7. Fixer, Tokenizer and Geocoder processes can be used individually and also can be used together.

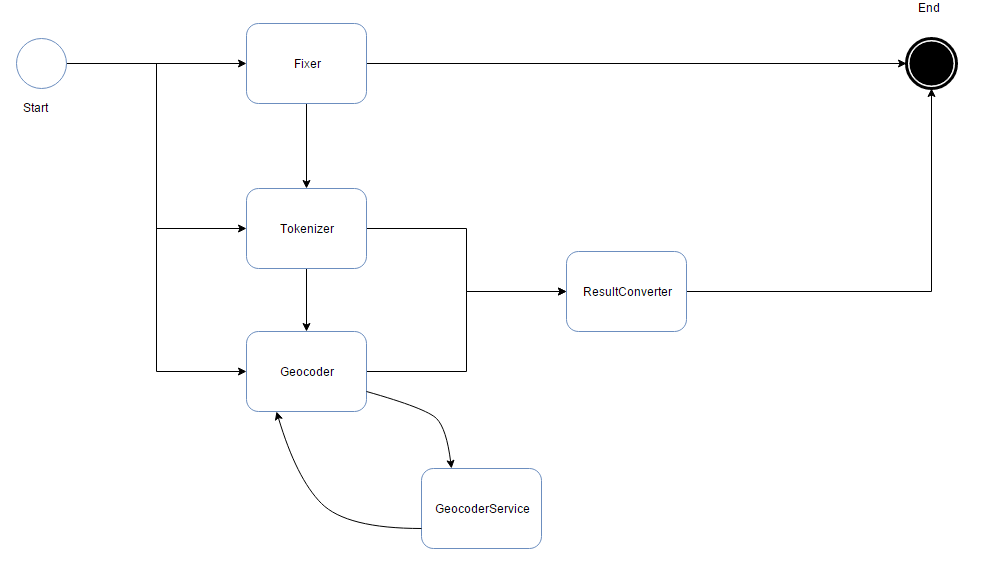


Figure 3.7 – The state diagram

Sequence diagram of GeocoderAPI is given in Figure 3.8. GeocoderAPI is called from an application. Fixer is called first. Fixer uses Prepare method to restructure the input address and returns it. Tokenizer is called after Fixer. Tokenizer uses PrepareAddress method to process the input address and it returns the output object. Output object passed to Geocoder. Geocoder processes the input object and returns the input object as output. ResultConverter is called at last. It converts the output object of Geocoder to another object called GeocoderAPIResultModel. ResultConverter returns the new object as an output.

