BE”H

Mini project – Windows (.NET) course

Mini-project guidelines

Driving test management system, Ministry of Transportation

2019

Contents

INTRODUCTION **3**

GENERAL DESCRIPTION **5**

DESCRIPTION OF THE SYSTEM ENTITIES **6**

BRIEF PARTIAL DESCRIPTION OF THE PROJECT **7**

FIRST STAGE - DEFINING THE PROJECTS FOR THE LAYERS MODEL AND THEIR IMPLEMENTATION **8**

PART I - ENTITIES - THE BE LAYER 9

PART II – DAL FOR LIST IMPLEMENTATION 11

PART III - THE BL LAYER 13

STAGE TWO - THE UI LAYER **16**

STAGE 3 - UPDATE THE DAL LAYER **18**

REALIZATION USING LINQ-TO-XML 18

SUBMISSION DATES **20**

APPENDIX **21**

Introduction

Goal: develop software programming (= project) ability during the course.

Objective: Practice and implementation of

- The principles of the C# language
- Software architecture principles
- Creating user interfaces using the modern open infrastructure WPF
- Programmatic thinking and self-learning.

Note: The guidance in this document is only general, since part of the requirement is to use creative thinking. In addition, part of the final test will be based on programming topics that you will be required to deal with during the project development. It is important to emphasize that a grade below 85 in this course, as in any other practical course, is not significant to the employers in the industry ...

Highlights:

- The work will be done only in pairs (not a single student or three). The pairs will be fixed throughout the semester (it will not be possible to move from one group to another during the semester).
- Each partner must be a full partner at each of the stages. (If it turns out that one partner has done a certain stage and which other partner did the other part, the score will be proportional to the number of steps taken by each of the partners.)
- At the end of the semester, an outstanding project will be selected from each exercise group. Of these, the outstanding projects will be chosen, and their creators will win a prize. **The main measure of excellence is originality, self-learning, and of course correct programming**.

Submission process:

- The project is divided into three stages, which are based on one other.
- At the end of each stage, a date for submission and presentation of what has been done so far has been set.
- Testing the first two stages will be done in a general manner (mainly testing that the software runs properly, comments on the quality of the work, and meeting the exact schedule required). The grade on these two stages will not be presented to the students.

- Upon submission of the final and final stage, a thorough examination of the work will be carried out, which will also include the defense of the two partners on the work, each one separately.
- The final grade on the project will relate to all its components, as well as to the quality of the presentation, and to meeting the submission deadlines.
- Defense of the final stage will take place during the last class of the semester, and no later.
- The lab score will be 50% of the final score (composed of 10% homework and 40% project) and will be weighted with the test score only if the test score is at least 55.

Guidelines for presenting files in the model system:
At each step, **before the defense**, a .zip file with the solution to the model must be uploaded (as instructed in the introductory exercise):
The file name will be Project01_xxxx_yyyy_dotNet5779.
dotNet5779= The name of the course and the Hebrew year
01 = Stage name (01,02,03)
xxxx = last four digits of the identity number of the first partner
yyyy = last four digits of the identity number of the other partner
Please be careful to follow this format in order to prevent the failure to receive a score on a particular stage.

Note:
It is highly recommended to work with one of the GIT programs for version management and collaboration between the two partners.

General description

In this project we will write an (only partial) system for conducting driving tests for obtaining a driver's license ("tests") from the Ministry of Transportation.