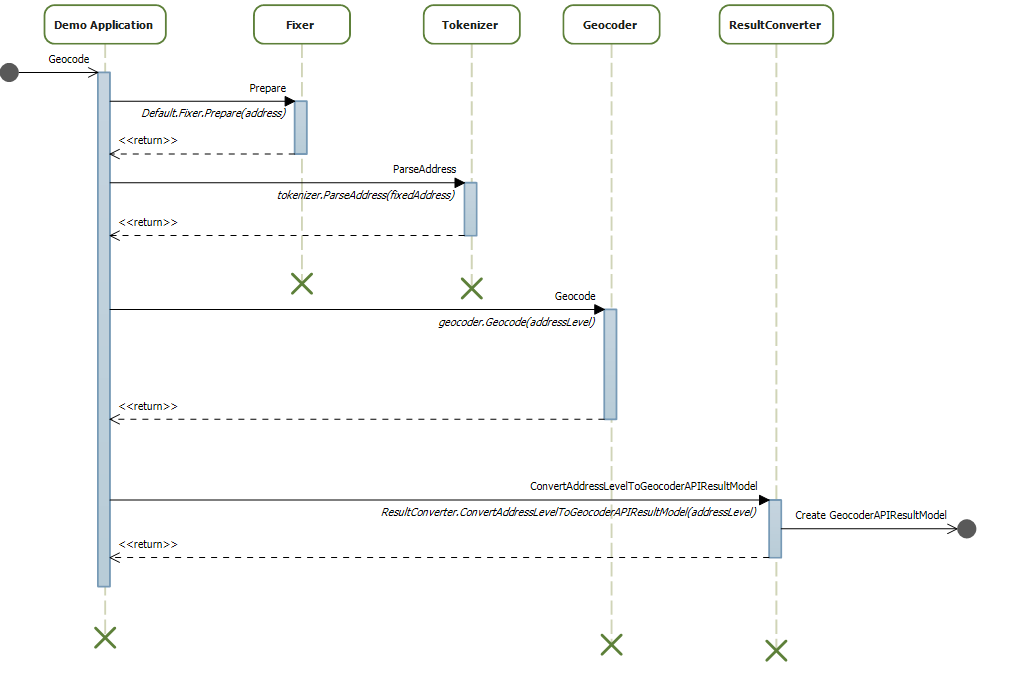


Figure 3.8 – The sequence diagram

# IMPLEMENTATION

This project is divided into two parts. First part is consists of Fixer, Tokenizer, and Geocoder processes. These processes use the input to search the geolocation. Second part is the Database side of this project.

## Fixer

The system asks user for an input. When the system gets the input postal address, it sends it to Fixer. Fixer gets the input address, clears the not necessary characters, spaces, fixes the defects that address have, restructures it and prepares address to the next step.

At first, Fixer cleans the extra spaces at the address. Fixer leaves only one space between the words. It starts with the first character in the input address and searches to end of it; this is shown in Figure 4.1.

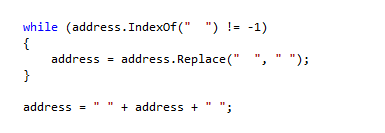


Figure 4.1 – Fixer clears the spaces

Then it converts it to uppercase letters by using the casing rules in Turkish language, this is shown in Figure 4.2. In order to do that, Fixer uses CultureInfo class. CultureInfo is a part of .NET Framework. It holds the language rules for every natural language. Fixer also uses ToUpper method to convert the given string to upper case. ToUpper is also is a .NET Framework item. ToUpper and CultureInfo can be used together to convert strings to upper case letters with respect to the language rules that provided. In our case, Fixer uses Turkish language rule set.



Figure 4.2 – Fixer clears the spaces and converts address to uppercase

After that, Fixer starts scanning the address and tries to replace the parts of the string to a known format that our algorithm recognizes; it is shown in Figure 4.3. It uses Replace method to convert the address to a restructured address. Replace is .NET Framework method. It takes two parameters; the string that user wants to replace and the new string that will be replaced.

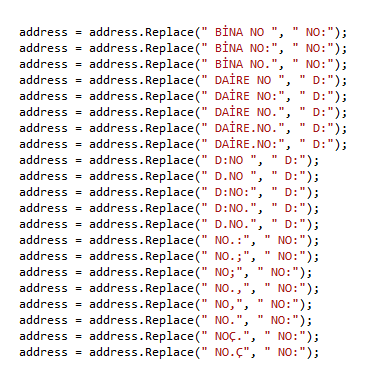


Figure 4.3 – Fixer restructures the address

At last, Fixer clears the parenthesis that input address contains; it is shown in Figure 4.4. .NET Framework’s IndexOf method is used to achieve that. IndexOf method searches the given string inside of another string and returns the position of the given string if exists.

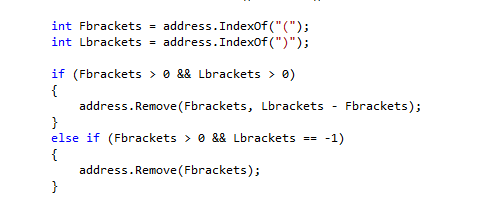


Figure 4.4 – Fixer clears the parenthesis

At the end of this process, the restructured address is returned to be used in the next process.

## Tokenizer

After restructuring the address, Tokenizer splits the address into its pieces, stores it as an array and starts processing the pieces by using the keywords that have been restructured by the Fixer. These keywords are shown in Figure 4.5.

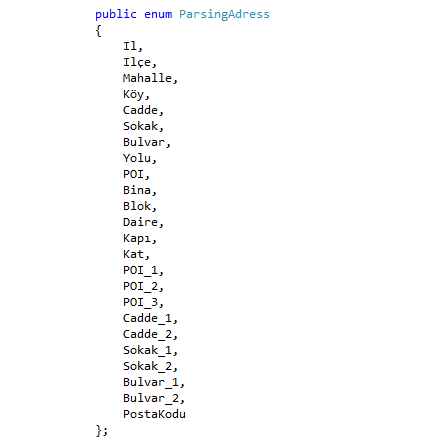


Figure 4.5 – List of the keywords

Tokenizer starts with splitting the address into pieces. In order to do that, Tokenizer uses spaces between the words; it is shown in Figure 4.6. It uses Split method that provided by .NET Framework. Split method splits strings into array by the given splitter element. Tokenizer splits the string by using the space special character. Regex is a .Net Framework Library and it has methods like Split to work with strings.

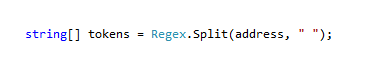


Figure 4.6 – Tokenizer splits the address into pieces

After splitting items, Tokenizer starts from the first item of the array and search for the keywords to the end of the array. When tokenizer recognizes a keyword, it starts moving backwards to the first item of the array until it finds another keyword or reaches to the boundaries of the array and takes the items that it passed and assigns it to a container called *AddressLevel* that is going to be the input of the geocoding process. *AddressLevel* is an object that holds the pieces of the input address. It is shown in Figure 4.7.

While processing these items, Tokenizer can’t identify some of them. These items stored in a list and returned as an output in order to process it later in Geocoder process.

At last, when system finishes processing the items, *AddressLevel* returned as an output and system exits.

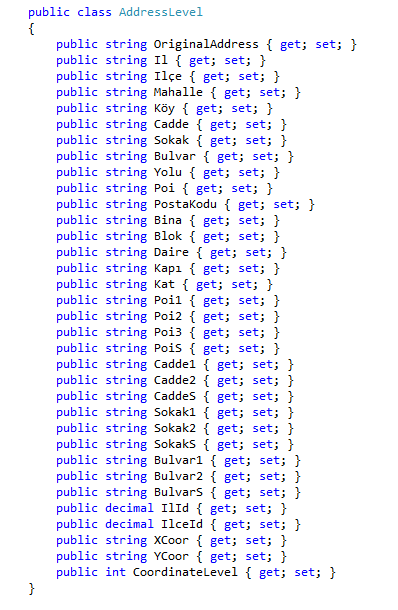


Figure 4.7 – AddressLevel object

## Geocoder

Geocoder is the main process of the system. This process can be used apart of the other processes to find geolocations. In this process, Geocoder starts with the top most element that found by the tokenizer and goes for the smallest element. In our address database, the data is stored in natural language and system compares these data with the processed address.

System first identifies the city information. City is mandatory because it is necessary to minimize the search area. If it doesn’t identify the city, Geocoder will terminate with empty coordinates but returns the fields that tokenizer found. Tokenizer can’t detect the city because city names are used in street, avenue or POI names in our country. Tokenizer returns these items as an output because of that. Geocoder tries to finds the city by searching these unidentified items and if it finds the city, it assigns it to *AddressLevel* object. *CheckForCity* method searches the given list of the string and tries to identify the city information. The method has two input parameters; *notParsedList* and *AddressLevel*. *notParsedList* holds the values of the unidentified items that provided from Tokenizer and *AddressLevel* is the output of Tokenizer. Method searches every single unidentified item in the address database. If it founds city information, it assigns its values to *AddressLevel* object and terminates its process. It is shown in Figure 4.8.