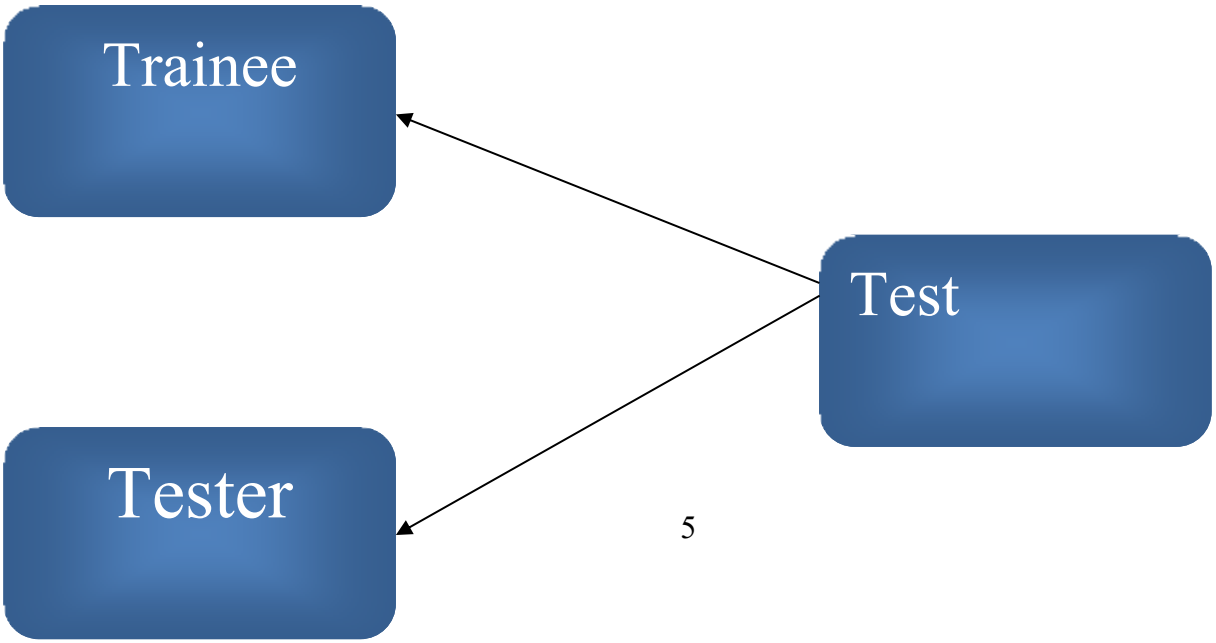
The test management system is a new and revolutionary idea proposed by the Ministry of Transportation as part of the privatization of the test system in an attempt to:

- Dramatically increase the availability of driving tests, and
- Make them easily accessible to the general public.

The new organization is managed by an interactive system, and contains several entities. We will simulate the system partially. Indeed, we will not yet write this project as a client/server web site (because that goes beyond the course content), we will write most of the layers in such a way that the majority will overlap with a parallel Internet system.

Any driving trainee (who meets certain criteria) can register for a driving test on a certain date and time that is convenient for him, and a convenient point of origin. The system assigns a tester to the trainee from the pool of testers at its disposal, in accordance with the availability of the tester.
Upon completion of the test, the tester updates the test data, and among other things, updates the origin and destination addresses, the examinee's score and relevant comments.
The organization has a huge nationwide 24 hours a day pool of testers.

A diagram describing the system's entities and the connections between them:



Description of the System Entities

Trainee

Contains details such as: ID number, family name, first name, date of birth, sex, telephone number, address (street, street number, city), the type of vehicle on which he has learned (private vehicle, two-wheeled vehicle, medium-sized truck, heavy truck), gearbox (manual, automatic), driving school name, driving instructor's name, number of driving lessons.

Tester

Contains details such as ID number, family name, first name, sex, cellular telephone number, specialty (private vehicle, two-wheeled vehicle, medium-sized truck, heavy truck; it is possible to assume that each tester has only one specialty) address, distance range from the address where he is willing to conduct tests.

Test

An entity that links the trainee to the tester and contains the following details: test number, ID number of the tester, ID number of the trainee, test date, test time, exit address, test criteria, test score, comments.

Brief partial description of the project

First stage

We will build the project according to the layers model.
The data source will appear as a separate project and will contain lists (List <T>) of objects.
Access to the data source will be done using the DAL layer.
The BL layer will contain the required logic,
And the UI layer will be a simple console interface.