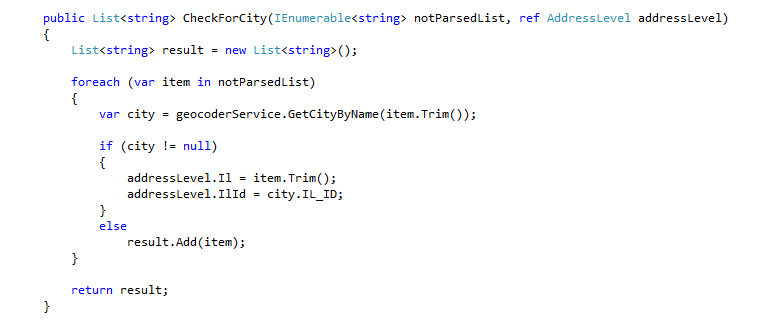


Figure 4.8 – Geocoder search method for city

After that, Geocoder continues with the next element which is town. Town names also used as a street, avenue or POI names, because of that Tokenizer can’t identify them. So, Geocoder tries to identify them. Town information is not necessary for searching geolocation but it minimizes the search field. Geocoder uses the city information and searches the town related to that city. *CheckForTown* method searches town information in the given list of the strings with the help of city information that found before. The method has two input parameters; *notParsedList* and *AddressLevel*. Method searches every single unidentified item in the address database. If it founds town information, it assigns its values to *AddressLevel* object and terminates its process. It is shown in Figure 4.9.

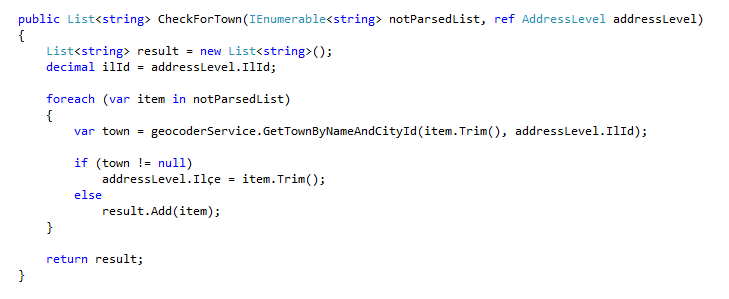


Figure 4.9 – Geocoder search method for town

After these two elements, system starts to scan left items and tries to find a relation between them. System’s priority is to find an exact coordinate of the address, so it tries to search POI’s under the found city and town. If POI is detected, system gets its coordinates and exits the processes. Otherwise, system continues with other items and tries to detect the coordinates of the items.

Remaining items are individually useless because these names are used in every city, town, POI or road in different places. Because of that, an Oracle procedure is created. This procedure collects every related data, processes the data, creates a relation with town, creates every probability and stores it as an Oracle view. This view is needed to create a connection between address levels town, quarter, avenue and street because users can enter incomplete addresses or a street or an avenue can be under more than two or more quarters. A piece of the procedure is given in Figure 4.10.

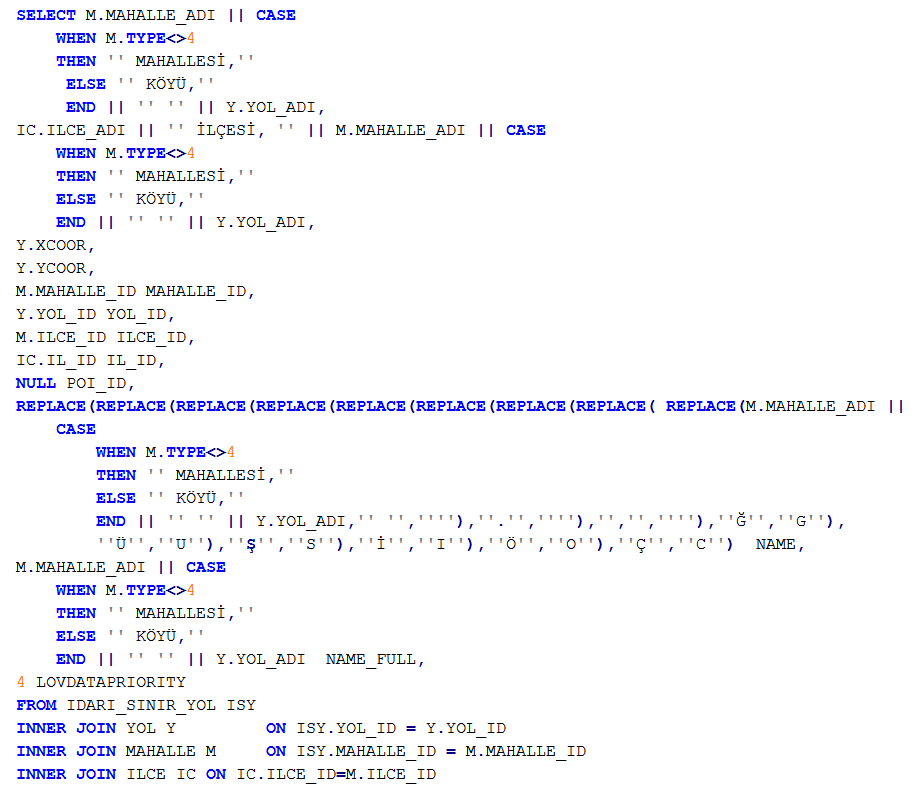


Figure 4.10 – Procedure collects quarter, road and town.

This part of the procedure is collecting quarter, road and town data into the view. With the help of this part, system can search the address that includes quarter, town or road. Geocoder gets town name first, then it gets road or quarter name and concatenate this items and send it to this view by using geocoding services.

After that, Geocoder tries to locate the door. The process of locating the door is tricky because our database doesn’t have all the door data in Turkey. Because of this, it has been decided to develop a door search system.

In Turkey, door numbers on a street or an avenue are sequential. One side of a road uses even numbers and the other side of the road uses odd numbers, it is shown in Figure 4.11. Geocoder has to find a street or an avenue to continue searching door information. In GeocoderAPI, if it doesn’t have the door number data, it looks to one upper level that found. Geocoder gets the entire door numbers that the street or avenue have. After that, it tries to find the closest door data that it has in the database.

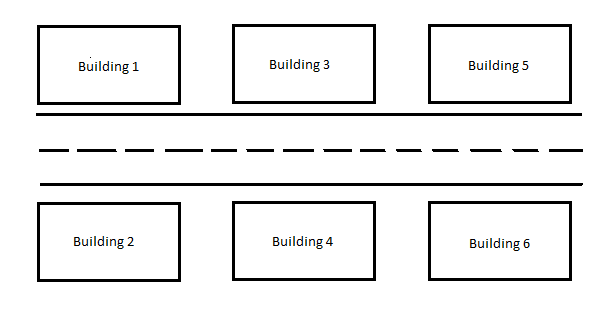


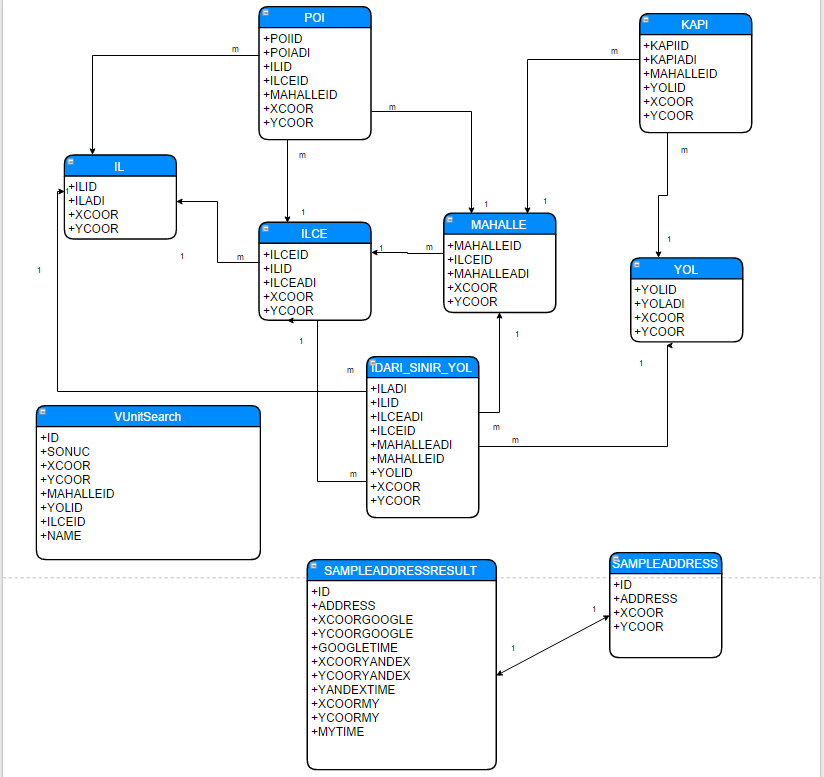
Figure 4.11 – Door numbering in Turkey

In order to find the closest door data, we look for the adjacent door information. If Geocoder searches door number 3 and if sample database doesn’t have its value, door search algorithm looks for door number 1 and 5. If a location related to this door numbers is found by the Geocoder, it sets detection level to door and assigns its information to AddressLevel object and terminates its process. If a location isn’t found, door search system looks for one more adjacent doors. After that, if a location is found, Geocoder terminates its process.