Second stage:

Developing the user interface for a graphical visual interface.
The programming infrastructure required by the visual interface is WPF.

Third stage:

We change the DAL layer so that the data source is based on .xml files.
We'll also work with an external site that provides real distance values (Google Maps). We will download the data through an execution thread, and use it for our project.

First stage - defining the projects for the layers model and their implementation

In this part of the project, we will define each of the layers that will form the mini project that will characterize the Ministry of Transportation's test system.

This stage includes 4 parts as follows:

Part a - Definition of the classes for the entities, i.e. the BE layer.
Part b - Defining how to access the data source, i.e. the DAL layer.
Part c - Definition of the system logic, i.e. the BL layer.
Part d - Defining the UI of the mini-project, i.e. the UI layer.

Note: Any malfunction resulting from any action will result in exception handling according to the exception mechanism in C #. For example: entering an invalid identity number, trying to set up a test for a trainee who does not exist in the system. Exceptions must be executed for each layer separately, regardless of how this exception is handled in the other layers (e.g., if the system does not allow the UI layer to set up a test for a trainee that does not exist in the system, the DAL layer will not rely on that, but will also check that the trainee does exist).
Of course, the exceptions must also pass from layer to layer if necessary.

Part I - Entities - the BE layer

Below we will list fields that will be included in the classes. You **must** add additional fields and actions.
Note! Do not use public functions in classes (except ToString)
The main logic is performed in the BL layer, as will be specified below.

Define a **Class Library** type project called **BE** with the classes listed below.
Each class must be defined in a separate file.

In a separate file, you must define all the required enum types (e.g. type of vehicle (private vehicle, two-wheeled vehicle, medium truck, heavy truck), gear type (manual, automatic), sex, etc.).

In another separate file, **a class called Configuration** should be defined to include all global variables as static fields (see below: minimum number of lessons, maximum tester age, minimum trainee age, time range between tests, etc.)

*A s**tructure that describes an address containing***:
a. Street
b. Number of building on the street
c. City

A class called Tester that represents a tester and includes:
a. An attribute that specifies the ID number of the tester. (Must appear).
b. An attribute that specifies the surname of the tester.
c. An attribute that specifies the tester's first name.
d. An attribute that specifies the testers date of birth.
e. An attribute that specifies the gender of the tester.
f. An attribute that specifies the tester's phone number.
g. An attribute that specifies the tester's address
h. An attribute that specifies the tester's number of years of experience.
i An attribute that specifies the maximum possible number of tests per week.
j. An attribute that specifies the type of vehicle in which the tester specializes.
k. An attribute that specifies the working days of the week and the hours of the tester (Boolean matrix, round hours only, between 9:00 and 15:00, Sunday to Thursday).
l. An attribute of the maximum distance from his address, at which a tester can perform a test.

m. Additional attributes as needed.
n. *ToString*.

A class called *Trainee* that represents a driving trainee and includes:
a. An attribute that specifies the trainee's ID number. (Must appear)
b. An attribute that specifies the trainee's first name.
c. An attribute that specifies the trainee's last name.
d. An attribute that specifies the gender of the trainee.
e. An attribute that specifies the trainee's mobile phone number.
f. An attribute that specifies the address of the trainee
g. An attribute that specifies the trainee's date of birth.
h. An attribute that specifies the type of vehicle on which he has studied.
i. An attribute that specifies the type of gearbox that was learned.
j. An attribute that specifies the name of the driving school in which the trainee studied.
k. An attribute that specifies the name of the driving instructor with whom the trainee studied.
l. An attribute that specifies the number of driving lessons passed.
m. Additional attributes as needed.
n. *ToStrin*g.