## Database

Geocoder API needs a database to store the locations data. In this project, Oracle 11g Express edition is chosen as a database because the reference address database is created in Oracle database. The reference database is given by the Infotect Company. Infotect is working on navigation systems. Infotect is collecting the data by using field teams all around Turkey.

Reference database consist of nine tables and one view. These tables are IL, ILCE, MAHALLE, YOL, POI, IDARI\_SINIR\_YOL and KAPI. ER diagram of this database is shown in Figure 4.12.

Figure 4.12 – ER Diagram of the reference database

IL table holds the all city information in Turkey. ILCE table holds all town information and it has many to one relation with IL table. All quarter information is stored at MAHALLE table and there is a many to one relation between MAHALLE and ILCE tables. All type of the roads such as street or avenue is stored at YOL table. KAPI table has door information and it has a many to one relation with MAHALLE table and many to one relation with YOL table. All POI information is stored at POI table and POI table has many to one relation with IL, ILCE and MAHALLE tables. IDARI\_SINIR\_YOL table is used to store the relation between roads and towns because a road can cross more than one town.

VUnitSearch is an Oracle View. It is consist of MAHALLE, YOL and ILCE tables.

VUnitSearch has every possible road, town and quarter combinations inside.

SAMPLEADDRESS table holds the test data. It has a one to one relationship with SAMPLEADDRESSRESULT table. SAMPLEADDRESSRESULT holds the test results. It has eleven fields inside; ID field for identification, ADDRESS field for the input address, XCOORGOOGLE and YCOORGOOGLE fields for geolocation result of Google geocoding service, XCOORYANDEX and YCOORYANDEX fields for geolocation result of Yandex geocoding service and XCOORMY and YCOORMY fields for geolocation result of GeocoderAPI.

Geocoder API is a database free application. The reference database format can be reformatted as a Microsoft SQL Server or any other RDMS style with minor configuration settings, Geocoder API can work without any modification on it. So Geocoder API doesn’t strictly bind to Oracle Database. In order to separate the application level and the database, system uses Entity Framework.

The connection between system and the database is established by the Entity Framework. Entity Framework is an ORM tool that connects applications and databases. It can work with every relational databases and gives freedom to programmers to focus on the algorithm.

Entity Framework uses configuration file to hold the necessary information about the database such as database type, database name, and connection information. With a small modification of this file, other databases can be reached by Entity Framework. Sample configuration is shown in Figure 4.13.

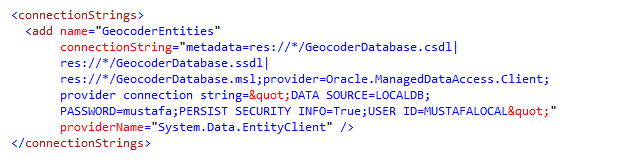


Figure 4.13 – Connection between Oracle Database and Entity Framework

Database tables are converted to an application level objects by the Entity Framework. This allows programmers to build application more secure and faster than usual because the database can directly be seen and reached inside the IDE. IL model created by Entity Framework is shown in Figure 4.14.

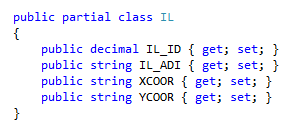


Figure 4.14 – IL object created by Entity Framework

In order to search the items, a service layer is implemented. It is placed between database and application level. This service layer holds every query that, GeocoderAPI needs and it is called GeocodingServiceLayer. It uses Entity Framework to connect the database and LINQ for query the database. LINQ allows programmers to write queries in programming languages. In Figure 4.15, a method called GetTown is implemented in order to get the town information from the database. LINQ uses keywords like normal SQL scripts uses such as WHERE, FIRST, ORDER DESC etc. Its syntax looks a little bit different from normal SQL scripts. In the given example, method takes two inputs, addressLevel and city and queries the ILCE table. geocoderEntities is the data access object created by Entity Framework. With the help of this data access object, service layer reaches the ILCE table and creates a query. In order to search the ILCE table, WHERE keyword is used. WHERE keyword lets programmers to filter the records. At GetTown method, service layer filters the ILCE records by using the cityId and town name.

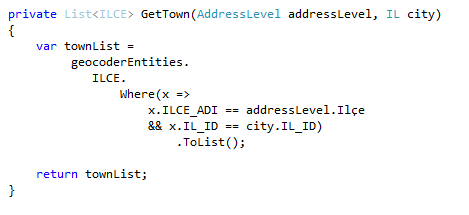


Figure 4.15 – LINQ query for town search

# EVALUATION

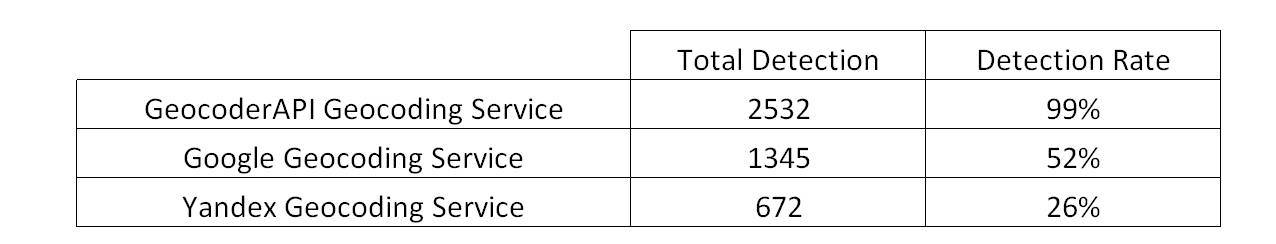
This project is tested with a sample address database. All the addresses in this database have exact geolocation information on it. The database has 2554 different addresses with geolocation information. This sample address database is given by Aras Kargo. Aras Kargo is a shipping company in Turkey. They obtain this addresses and locations by using hand terminals and their GPS systems.

Sample address database has a unique key column to identify the address, an address column to hold the postal address, a city name column, town name column, and geolocation coordinates columns.

Google and Yandex geocoding services are used in order to calculate the success rate of this project. Google and Yandex geocoding services may return more than one geolocation information. In this case, this result is marked as false because there must be only one geolocation information for an input address.

Geolocation information found by the services is round to have three digits after comma in order to compare the results of the services with the sample geocoding data.

Test cases of this project are geolocation detection rate, geolocation detection success rate, and geolocation search time performance. Detection rate is the geolocation information found by the geocoding services. Detection success rate is correct geocoding information found by these services. Search time performance is the overall time to search for geolocation.

Table 5.1 - Geocoding Detection Rate

Geocoding detection rates are given in Table 5.1. Google geocoding service returns only one geolocation information for 1345 input address. Yandex geocoding service detects 672 of the input addresses. GeocoderAPI detects nearly all the addresses and gives an output geolocation. 2532 address detection was made by GeocoderAPI service.

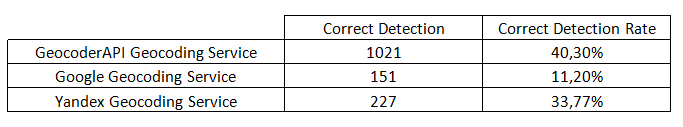


Table 5.2 – True Geocoding Detection Rate

Correct geocoding results are given in Table 5.2. Google geocoding service finds 151 exact geolocation from 1345 geolocation that it found. Its geolocation success rate is %11.2. Yandex geocoding service finds 227 exact geolocation out of 672 geolocations it found. It has a %33.77 success rate. Geocoder API finds 1021 exact geolocation from 2532 geolocation it finds before. %40.3 success rate is achieved.