A class called: *Test* that represents a driving test (i.e., the relationship between the tester and the trainee) and includes:
a. An attribute that specifies the test number. **(unique 8-digit sequential code)**
b. An attribute that specifies the tester's ID number (must appear)
c. An attribute that specifies the trainee's ID number (must appear)
d. An attribute that specifies the date set for the test.
e. An attribute that specifies the date and time of the test (it can be assumed that this is always a round hour, including the arrival of the tester)
f. An attribute that specifies the address from which to start the test.
g. Attributes that specify the different driving criteria that are examined during the driving test (maintaining distance, parking in reverse, using mirrors, signals, etc.), and success in them (passed/failed)
h. An attribute that specifies the test score (passed/failed).
i. An attribute that specifies the tester's comment.
j. *ToString*

Part II – DAL for list implementation

Below are details of possible actions. Additional actions **must** be added.

Add a class library project named DAL, and execute the following actions in it:

a. Configure an interface named Idal.
In the interface above, set up a signature of useful functions for the application such as:

• Add a test.
• Delete a test.
• Update existing test details.

• Adding a trainee.
• Deleting a trainee.
• Updating an existing trainee.

• Add a test
• Update test at the end

• Receiving a list of all testers.
• Receive a list of all trainees.
• Receive a list of all tests.

b. Create a new project called DS and set up a class called DataSource that will contain the data (lists) of our entities.
This class contains 3 static lists (<list>) of the classes located in BE.
(Note: This is a temporary class at this stage of the project. You can initialize the lists using input, but it is recommended to initialize the lists with several entries in the initializing code to make it easier.)

c. Define a class named: Dal_imp that implements the above Idal interface.
The functions of this class will work with the lists in the DataSource class.

Emphasis - Note!
Due to the need to separate the layers, the DAL layer will not allow the other layers to access the data source - that is, if necessary, this layer will send a copy of the data, not the data itself.

Note:
The DAL layer must verify that the ID number does not yet exist.
The DAL layer is the one that adds a "test" entity with a sequential test number.
In order to determine the initial test number, appropriate data must be stored in the Configuration class.

Part III - the BL layer

Set up an interface called: IBL where the methods have exactly the same signatures as in IDAL

The BL layer must enforce the following logic (of course, you must add your own logic, according to simple logic and requirements for your project):

- It is not possible to add a tester under the age of 40
- It is not possible to add a trainee younger than 18 years of age.
- It is not possible to add a test before 7 days have passed from the trainee's previous test (if any).
- It is not possible to add a test to a trainee who has done fewer than 20 lessons.
- It is not possible to add a test if no tester is available on the requested test date, provided that the tester was not assigned at the same time to another test. In this case, the system can offer the student an alternate time for the test.
- It is not possible to add a tester to the test if the tester has exceeded the number of weekly tests he can perform.
- A test cannot be updated unless all the fields that the tester must fill out are present.
- Two tests at the same time cannot be set up for the tester/trainee.
- It is not possible to set up a test on a type of vehicle for a student who has already passed a driving test on that type of vehicle.
- In setting up a test, the type of vehicle on which the trainee studied and the tester's specialty must match.

Note: All numeric data (minimum number of lessons, maximum tester age, minimum trainee age, test time interval, etc.) should be stored in static variables in the configuration class, so that they can be modified simply if a change in procedures is decided upon.

In addition, the BL layer should add the signatures of the following functions:

- A function that receives an address and returns all testers who live within a distance of X km from the address (the distance between addresses will be done by a WebRequest, which is accesses the Google Maps url in the third and final phase of the project. At this stage, set the distance by an appropriate random number).

• A function that receives a date and time and returns all testers who are available at that time. The function will check whether the date and time are the tester's working hours and whether the tester is giving another test at that time.
• A function that can return **all tests** that meet a specific condition (i.e. the function accepts a delegate that matches the functions that receive a test and returns bool, and thus the condition is defined).
• A function that a receives a trainee number, and returns the number of tests he took.
• A function that a receives a trainee number and returns if he is entitled to a driving license (if he has passed a driving test).
• A function that returns a list of all scheduled tests by day / month

Define functions that return the following groups (by using Grouping)
Each of the following functions will receive a Boolean variable that specifies whether to return the results in a sorted form (how to sort it at your discretion). The default value of this parameter is false (unsorted)
• A group of testers according to type of specialization.
• A list of trainees grouped according to the driving school in which they studied
• A list of trainees grouped according to the driving instructor with whom they studied
• A list of trainees grouped according to the number of tests they took

Define a new class in the BL layer that will implement the above IBL interface.

The implementation will include the use of Linq to object and Lambda expressions.

You must use your creative talent and add at least 6 more functions that fit into the BL layer and work on the data returned from the DAL.

Use at least 4 Linq expressions in the BL and DAL layer each - the use should be varied, including use of new, select. and let.
Use at least 4 lambda expressions in addition to the Linq expressions above.
As well as using anonymous delegates, and using predicates in FUNC.

Remarks:

This layer should ensure that basic logical rules are met, for example:
• If a particular test is added, then make sure that the trainee being tested exists.

• Make sure that trainees are only tested on the type of vehicle they have learned on.
• It is not possible that a trainee will fail in most of the criteria in the test, and nevertheless receive a passing grade.
• and so...

This is where the first submission will be made - you can use a console interface or a simple graphical interface.

To test things, we will create a temporary project called PL.
You can implement it either with a console application or a simple WPF GUI. It is possible at this stage to do everything in one window.
In any case you should call and check the functions found in BL.

Note:
To work in stages, we recommend that you first create something that checks the BE - then the functionality in the DAL to see that it works and then connect the BL.

In addition, proper documentation should be made:

- ✓ Over the IDAL and IBL intrapages functions using \\\
- ✓ Add documentation for implementations just above creative BL functions that enforce certain procedures you have invented such as "Find all testers from Jerusalem who specialize in private vehicles" in short, include documentation only for things that are not trivial, or things that use complex queries. Everything else is not necessary to document.
- ✓ Use meaningful names for variables.

Stage two - the UI layer

At this stage we will create a graphical interface for the project, using the development infrastructure WPF.

We will create a new project of type: WPF, called **PLWPF**.
Of course you will need to make an appropriate reference for it for BL and for BE.
You must define an entity named: BL of type: IBL
The Factory Method must be used for DAL and BL. Also, make sure that we do not create a new instance of the BL and the DAL each time.

Design the screens that will read and allow the user to access the functionality found in BL.
You must plan a screen for each entity, including at least the following screens:
• Add, update, delete a tester screen.
• Add, update, delete a trainee screen.
• Add, update, delete a test screen. When you want to add a "test", you must connect the trainee to one of the possible examiners according to relevant constraints such as: work days, distance.
• In addition - at least 3 other screens displaying queries that use BL.

The program will open on a main screen that references the various options.
Note: When there is a screen that allows you to add or update a value such as "Vehicle Type" and so on that already exist in the system, you must enable the selection of this field by a Combo Box control from which we will get a list from which to select.

Also, when we enable an update, after selecting from the list of entities to update, the key will appear on the screen and the other fields will be filled in automatically; the key value cannot be changed.
It is highly recommended that the record to be updated can also be searched for based on other data in the entity (e.g. name, surname, etc.) and not only based on the key (ID)
It is possible that add, update, and delete will actually be the same screen, as needed at that moment.

(Do not add functionality beyond basic CRUD [Create, Read, Update, Delete] to DAL)

You should also ensure that you insert rejection of exceptions in the appropriate places, and capture the exceptions so that the program does not "bomb" if a particular action fails at run time.

In this layer, you need to handle the thread that calls Google Maps, so that it executes from the object that calls it. This can be postponed to the third stage of the project.

After completing all the lessons on WPF studies, all WPF elements learned must be assimilated into the project. Where and how to implement this, depends on each and every one, according to his creativity.
Also, rejected and handled exceptions that belong to the UI layer itself must be handled, such as a user who has not filled in all required fields or entered characters into numeric fields etc.

Important note: Use:
Resources, data binding, converter, data context, dependency property, trigger, thread.

The second submission will be made here.