GeocoderAPI is better in finding exact location. However, when it comes to search time performance, Google geocoding service and Yandex geocoding service are better. Google geocoding service returns a geolocation for an input address in 0.23 milliseconds. Yandex geocoding service returns a geolocation in 0.41 milliseconds. GeocoderAPI is searching a geolocation in 5.79 seconds. In term of search performance, Google geocoding service is better than Yandex geocoding service and GeocoderAPI. Detail information is given in Table 5.3.

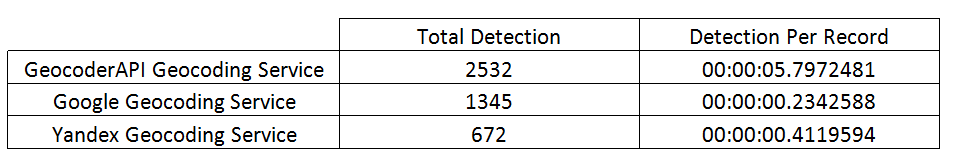


Table 5.3 –Geocoding Search Performance

# CONCLUSION

Geocoding has been on spotlight for researchers and scientists because geolocation gives a lot of answers to their questions. A lot of geolocation search service is built because of this reason. GeocoderAPI is built in order to improve the accuracy of the geocoding.

GeocoderAPI is better than Google geocoding service and Yandex geocoding service for finding exact geolocation in Turkey; however it can still be improved. Geolocation search success is related the keywords that GeocoderAPI is used in this project. If more keywords are supplied to GeocoderAPI, its success will be better.

Geolocation search success also can be improved by using self-learning algorithms. If it can be implemented to GeocoderAPI, geocoding success will be improved and also search time will be decreased.

GeocoderAPI does not use fuzzy match algorithms. Fuzzy match algorithms also improve the geolocation success if implemented.

In terms of search speed, GeocoderAPI is not efficient as Google geocoder service or Yandex geocoder services. GeocoderAPI is slow because it uses a relational database system and there is no search optimizer algorithms implemented in this project. However, this can be improved by using search algorithms and NoSQL databases.

This project can be used for route optimization, navigation, delivery optimization, and location based researches.

To sum up, this project aim to create a geolocation search API that is reliable and successful and from the test results, GeocoderAPI is more reliable and successful than Google geocoding service and Yandex geocoding service at finding exact geolocation.

# REFERENCES

[1] Google Geocoding Service, available at

[https://maps.googleapis.com/maps/api/geocode/xml?address={0}&key={1}](https://maps.googleapis.com/maps/api/geocode/xml?address=%7b0%7d&key=%7b1%7d)

(Accessed: 27 December 2015)

[2] Yandex Geocoding Service, available at

[https://geocode-maps.yandex.ru/1.x/?geocode={0}&lang=tr-TR](https://geocode-maps.yandex.ru/1.x/?geocode=%7b0%7d&lang=tr-TR)

(Accessed: 27 December 2015)

[3] Yandex Geocoding Service, available at

[http://dev.virtualearth.net/REST/v1/Locations?q={0}&key={1}](http://dev.virtualearth.net/REST/v1/Locations?q=%7b0%7d&key=%7b1%7d)

(Accessed: 27 December 2015)

[4] Google Maps, available at <https://www.google.com.tr/maps/> (Accessed: 27 December 2015)

[5] Yandex Maps, available at <https://harita.yandex.com.tr/> (Accessed: 27 December 2015)

[6] Bing Maps, available at <https://www.bing.com/maps/> (Accessed: 27 December 2015)

[7] Microsoft, available at <https://www.microsoft.com/tr-tr/> (Accessed: 27 December 2015)

[8] Oracle Database, available at <https://www.oracle.com/database/index.html> (Accessed: 27 December 2015)

[9] Microsoft SQL Server, available at

<https://www.microsoft.com/tr-tr/server-cloud/products/sql-server/> (Accessed: 27 December 2015)

[10] Microsoft SQL Server, available at <https://dev.mysql.com/downloads/mysql/> (Accessed: 27 December 2015)

[11] IBM DB2, available at <http://www-01.ibm.com/software/data/db2/> (Accessed: 27 December 2015)