Stage 3 - Update the DAL layer

Realization using Linq-To-XML

Adding and implementing an additional class in DAL that implements the IDAL using Linq - to - XML:

You must add another class named Dal_XML_imp to the **DAL** project

Xml files should be prepared to replace the collections we have already created (i.e. replace the data source) for each entity, which will be written in the format corresponding to the structure of the entity it represents. These files will be in a separate folder in the solution.

You need to create the initializations, the option to save and upload a file, and also to query by Linq query.
All IDL interface methods must then be implemented.

For working with local XML files, you can use the following tutorial:

For initializing the XML files:
for example:
studentRoot = new XElement ("students");
Now we can add / download and update elements in the file
If the file already exists, we simply will not create it. The code would look something like this:

```
public XmlSample()         {
            if (!File.Exists(FPath))
                CreateFiles();
        }

        private void CreateFiles()
        {
            studentRoot = new XElement("students");
        `
```

For the sequential number of some of the classes and some settings:

The problem:
When we used the sequence number we used a static variable that grew in the course of the program.
Now that we save the data in xml the next time we open the program the variable will reset
itself and then the sequential number will start from 0.

The solution:
We will define an XML file named config.xml and there we will define the sequential number and other settings we want.
Each time we save an object that uses a sequential number we will also update the config.xml file
The first time the dal is created it will make sure to update it in the class.

Note:
For the config file and one more file for one of the classes, use linq to xml for all add, update, delete, and retrieve operations.
In the implementation of the other classes serialize can be used, that is to work with the data in the list and ultimately save it to XML using xmlSerialize

Here the final defense will take place.

Good luck!

Submission dates

First submission (parts 1-3 with basic UI) Submission date: by 3 Tevet
Second submission: (part 4 UI with WPF) Submission date: by 23 Tevet
Third submission: After stage 3 – final - **Submission date**: last meeting of semester (by 9 Shvat)

Defense in the last week during lab hours. After the end of the semester, no inspection of student work will take place.

Appendix

An example of calculating the distance and duration of travel between two addresses using Google Maps

Below is an example of an attempt to calculate the distance and duration of travel between two addresses. The example includes:

1. Construct a string in a certain format containing the address (English or Hebrew)
2. Requesting a Google service with a Web request (WebRequest)
3. Receive a response in XML format
4. Analysis of the answer, according to the following options:
   a. No answer was received, apparently because of a network overload.
   b. An answer was received, but at least one of the addresses was incorrect.
   c. An answer was received, which includes distance and duration of travel between the two addresses.

Because the network is sometimes overloaded, a network request can take several seconds and only respond after several attempts.
Therefore, you must wrap the one experiment that is demonstrated in this file in a **BackgroundWorker** thread.
The thread will continue to attempt network requests until a response is received.
The number of references to Google's free map service is limited, and anyone who exceeds it is marked as a bothersome customer and stops receiving answers.
The recommendation is to perform a delay of at least 2 seconds within the thread between one network request and another, and not to make more than 2500 calls per day.
If you have exceeded the limit per day, then you may be able to return after 24 hours. But you never know ...
Use the Google Maps service wisely.

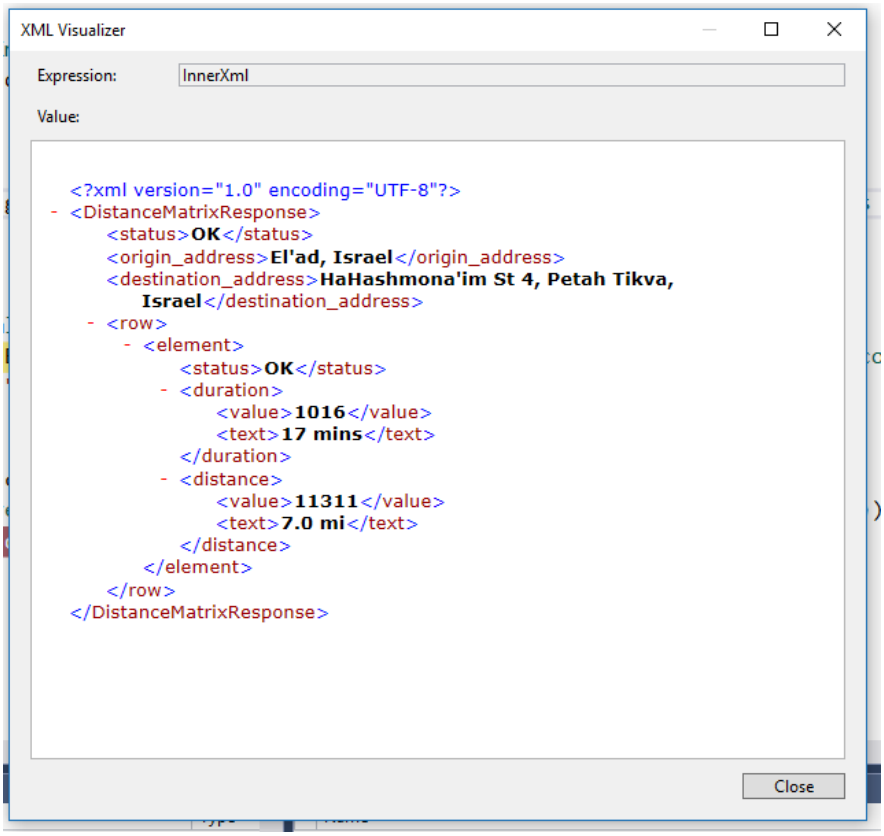
Regarding the project:
If, for any reason, you have not received a response regarding the distance between 2 addresses, please use any default value you set.

Example of code for one attempt to calculate distance and travel time between two addresses:

```
using System;
using System.IO;
using System.Net;
using System.Xml;

string origin = "pisga 45 st. jerusalem"; //or "אחד העם 100 פתח תקווה" etc.
string destination = "gilgal 78 st. ramat-gan";//or "ז'בוטינסקי 10 רמת גן" etc.

//build the url that includes the 2 addresses
string url = @"http://maps.googleapis.com/maps/api/distancematrix/xml?origins=" +
origin + "&destinations=" + destination +
"&mode=driving&sensor=false&language=en-EN&units=imperial";

//request from google service the distance between the 2 addresses
HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);

WebResponse response = request.GetResponse();

Stream dataStream = response.GetResponseStream();
StreamReader sreader = new StreamReader(dataStream);
string responsereader = sreader.ReadToEnd();
response.Close();

//the response is given in an XML format
XmlDocument xmldoc = new XmlDocument();
xmldoc.LoadXml(responsereader);

if (xmldoc.GetElementsByTagName("status")[0].ChildNodes[0].InnerText == "OK")
//we have an answer
{
   if (xmldoc.GetElementsByTagName("status")[1].ChildNodes[0].InnerText == "NOT_FOUND")
   //one of the addresses is not found
      Console.WriteLine("one of the addresses is not found");
   else
   {//2 of the addresses are OK
      //display the returned distance
      XmlNodeList distance = xmldoc.GetElementsByTagName("distance");
      double dist = Convert.ToDouble(distance[0].ChildNodes[1].InnerText.Replace(" mi", ""));
      Console.WriteLine(dist * 1.609344); //each mile is 1.609344 kilometer
```

```
        //display the returned duration
        XmlNodeList duration = xmldoc.GetElementsByTagName("duration");
        string dur = duration[0].ChildNodes[1].InnerText;
        Console.WriteLine(dur);
    }
}
else
//we have no answer, the web is busy, the waiting time for answer is limited
(QUERY_OVER_LIMIT), we should try again (at least 2 seconds between 2 requests)
{
    Console.WriteLine("We didn't get an answer, maybe the net is busy...");
}
```

Example of XML which is received if a valid answer is received of distance and travelling tome between 2 addresses:



Example of XML which is received if an answer is received but one of the addresses is invalid: